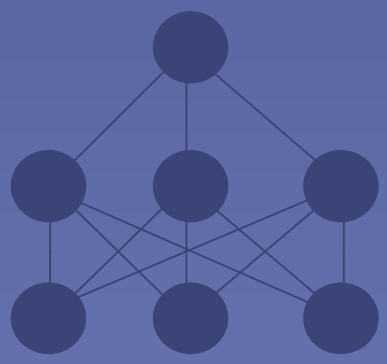
Telegram: @Farabar_BI

www.farabar.net

یادگیری ماشین



نوشته ی Tom M. Mitchell

مترجم: محمّد نخبه زعیم

www.farabar.net

یادگیری ماشین

نوشته ی Tom M. Mitchell

ترجمه ی محمّد نخبه زعیم

Telegram: @Farabar_BI
www.farabar.net

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

پیشگفتار مترجم

با سلام، کتابی که پیش روی شماست شامل الگوریتم ها، روش ها و نکات مربوطه ی "یادگیری ماشین" است. کتاب طوری نوشته شده است که قابل خواندن برای تمامی دانشجویان با هر میزان اطلاعات اولیه باشد. خلاصه می کنم امیدوارم از خواندن کتاب لذت ببرید و البته چیزهایی مفیدی یاد بگیرید. برای من که بسیار مفید بود!

در ترجمه ی کتاب سعی شده تا خواندن متن برای خواند راحتر باشد، و گاهی بعضی جملات و کلمات و پاراگراف ها (!) تغییر کرده اند. اگر هر گونه اشتباه غلت املایی یا اشتباه نگارشی در متن دیدید مرا ببخشید و آنرا به من اطلاع دهید.

تابستان ۸۹

پی نوشت:

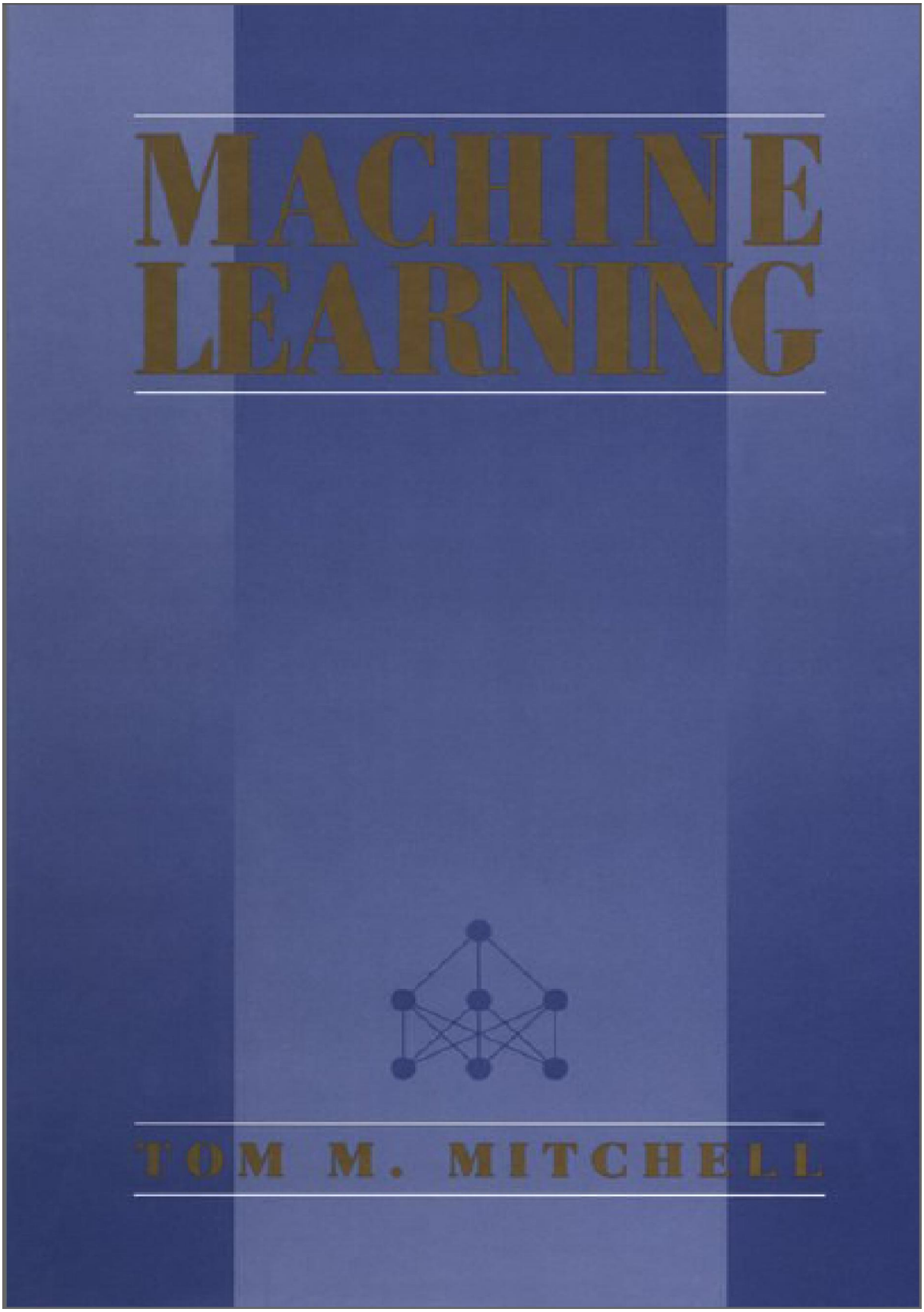
تاریخ آخرین اصلاحات اعمالی: مهر ۹۱

Mail: nokhbeh100@gmail.com

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net



www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

پیشگفتار

چگونه می توان به ماشین یاد داد؟!

در مبحث یادگیری ماشین[1] همواره با این سوال درگیر هستیم که "چگونه" می توان ساختاری برای برنامه های کامپیوتری طراحی کرد که بتوانند با استفاده از آزمایشات متعدد بر تجربیات (مهارت) خود بیفزایند. امروزه کاربرد یاددهی به سیستم ها، در عرصه های گوناگون گسترش یافته است، برای مثال نرم افزار های کاوش داده[2] ای ایجاد شده که می توانند در برابر حملات و سرقت های اینترنتی مقابله کنند، سیستم های اطلاعاتی ای که می توانند علاقه ی هر فرد به انوع اطلاعات را مشخص کنند و یا حتی خودرو های اتوماتیک که می توانند یاد بگیرند چگونه بدون راننده در خیابان، رانندگی کنند! این در حالی است که این علم با سرعت بسیار زیاد در حال پیشرفت و تکامل است.

هدف این نوشته معرفی راه حل ها و الگوریتم های کلیدی تشکیل دهنده ی هسته ی یادگیری ماشین به خواننده است. باید گفت در این راه سعی خواهیم کرد که از دانش های گوناگون نظیر آمار[3]، هوش مصنوعی[4]، فلسفه[5]، تئوری اطلاعات[6]، بیولوژی[7]، cognitive science، پیچیدگی محاسباتی[8] و تئوری کنترل[9] در رسیدن به این هدف کمک بگیریم. از نظر ما، بهترین راه برای آموختن یادگیری ماشین نزدیک شدن به مطالب، از تمامی وجوه و مفاهیم آن است. در گذشته این کار بخاطر نبود اطلاعات اولیه در تمامی جنبه ها در یک جا سخت به نظر می رسید. هدف اولیه ی این کتاب ارائه ی چنین اطلاعاتی در کنار هم است.

---

[1] Machine Learning
[2] Data-mining Softwares
[3] statistics
[4] Artificial intelligence
[5] philosophy
[6] Information theory
[7] bilogy
[8] Computational complexity
[9] Control theory

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

بخاطر همین بی نظمی طبیعی مطلب، این کتاب فرضهای خیلی کمی در مورد اطلاعات اولیه ی خوانند انجام می دهد. در عوض مفاهیم اولیه ی آمار، هوش مصنوعی، تئوری اطلاعات و دیگر علوم در صورت نیاز با تمرکز بر قسمت مورد نیاز توضیح داده می شوند. این کتاب می تواند هم توسط دانشجویان دوره ی کارشناسی و هم توسط دانشجویان دوره ی ارشد رشته های علوم کامپیوتر، مهندسی کامپیوتر، آمار و ... مطالعه شود. همچنین این کتاب می توان به عنوان مرجع برای برنامه نویسان حرفه ای و غیر حرفه ای باشد. در نوشتن این کتاب دو نکته رعایت شده است: ابتدا اینکه کتاب باید برای دانشجویان دوره ی کارشناسی قابل خواندن باشد و همچنین مطالبی که پیشنیاز شروع دوره ی دکترای یادگیری ماشین است را در بر گیرد.

سومین نکته ای که در نوشتن کتاب رعایت شده است این است که باید تعادلی از کاربرد و تئوری در آن حضور داشته باشد. تئوری یادگیری ماشین سعی دارد به سوالاتی نظیر "با افزایش نمونه های آموزشی کارایی یادگیر چگونه تغییر می کند؟" و "کدام الگوریتم های یادگیری برای کدام کارهای یادگیری مناسب اند؟" پاسخ دهد. این کتاب شامل بحث هایی در این موارد و دیگر موارد مربوطه به علاوه ی پایه های آماری، پیچیدگی محاسباتی و بررسی بیزی است. تمرین یادگیری ماشین با ارائه ی الگوریتم های اصلی این زمینه و دنبال کردن عمکرد آنها انجام می گیرد. داده های واقعی و بسیاری از الگوریتم ها در http://www.cs.cmu.edu/~tom/mlbook.html آورده شده است. کد های شبکه های عصبی، داده های بازشناختی[1] ، کد های یادگیری درختی، داده های بررسی های وام گیرندگان، کد های دسته بندی کننده ی های بیز و داده های بررسی های متون در آنجا آورده شده اند.

[1] recognition

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

فصل اول: مقدمه

از زمانی که رایانه‌ها ساخته شده‌اند، انسان‌ها همواره به دنبال راهی بوده تا بتوانند آن‌ها را برای مقاصد دلخواهشان، آموزش دهند تا شاید بتوانند روزی آن‌ها را طوری برنامه ریزی کنند که بتوانند خودشان با گذر از آزمایشات، بر تجربه‌ی خود بیفزایند و هوشمند شوند. می‌توان روزی را تصور کرد که رایانه‌ها می‌توانند از روی داده های درمانی نحوه‌ی تشخیص بیماری و روش درمان مؤثرتر را پیدا کنند؛ در ساختمان‌ها در اثر گذشت زمان و با در نظر گرفتن داده های انرژی، بهینه‌ترین برنامه‌ی انرژی را برای ساختمان تنظیم کنند؛ در نرم افزار های شخصی، با توجه به سلیقه‌تان، برنامه‌ی مورد نظر را برایتان پیشنهاد دهند. در واقع با موفقیت در آموزش صحیح به رایانه‌ها، دروازه های جدیدی از زندگی برای انسان‌ها، باز خواهد شد. همچنین پیشرفت بهتری در زمینه‌ی الگوریتم‌های تجزیه و تحلیل اطلاعات، به ما کمک خواهد کرد تا توانایی‌های انسان (یا حتی محدودیت‌های آن را!) بهتر دریابیم.

در حال حاضر، ما دقیقاً نمی‌دانیم چگونه باید رایانه‌ها را برنامه ریزی کنیم تا به خوبی انسان‌ها یاد بگیرند. هر چند که روش‌هایی که تاکنون کشف شده‌اند، برای اهدافی خاص، بسیار موثر عمل می‌کنند اما برای تمامی اهداف مناسب نیستند. برای مثال در کاوش اطلاعات[1]، استفاده از الگوریتم‌های یاددهی به ماشین، بسیار متداول است. حتی در زمینه‌هایی با داده‌ها سر و کار دارند، این الگوریتم‌ها بسیار بیش از حد انتظار عمل کرده و جواب داده‌اند. به عنوان مثال در مسائلی مانند شناسایی گفتار[2]، الگوریتم‌های مبتنی بر یادگیری ماشین، بسیار بهتر از سایر روش‌ها، جواب داده‌اند. ظاهراً به نظر می‌رسد دانش ما از رایانه‌ها، رفته رفته، به بلوغ می‌رسد. به جرات می‌توان گفت، مبحث یاد دهی به ماشین، نقشی به شدت پررنگ در زمینه‌ی علوم کامپیوتر و تکنولوژی کامپیوتری بازی می‌کند.

دستاوردهایی در این زمینه بدست آمده است: برنامه‌هایی نوشته شده‌اند که یاد می‌گیرند که صدای کلمات را تشخیص دهند (Waibel 1989; Lee 1989)، ضریب بهبود بیماران ذات‌الریه را پیش بینی کنند (Cooper 1997)، کلاه برداری با کارت اعتباری را تشخیص دهند، اتومبیل‌ها را در بزرگراه هدایت کنند (Pomerleau 1989) و بازی‌هایی مثل تخت نرد[3] را در حد انسان‌های ماهر بازی کنند

[1] Data Mining
[2] Speech Recognition
[3] Backgammon

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

(Tesauro 1992,1995). نتیجه های تئوری‌ای بدست آمده که روابط پایه ای بین تعداد نمونه های آموزشی مشاهده شده و تعداد فرضیه های ممکن و امید میزان خطا در فرضیه‌ها را مشخص می‌کنند. آدمی در عصر حاضر کم کم به مدل‌های اولیه‌ی یادگیری انسان و حیوان پی می‌برد و کم کم رابطه‌ی الگوریتم‌های یادگیری کامپیوتری را با این مدل‌ها پیدا می‌کند (Lair 1986; Anderson 1991; Qin 1992; Chi and Bassock 1989; Ahn and Brewer 1993). در عمل نیز، در دهه های اخیر الگوریتم‌ها، تئوری و تحقیقات بر روی سیستم‌های زیستی یادگیری پیشرفت قابل توجهی کرده‌اند. خلاصه‌ی تعداد بسیاری از پروژه های یادگیری ماشین در جدول ۱.۱ آمده است. Langley and Simon (1995) و Rumelhart (1994) کاربرد های دیگری را در یادگیری ماشین تحقیق کردند.

لذا در این نوشته ما سعی خواهیم کرد، مباحث ، الگوریتم‌های یادگیری ، نتایج نظری و کاربرد های آن‌ها را مورد بررسی قرار دهیم. به دلیل ویژگی ذاتی این مبحث در ارتباط آن با رشته‌ها و زمینه های گوناگون، مانند مباحث هوش مصنوعی، آمار و احتمالات، هندسه محاسباتی، تئوری کنترل،تئوری اطلاعات، فلسفه، روان شناسی، عصب شناسی و ... ، هر جا که لازم باشد، مباحث را در حد نیاز بررسی خواهیم کرد. جدول ۱.۲ ایده های اصلی که یاددهی به ماشین با علوم مختلف دیگر دارد را به صورت خلاصه بیان کرده است. از آنجایی که هدف از این کتاب بکار گیری نتیجه های بدست آمده از این تحقیقات است، لازم نیست خواننده در این زمینه‌ها حرفه ای باشد. نکته های کلیدی این زمینه‌ها معمولاً با زبانی ساده بیان شده است و جملات و عبارات نا آشنا نیز تعریف خواهند شد.

۱.۱ مسائل یادگیری خوش وضع[1]

بیایید مطالعه‌ی یادگیری ماشین را با معرفی چند عمل یادگیری شروع کنیم. ابتدا مفهوم یادگیری[2] را به فرمی تعریف می‌کنیم که هرگونه برنامه کامپیوتری که کارایی‌اش در کار خاصی با تجربه بهبود یابد را در بر گیرد. به عبارت دقیق‌تر،

تعریف: زمانی گفته می‌شود که یک برنامه‌ی کامپیوتری از تجربه‌ی[3] E در مورد کار[4] T بر حسب معیار کارایی[5] P یادگیری دارد که کارایی‌اش بعد از تجربه‌ی E برای کار T بهبود بیابد.

برای مثال، برنامه‌ی کامپیوتری‌ای که یاد می‌گیرد تا چکرز[6] بازی کند می‌تواند کارایی خود را که با "توانایی بردن" معلوم می‌گردد، اعمال ممکن بازی چکرز را اعمال ممکن و تجربه ای که از بازی در مقابل خودش بدست می‌آورد را تجربه در نظر گرفت. در کل، برای اینکه مسئله، مسئله ای خوش وضع باشد، باید ویژگی‌های روبرو را برای آن معلوم کنیم: مجموعه‌ی اعمال ممکن، کارایی‌ای که باید بهبود یابد و منبع تجربیات.

مسئله‌ی یادگیری بازی چکرز:

- عمل T: بازی کردن چکرز.

---

[1] well-posed
[2] learning
[3] experience
[4] task
[5] performance
[6] checkers

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- کارایی P: درصد بازی‌های برده در مقابل حریف.
- تجربیات آموزشی E: بازی تمرینی در مقابل خودش.

به همین منوال می‌توان مسئله های یادگیری خوش وضع بسیاری را نظیر مسئله‌هایی چون یادگیری تشخیص دستخط[1] و یا یادگیری هدایت یک اتومبیل مشخص کرد.

مسئله‌ی یادگیری تشخیص دستخط:

- عمل T: تشخیص و دسته بندی کلمات دست نویس در تصاویر
- کارایی P: درصد کلماتی که درست دسته بندی شده‌اند
- تجربیات آموزشی E: پایگاه داده ای از کلمات دست نویس با دسته بندی‌هایشان.

مسئله‌ی یادگیری هدایت یک اتومبیل:

- عمل T: هدایت اتومبیل در آزاد راه با استفاده از دوربین‌های نصب شده
- کارایی P: میزان طولی که بدون خطا اتومبیل هدایت شده (خطا ممکن است توسط عامل انسانی تشخیص داده شود)
- تجربیات آموزشی E: مجموعه ای از دستور های هدایت و عکس‌های مربوطه‌ی دوربین‌ها در زمان هدایت اتومبیل توسط انسان

- یادگیری تشخیص صوت کلمات[2]

تقریباً همه‌ی سیستم‌های موفق تشخیص گفتار[3] از یادگیری ماشین به نحوی استفاده می‌کنند. برای مثال، سیستم (Lee Sphinx 1989) استراتژی تشخیص صداهای اولیه[4] و کلمات را از سیگنال‌های مشاهده شده یاد می‌گیرد. متد های شبکه های عصبی (Waibel 1989) و متد های یادگیری مدل‌های پنهان مارکوو (Lee 1989) برای تغییر سیستم برای حساس بودن به افراد مختلف، فرهنگ لغات مختلف، میکروفون‌های مختلف، صدا با نویز و ... موثر است. متد های مشابهی کاربرد مشابهی در بسیاری از سیستم‌های تفسیر سیگنال[5] دارند.

- یادگیری هدایت یک اتومبیل.

متد های یادگیری ماشین در آموزش اتومبیل‌های خودکار[6] در انواع جاده‌ها و خیابان‌ها به درستی به کار رفته‌اند. برای مثال، سیستم ALVINN (Pomereau 1989) برای هدایت اتومبیل در سرعت ۷۰ مایل بر ساعت و طول ۹۰ مایل در میان اتومبیل‌های دیگر به درستی عمل کرده است. تکنیک‌های مشابهی کاربردهای احتمالی در بسیاری از مسائل حسگری[7] دارند.

- یادگیری دسته بندی ساختار های نجومی جدید.

یادگیری ماشین در پایگاه داده های بزرگ مختلفی برای یادگیری نظم‌های کلی به کار رفته‌اند. برای مثال، الگوریتم‌های درخت یادگیری در ناسا[1] برای یادگیری چگونگی دسته بندی اشیاء اسمانی در تحقیق (second Palomar Observatory Sky

[1] recogonize handwritten words
[2] spoken words
[3] speech recognition
[4] phonemes
[5] signal-interpretation
[6] computer-controlled
[7] sensor-based

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

Survey 1995) به کار رفته‌اند.

- یادگیری بازی‌های کلماتی نظیر تخته نرد

موفق‌ترین برنامه های بازی‌هایی مثل تخته نرد بر پایه‌ی الگوریتم‌های یادگیری ماشین نوشته شده‌اند. برای مثال، بهترین برنامه‌ی جهان برای تخته نرد، TD-Gammon (Tesauro 1992, 1995) استراتژی بازی را با یک میلیون بازی کردن در مقابل خودش یاد می‌گیرد. این برنامه هم اکنون در مسابقات جهانی با انسان‌ها مسابقه می‌دهد. تکنیک‌های مشابه در بسیاری از مسائل کاربردی که در آن‌ها فضای جستجو بسیار بزرگ است را می‌توان به کار برد.

جدول ۱.۱ چندین نمونه کاربرد موفق یادگیری ماشین.

- هوش مصنوعی

یادگیری نمایش نمادی مفاهیم. یادگیری ماشین به نگاه جستجو. یادگیری به عنوان روشی برای بهبود حل مسئله. استفاده‌ی همزمان از دانش قبلی و داده های آموزشی برای یادگیری.

- متد های بیزی

قضیه‌ی بیز به عنوان پایه‌ی محاسبه‌ی احتمالات فرضیه‌ها. دسته بندی کننده‌ی ساده‌ی بیز. الگوریتم‌های تخمین مقدار متغیر های نامعلوم.

- تئوری پیچیدگی محاسباتی[۲]

محدودیت‌های تئوری موجود بر روی پیچیدگی مسائل یادگیری مختلف، که در غالب پیچیدگی محاسباتی، تعداد نمونه های آموزشی، تعداد خطای قابل تحمل و ... بیان می‌شود.

- تئوری کنترل (پیش بینی)[۳]

رویه‌هایی[۴] که یاد می‌گیرد تا مقادیر از پیش تعیین شده ای را بهینه و مرحله‌ی بعدی فرآیند که کنترل می‌شود را پیش بینی کند.

- تئوری اطلاعات[۵]

معیار آنتروپی و مفهوم اطلاعات. روش کوتاه‌ترین توضیح در یادگیری. کد سازی بهینه و رابطه‌ی آن با سری آموزشی بهینه برای توصیف یک فرضیه.

- فلسفه

تیغ occam، که توصیه می‌کند بهترین فرضیه ساده‌ترین آن‌هاست. بررسی توجیه برای تعمیم فرای داده های آموزشی مشاهده شده.

- روانشناسی و عصب شناسی

قانون قدرت تمرین[۶]، که می‌گوید که سرعت عکس‌العمل انسان بر اثر تمرین بر روی مسائل مختلف یادگیری بهبود می‌یابد. تحقیقات عصب شناسی پایه‌ی مدل‌های شبکه های عصبی مصنوعی در یادگیری را تشکیل می‌دهند.

- آمار

[1] NASA
[2] computational complexity theory
[3] control theory
[4] Procedures
[5] information theory
[6] power low of practice

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

توصیف ویژگی‌های خطا (مثل، بایاس و واریانس) که موقع تخمین دقت یک فرضیه بر اساس نمونه داده های محدود انجام می‌گیرد. بازه های اطمینان، آزمون‌های آماری.

جدول ۱.۲ بعضی رشته های علمی و نمونه ای از تأثیرشان در یادگیری ماشین.

تعریف ما از یادگیری به اندازه‌ی کافی کلی است تا تمامی کارهایی که به طور کلی "یادگیری" نامیده می‌شود را در بر بگیرد. این تعریف به اندازه‌ی کافی نیز کلی هست تا برنامه های کامپیوتری‌ای که کارایی‌شان با تجربه بیشتر می‌شود را در بر بگیرد. برای مثال، یک پایگاه داده که به کاربرانش اجازه می‌دهد تا داده‌ها را تغییر دهند نیز با این تعریف ما از سیستم یادگیر تطابق دارد: زیرا که کارایی آن نیز با تجربه ای که حاصل تغییر داده های پایگاه داده افزایش می‌یابد. بدون نگرانی در شمول بیش از حد این تعریف می‌توان برنامه های یادگیر را برنامه‌هایی دانست که بر اثر تجربیات پیشرفت می‌کنند. در اینجا هدف از بحث بررسی مفهوم کلمه‌ی "یادگیری" نیست بلکه هدف در اینجا تعریف دقیق دسته ای از مسائل است که به نحوی به یادگیری مربوط می‌شوند. برای بررسی الگوریتم‌های حل چنین مسائلی و درک بهتر مبانی ساختاری مسائل و فرایند های یادگیری به چنین تعریف دقیقی نیاز داریم.

۱.۲ طراحی یک سیستم یادگیری

برای به تصویر کشیدن بعضی از مشکلات طراحی و روش‌های یادگیری ماشین بیایید طراحی برنامه ای برای یادگیری بازی چکرز با هدف بازی در مسابقات چکرز را بررسی کنیم. واضح است که کارایی را درصد بازی‌های برده در این مسابقات تعیین می‌کنیم.

۱.۲.۱ انتخاب تجربیات آموزشی

اولین انتخاب طراحی، انتخاب نوع تجربیات آموزشی است که انتظار می‌رود سیستم با آن‌ها یاد بگیرد است. انتخاب نوع تجربیات آموزشی می‌تواند اثر چشم گیری در موفقیت یا شکست یادگیر داشته باشد. یکی از ویژگی‌های مهم تجربیات آموزشی مستقیم یا غیر مستقیم بودن آن است. برای مثال، در یادگیری بازی چکرز، ممکن است تجربیات آموزشی چینش های صفحه‌ی چکرز با حرکت مناسب مربوطه باشند، که نمونه ای از تجربیات آموزشی مستقیم است. اما ممکن است اطلاعات به طور غیر مستقیم باشد، مثلاً سری‌ای از حرکات یادگیر و نتیجه‌ی بازی باشد. در این حالت، درستی هر حرکت خاص در این بازی باید به طور غیر مستقیم از این حقیقت که نتیجه‌ی بازی برد یا باخت بوده استنباط شود. پس یادگیر با مسئله‌ی دیگری، ارزش دهی[1] یا تعیین میزان تأثیر حرکات در نتیجه بازی مواجه است. نسبت دادن ارزش به حرکات نیز می‌تواند بسیار سخت باشد، زیرا که ممکن است حرکات ابتدایی بازی بسیار عالی بوده و اما نتیجه‌ی بازی باخت شده است. پس در حالت کلی یادگیری از تجربیات آموزشی مستقیم بسیار ساده تر از تجربیات غیر مستقیم است.

ویژگی مهم دوم تجربیات آموزشی درجه اختیار یادگیر در کنترل سری‌های نمونه های آموزشی است. برای مثال، ممکن است نمونه های ارائه شده به یادگیر توسط معلمی تعیین شود، یعنی معلمی چینش های صفحه را انتخاب کرده و آن‌ها را با حرکت متناسبشان به یادگیر بدهد. یا از طرف دیگر، یادگیر صفحاتی را که برایش ابهام دارند به معلم بدهد تا وی حرکت متناسبش را تعیین کند. یا حتی ممکن است یادگیر کنترلی هم بر چینش صفحات و هم به طور غیر مستقیم دسته بندی صفحات داشته باشد، برای مثال معلمی وجود نداشته باشد و برنامه در مقابل خودش بازی چکرز را انجام دهد. توجه دارید که در این صورت، یادگیر ممکن است انتخاب کند که وضعیت‌های نویی را که هنوز با آن مواجه نشده را بررسی کند یا در مقابل است وضعیت‌های گذشته‌اش را امتحان کند تا میزان امیدوار کننده بودن هر یک از وضعیت‌ها را معلوم کند. در فصول

[1] Credit assignment

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

آتی تعدادی از تعریف مسئله های یادگیری شامل مسائلی که در آن نمونه های آموزشی بـه طـور تصـادفی و خـارج از کنتـرل یـادگیر انتخـاب می‌شوند، مسائلی که یادگیر انواع مختلفی از آزمایش را به معلمی حرفه ای ارائه می‌کند و جواب را جویا مـی‌شـود، و مسـائلی کـه در آن یـادگیر نمونه های آموزشی را با حرکت خودکار در محیط اطراف خود بدست می‌آورد را بررسی خواهیم کرد.

ویژگی مهم سوم تجربیات آموزشی، میزان نمایندگی آن از توزیع نمونه‌هایی است که برای تعیین کردن کارایی P سیستم نهایی استفاده می‌شود. در کل، زمانی که نمونه های آموزشی توزیعی مشابه نمونه های تست دارند یادگیری قابل اعتماد تر است. در مسئله‌ی یادگیری بازی چکـرز مـا، معیار کارایی P درصد بازی‌های برده در مسابقات جهانی است. اگر تجربیات آموزشی E فقط از بازی مقابل خود سیستم بدست آمده باشـد، ایـن خطر به وضوح موجود است که تجربیات آموزشی ممکن است نمونه کاملی از توزیع حالات ممکنی که بعـداً در مسـابقات سیسـتم بـا آن تسـت می‌شود نباشد. برای مثال، یادگیر ممکن است هیچ گاه با حالات بسیار وخیمی که بسیار در بازی با انسان به وجود می‌آید مواجه نشـده باشـد. در عمل، گاهی لازم است که یادگیری بر روی مجموعه ای از نمونه‌هایی که با نمونه های تست نهایی متفاوتنـد آمـوزش داده شـود (بـرای مثـال ممکن است که مسابقات جهانی علاقه ای به آموزش سیستم ما نداشته باشد). در چنین شرایطی مشکل زا هستند زیرا که تسـلط بـر تـوزیعی از نمونه‌ها الزاماً به کارایی بالا در توزیع دیگر نمی انجامد. همان طور که خواهیم دید، مهم‌ترین تئوری یـادگیری ماشـین بـه ایـن فـرض اساسـی وابسته است که توزیع نمونه های آموزشی مشابه توزیع نمونه های تست است. بر خلاف این فرض که بـرای رسـیدن بـه نتـایج تئـوری انجـام می‌دهیم، باید در نظر داشت که گاهی در عمل این فرض کاملاً بر قرار نیست.

برای ادامه‌ی طراحی بیایید فرض کنیم که سیستم از بازی مقابل خودش آموزش می‌بیند. این فرض از این جهت که الزام وجود معلم خـارجی را از بین می‌برد مزیت دارد، از طرف دیگر سیستم می‌تواند تا جایی که زمان اجازه می‌دهد داده‌ی آموزشی ایجاد کند. حال مسـئله بـه طـور کامـل تعریف شده است:

مسئله‌ی یادگیری بازی چکرز:

- عمل T: بازی کردن چکرز
- کارایی P: درصد بازی‌های برده در مسابقات
- تجربیات آموزشی E: بازی‌هایی که در مقابل خودش انجام می‌دهد

حال برای کامل کردن طراحی سیستم یادگیری باید موارد زیر را معلوم کنیم،

۱. نوع دقیق دانشی که قصد داریم سیستم یاد بگیرد
۲. نمایشی برای این دانش هدف
۳. روشی برای یادگیری

۱.۲.۲ انتخاب تابع هدف

مرحله‌ی بعدی طراحی تعیین دقیق نوع دانشی و چگونگی استفاده از این دانش برای بهبود کارایی سیستم است. بیایید بـا یـک برنامـه‌ی بـازی چکرز شروع کنیم که حرکات مجاز را در هر چینش صفحه تشخیص می‌دهد. حال کافی است فقط راهی برای تعیین بهتـرین حرکـت در میـان حرکات مجاز یاد بگیریم. این کار یادگیری نماینده‌ی دسته‌ی بزرگی از کارهای یادگیری است که در آن تعدادی عمل مجاز در دسترس اسـت و

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

فضای جستجو بسیار بزرگی نیز مشخص شده است اما روش پیدا کردن بهترین حرکت معلوم نیست. بسیاری از مسائل بهینه سازی[1] از این دسته مسائلند، مسائلی مثل برنامه ریزی و کنترل خط تولید که در آن‌ها مراحل تولید مشخصند اما بهترین استراتژی ترتیب آن‌ها مشخص نیست مثالی از این گونه مسائل است.

با این تعریف مسئله، باید یاد بگیریم تا از میان حرکت‌های مجاز یکی را انتخاب کنیم، واضح‌ترین گزینه برای نوع اطلاعات یادگیری، یک برنامه یا تابع است که بهترین حرکت را با داشتن چینش صفحه پیدا می‌کند. بیایید این تابع را ChooseMove بنامیم و ChooseMove: B → M. توجه دارید که این تابع چینشی مجاز از چینش های مجاز صفحه B را دریافت کرده و حرکتی را از میان حرکات مجاز M به عنوان خروجی می‌دهد. در سراسر بحث یادگیری ماشین، همیشه بد نیست که مسئله‌ی بهینه سازی کارایی P در عمل T را به مسئله‌ی یادگیری یک تابع مثل ChooseMove کاهش دهیم. بنابراین انتخاب تابع هدف یکی از انتخاب‌های کلیدی طراحی خواهد بود.

با وجود اینکه تابع ChooseMove در مثال ما بسیار ساده تعریف می‌شود اما یادگیری آن با داشتن تجربیات آموزشی غیر مستقیم برای سیستم بسیار سخت خواهد بود. می‌توان بجای چنین تابعی، تابعی دیگر، که در این تعریف مسئله یادگیری‌اش بسیار ساده تر است، را یاد گرفت، این تابع تابعی ارزیاب[2] است که به هر چینش صفحه یک ارزش یا امتیاز نسبت می‌دهد. بیایید این تابع را V بنامیم و با توجه به نام گذاری‌های قبلی خواهیم داشت، $V: B \rightarrow \Re$، یعنی تابع V به هر چینش صفحه‌ی مجاز یک عدد حقیقی نسبت می‌دهد ($\Re$ برای نماد اعداد حقیقی به کار می‌رود). ما می‌خواهیم که تابع هدف V به چینش های بهتر صفحه عددی بیشتر نسبت دهد. اگر سیستم بتواند با موفقیت چنین تابع V ای را یاد بگیرد می‌تواند به راحتی بهترین حرکت در هر چینش صفحه را انتخاب کند. این کار را می‌توان با تولید چینش های آتی صفحه که پس از هر یک از حرکات مجاز ایجاد می‌شود و مقایسه‌ی مقادیر V آن‌ها انجام داد (حرکت نظیر بهترین چینش بهترین حرکت مجاز است).

اما دقیقاً چگونه می‌توان مقدار تابع هدف V را برای هر چینش صفحه مشخص کرد؟ البته، هر تابع ارزیابی‌ای که به چینش های بهتر عدد بیشتری نسبت دهد قابل قبول است. با این وجود بهتر است که تابع هدفی خاص را در میان تمامی توابعی که حرکت بهینه را تشخیص می‌دهند برای V مشخص کنیم. همان طور که بعداً نیز خواهیم دید، بهتر است الگوریتمی برای یادگیری طراحی شود. پس بیایید مقدار V(b) را که b چینشی از مجموعه چینش های ممکن B است را به صورت زیر تعریف کنیم:

۱. اگر b چینشی انتهایی برنده بود، V(b)=100
۲. اگر b چینشی انتهایی بازنده بود، V(b)=-100
۳. اگر b چینشی انتهایی مساوی بود، V(b)=0
۴. اگر b چینشی در انتهای بازی نبود، V(b)=V(b′) که b′ بهترین چینش صفحه‌ی ممکن حاصل از چینش b با بازی بهینه تا آخر بازی (با فرض اینکه حریف نیز بهینه بازی کند) خواهد بود.

با اینکه تعریف بازگشتی از مقدار V(b) برای هر چینش صفحه‌ی b تعیین می‌کند، این تعریف برای بازیکن چکرز ما قابل استفاده نخواهد بود زیرا که مقادیر قابل محاسبه نیست. مگر در حالت‌های انتهایی (موارد ۱ تا ۳) که در آن‌ها بازی تمام شده است و مشخص کردن V(b) ارزشی ندارد، مشخص کردن مقدار V(b) برای یک چینش صفحه‌ی خاص (مورد ۴) به جستجو برای سری بهینه ای از حرکات می‌انجامد که بازی را

[1] optimation
[2] Evaluation function

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

به آخر می‌رساند! چون این تعریف برای برنامه‌ی چکرز ما قابل محاسبه نیست، این تعریف تعریفی غیرعملی[1] نامیده می‌شود. هدف یادگیری در این مرحله پیدا کردن تعریفی عملی[2] از V است؛ تعریفی که بتوان آن‌را در برنامه‌ی بازی چکرز برای ارزیابی چینش‌ها و انتخاب حرکات به کار گرفت.

بنابراین، کار یادگیری را در این مثال به مسئله‌ی پیدا کردن تعریفی عملی از تابع هدف V کاهش دادیم. یادگیری فرم دقیقی از V در حالت کلی خیلی سخت خواهد بود. در واقع، گاهی اوقات فقط انتظار داریم که الگوریتم‌های یادگیری تخمینی از تابع هدف را پیدا کنند و به همین دلیل فرایند یادگیری تابع هدف تخمین[3] تابع هدف نیز نامیده می‌شود. در بحث فعلی از نماد $\hat{V}$ برای تابع یادگیری شده (تخمین تابع هدف V) استفاده می‌کنیم.

۱.۲.۳ انتخاب نحوه‌ی نمایش تابع هدف

حال که تابع هدف V را مشخص کردیم، باید نمایشی انتخاب کرده تا برنامه بتواند تابع $\hat{V}$ را با آن نشان دهد. مثل انتخاب‌های قبلی طراحی در اینجا نیز با گزینه های بسیاری مواجهیم. برای مثال، می‌توانیم به برنامه اجازه دهیم که $\hat{V}$ را با جدول بزرگی از مقادیر برای هر یک از چینش های صفحه نشان دهد. یا می‌توانیم به آن اجازه دهیم تا $\hat{V}$ را با مجموعه ای از قوانین که با ویژگی‌های چینش صفحه مطابقت دارد یا تابعی درجه دو از ویژگی‌هایی از پیش تعریف شده یا یک شبکه‌ی عصبی مصنوعی نمایش دهد. در کل، این انتخاب نمایش شامل یک مقایسه‌ی مهم است. در یک طرف، سعی می‌کنیم نمایشی که انتخاب کنیم کاملاً شامل باشد تا بتوان آن‌را به اندازه‌ی کافی به تعریف ایده آل V نزدیک کرد. از طرف دیگر، با شامل‌تر بودن این نمایش تعداد داده های آموزشی که برنامه نیاز خواهد داشت تا میان فرضیه‌ها بتواند مناسب‌ترین را انتخاب کند بیشتر خواهد شد. خلاصه اینکه، بیایید یک نمایش ساده را انتخاب کنیم: V را به عنوان ترکیب خطی[4] ویژگی‌های زیر در نظر می‌گیریم:

- x_1: تعداد مهره های سیاه در صفحه
- x_2: تعداد مهره های قرمز در صفحه
- x_3: تعداد مهره های شاه سیاه در صفحه
- x_4: تعداد مهره های شاه قرمز در صفحه
- x_5: تعداد مهره های سیاه تهدید شده توسط قرمز (که سیاه می‌تواند در حرکت بعدی آن‌را بگیرد)
- x_6: تعداد مهره های قرمز تهدید شده توسط سیاه

بنابراین برنامه تابع $\hat{V}(b)$ را با تابعی خطی به فرم زیر بیان خواهد کرد:

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

در این رابطه w_0 تا w_6 ضرایب عددی یا همان وزن‌ها هستند که توسط الگوریتم یادگیری تعیین می‌شوند. مقادیر w_1 تا w_6 اهمیت نسبی متغیر های مختلف صفحه مشخص می‌کنند و w_0 نیز ثابتی به این مقدار صفحه اضافه می‌کند.

---

[1] nonoperational definition
[2] operational definition
[3] approximation
[4] linear combination

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

به طور خلاصه، با انتخاب‌های طراحی مان تا به حال، نوع تجربیات یادگیر، تابع هدف تخمینی و فرمی برای نمایش آن بـوده اسـت. مسـئله در حال حاضر به شکل زیر است:

طراحی میانی برنامه‌ی یادگیری چکرز:

- کار T: بازی چکرز
- کارایی P: درصد بازی‌های برده در مسابقات
- تجربیات آموزشی E: بازی‌هایی که برنامه مقابل خود انجام می‌دهد
- تابع هدف: $V : Board \rightarrow \Re$
- نمایش تابع هدف:

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

سه قسمت اول کار یادگیری را مشخص می‌کنند در حالی که دو قسمت انتهایی مربوط به انتخاب‌های طراحی ما بـرای پیـاده سـازی برنامـه‌ی یادگیری هستند. توجه کنید که اضافه کردن این قسمت‌ها فقط برای کاهش مسئله‌ی یادگیری اسـتراتژی بـازی چکـرز بـه مسـئله‌ی یـادگیری مقادیر ضرایب w_0 تا w_6 موجود در نمایش تابع هدف است.

۱.۲.٤ انتخاب یک الگوریتم تخمین تابع

برای یادگیری تابع هدف $\hat{V}$ نیاز به مجموعه ای از نمونه‌های آموزشی داریم، که هر کـدام یـک چینش صفحه‌ی b و یـک مقـدار یـادگیری $V_{train}(b)$ برای b است. به عبارت دیگر، هر نمونه‌ی آموزشی زوج مرتبی به شکل $< b, V_{train}(b) >$ است. برای مثال، نمونه‌ی زیـر چینشی را نشان می‌دهد که سیاه بازی را برده است (توجه دارید که $x_2 = 0$ بدین معناست که قرمز مهره‌ی دیگری در صفحه ندارد) بنـابراین مقدار V در این نمونه 100+ خواهد بود.

$$<< x_3 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0 >, +100 >$$

در زیر فرایندی را که ابتدا چنین نمونه های آموزشی‌ای را از تجربیات آموزشی غیر مستقیم استخراج می‌کننـد و سـپس وزن‌هـای w_i را بـرای نمونه های آموزشی پیدا می‌کنند را توضیح خواهیم داد.

۱.۲.٤.۱ تخمین مقادیر آموزشی

با توجه به فرضی که در مسئله‌ی یادگیری کردیم، تنها اطلاعات آموزشی موجود برای یادگیر این است که آیا بازی نتیجه‌ی بازی بـرد بـوده یـا باخت. در مقابل، نمونه های آموزشی‌ای لازم داریم که به هر یک از چینش های صفحه یک امتیاز نسبت دهنـد. بـا وجـود اینکـه نسـبت دادن مقدار به چینش‌هایی که انتهای بازی هستند بسیار ساده است اما نسبت دادن مقادیر آموزشی عددی به چینش‌هایی که در وسط بازی قرار دارند، اصلاً ساده نیست. البته این حقیقت که نتیجه‌ی بازی برد یا باخت بوده نشان نمی‌دهد که تک تک چینش‌های صفحه بازی خـوب یـا بـد بـوده است. برای مثال اگر برنامه بازی را ببازد، ممکن است بدین معنا باشد که چینش‌های ابتدایی صفحه بازی باید عدد بیشتری داشته و چینش‌های انتهایی عدد کمتری دارد و باخت نتیجه‌ی سری حرکات ضعیف میانی‌ای بوده است.

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

برخلاف ابهام ذاتی در تخمین مقادیر آموزشی چینش‌های میانی بازی، یک روش ساده بسیار مفید عمل می‌کند. این روش مقدار آموزشی $V_{train}(b)$ را برای هر چینش میانی b، $\hat{V}(successor(b))$ مقدار دهی می‌کند، در این مقدار $\hat{V}$ تخمین فعلی یاد گرفته شده از V و successor(b) نیز چینشی است که بعد از حرکت برنامه دوباره نوبت به وی می‌رسد (چینشی که پس از حرکت برنامه و حرکت حریف ایجاد می‌شود). این قانون برای تخمین مقادیر آموزشی را می‌توان به صورت زیر بیان کرد:

قانون تخمین مقادیر آموزشی:

$$V_{train}(b) \leftarrow \hat{V}\big(successor(b)\big) \qquad (1.1)$$

با وجود اینکه استفاده از $\hat{V}$ تخمینی (که خود از همین داده‌ها تخمین زده می‌شود) برای تخمین جدید مقادیر عجیب به نظر می‌رسد، اما این روش طبق تجربه موجه است. توجه دارید که از مقدار successor(b) برای تخمین مقدار چینش b استفاده می‌کنیم. شهوداً واضح است که دقت $\hat{V}$ در نزدیکی چینش‌های انتهایی افزایش می‌یابد. در واقع در شرایطی (که در فصل ۱۳ بحث خواهد شد) روش تخمین تکراری مقادیر آموزشی بر اساس تخمین چینش‌های successor ثابت می‌شود که به V_{train} میل خواهد کرد.

۱.۲.٤.۲ تنظیم وزن‌ها

تنها کار باقی مانده معین کردن الگوریتم یادگیری برای انتخاب وزن‌های w_i به صورتی است که بهترین تناسب را با نمونه های آموزشی $\{< b, V_{trian}(b) >\}$ داشته باشد است. به عنوان اولین مرحله، ابتدا باید تعریف کنیم که منظور از بهترین تناسب با داده های آموزشی چیست. یکی از روش‌های ممکن تعریف این بهترین فرضیه، یا بهترین مجموعه وزن‌ها به صورتی است که خطای مربعی E بین مقادیر آموزشی و مقادیر تخمینی $\hat{V}$ را مینیمم کنیم.

$$E \equiv \sum_{<b,V_{train}(b)> \in training\ examples} \left(V_{train}(b) - \hat{V}(b)\right)^2$$

بنابراین ما به دنبال وزن‌هایی هستیم، یا به طور مشابه به دنبال $\hat{V}$ ی هستیم که مقدار E را برای نمونه های آموزشی مشاهده شده مینیمم کند. در فصل ۶ ثابت می‌کنیم که در مسئله‌هایی مثل مسئله‌ی ما آن مینیمم کردن مجموع خطای مربعی متناظر با پیدا کردن محتمل‌ترین فرضیه با داشتن داده های آموزشی است.

الگوریتم‌های بسیاری برای پیدا کردن وزن‌های توابع خطی که E را مینیمم می‌کنند وجود دارد. در این حالت الگوریتمی مورد نیاز است که مرحله به مرحله با افزایش نمونه های آموزشی در وزن‌ها تجدید نظر کند و همچنین نسبت به خطای تخمین مقادیر نمونه های آموزشی حساسیت کمی داشته باشد. یکی از این الگوریتم‌ها، الگوریتم کمترین خطای مربعی یا LMS[1] نامیده می‌شود. این الگوریتم برای هر نمونه‌ی آموزشی مشاهده شده وزن‌ها را به اندازه‌ی کوچک در جهتی که خطا را برای نمونه‌ی آموزشی کم می‌کند تغییر خواهد داد. همان طور که در فصل ۴ نیز بررسی خواهیم کرد، این الگوریتم را می‌توان جستجوی شیب نزول تصادفی‌ای در فضای فرضیه های ممکن (مقادیر مختلف وزن‌ها) برای مینیمم کردن خطای مربعی E دانست . الگوریتم LMS به فرم زیر تعریف می‌شود:

[1] Least Mean Squars (LMS)

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

قانون تغییر وزن LMS.

برای هر نمونه‌ی آموزشی $< b, V_{train}(b) >$

- از وزن‌های فعلی برای محاسبه‌ی $\hat{V}(b)$ استفاده کن.
- برای هر وزن w_i، تغییر زیر را اعمال کن

$$w_i \leftarrow w_i + \eta \left(V_{train}(b) - \hat{V}(b) \right) x_i$$

در اینجا η ثابت کوچکی (مثل ۰.۱) است که اندازه‌ی تغییر وزن را متعادل می‌کند. برای درک شهودی اینکه چرا این قانون تغییر وزن درست کار می‌کند، توجه کنید که زمانی که خطای $\left(V_{train}(b) - \hat{V}(b) \right)$ صفر است، وزن‌ها تغییری نخواهند کرد و زمانی که $\left(V_{train}(b) - \hat{V}(b) \right)$ مثبت است (برای مثال $\hat{V}(b)$ کمتر از انتظار است)، به هر وزن به نسبتی افزایش خواهد داد. این عمل مقدار $\hat{V}(b)$ را افزایش داده و در نهایت میزان خطا کمتر می‌شود. توجه داشته باشید که اگر مقدار ویژگی x_i صفر باشد، مستقل از این که خطا چه مقدار باشد وزن تغییری نخواهد کرد، بنابراین، تنها وزن‌ها نظیر متغیرهایی تغییر خواهند کرد که واقعاً در صفحه‌ی بازی اتفاق می‌افتند. جالب است که، اثبات می‌شود که این متد تنظیم وزن ساده حتماً به کمترین خطای مربعی تقریبی برای مقادیر V_{train} میل خواهد کرد (فصل ۴).

۱.۲.۵ طراحی نهایی

طراحی نهایی سیستم یادگیری چکرز را می‌توان با ۴ قسمت[1] برنامه نشان داد که پایه‌ی اصلی بسیاری از سیستم‌های یادگیری هستند. این چهار قسمت در شکل ۱.۱ به طور خلاصه نشان داده شده‌اند:

- **سیستم کارایی** قسمت است که باید مسئله‌ی پیدا کردن کارایی را حل کند، در مثال چکرز، این کار باید با استفاده از تابع هدف‌های یاد گرفته شده انجام شود. این قسمت نمونه ای از مسئله ای جدید (بازی جدید) به عنوان ورودی دریافت کرده و مسیری[2] برای حل آن [3] خروجی می‌دهد. در مثال ما، استراتژی سیستم کارایی در انتخاب حرکت بعدی در هر مرحله توسط تابع $\hat{V}$ مشخص می‌شود. بنابراین انتظار داریم که کارایی سیستم با افزایش دقت این تابع ارزیابی افزایش یابد.
- **کارشناس** مسیری از حرکات بازی را دریافت کرده و آن‌ها را به مجموعه ای از نمونه های آموزشی تبدیل می‌کند و خروجی می‌دهد. همان طور که در شکل نیز نشان داده شده است، هر نمونه‌ی آموزشی در این مثال متناسب با چینشی از صفحه در مسیر بازی و مقادیر تخمینی V_{train} شان است. در مثال ما، همان قانون یادگیری رابطه‌ی ۱.۱ است.
- **تامیم دهنده** مجموعه ای از نمونه های آموزشی را دریافت کرده و فرضیه ای متناسب با آن خروجی می‌دهد، این فرضیه همان تخمین تابع هدف است. این قسمت نمونه های آموزشی محدود را تامیم می‌دهد، و فرضیه ای که تابعی کلی است و این مجموعه و دیگر نمونه‌ها را می‌پوشاند ارائه می‌کند. در مثال ما، تامیم دهنده الگوریتم LMS بود و خروجی آن نیز $\hat{V}$ بود که با وزن‌های w_0 تا w_6 مشخص می‌شد.

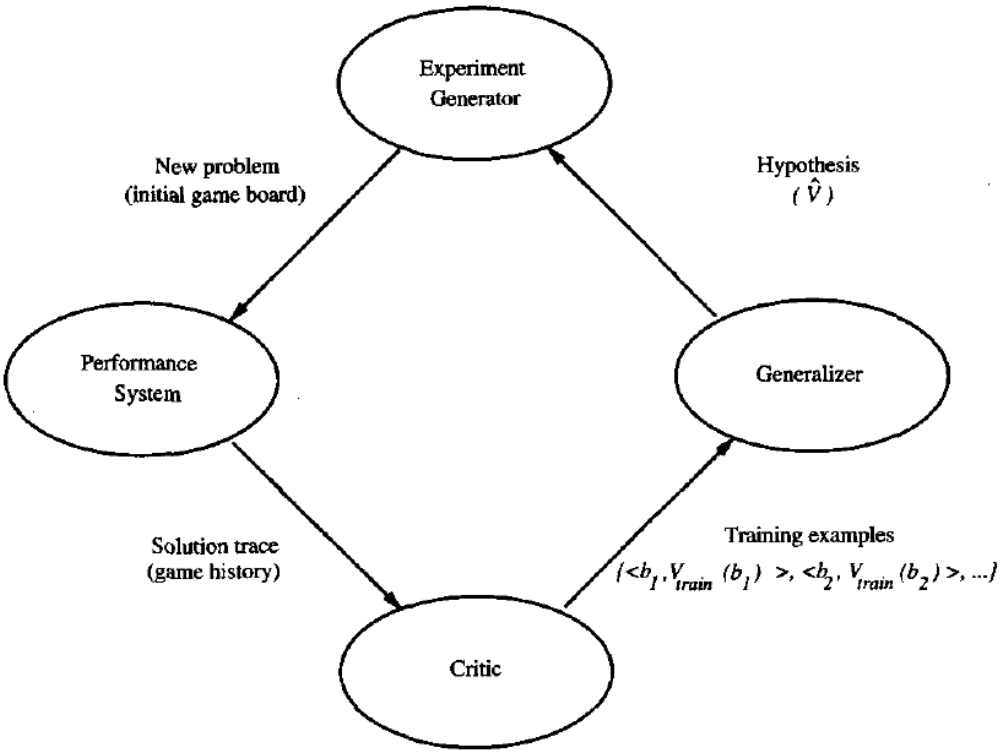
---

[1] module
[2] trace
[3] game history

www.farabar.net

Telegram: @Farabar_BI

- **ایجاد کننده‌ی تجربه** فرضیه‌ی فعلی (تابعی که تا کنون یاد گرفته‌ایم) را به عنوان ورودی دریافت می‌کند و مسئله ای جدید ایجاد می‌کند (صفحه ای جدید ایجاد می‌کند) تا سیستم کارایی در آن به کاوش بپردازد. نقش این سیستم انتخاب مسئله های تمرینی جدیدی که سرعت یادگیری را به حداکثر برساند. در مثال ما، ایجاد کننده‌ی تجربه یک استراتژی بسیار ساده را دنبال می‌کرد: همیشه یک چینش صفحه‌ی ثابت را برای ایجاد بازی جدید انتخاب می‌کرد. در استراتژی‌های پیچیده تر را می‌توان برای کاوش ناحیه های خاص فضای چینش صفحه به کار برد.



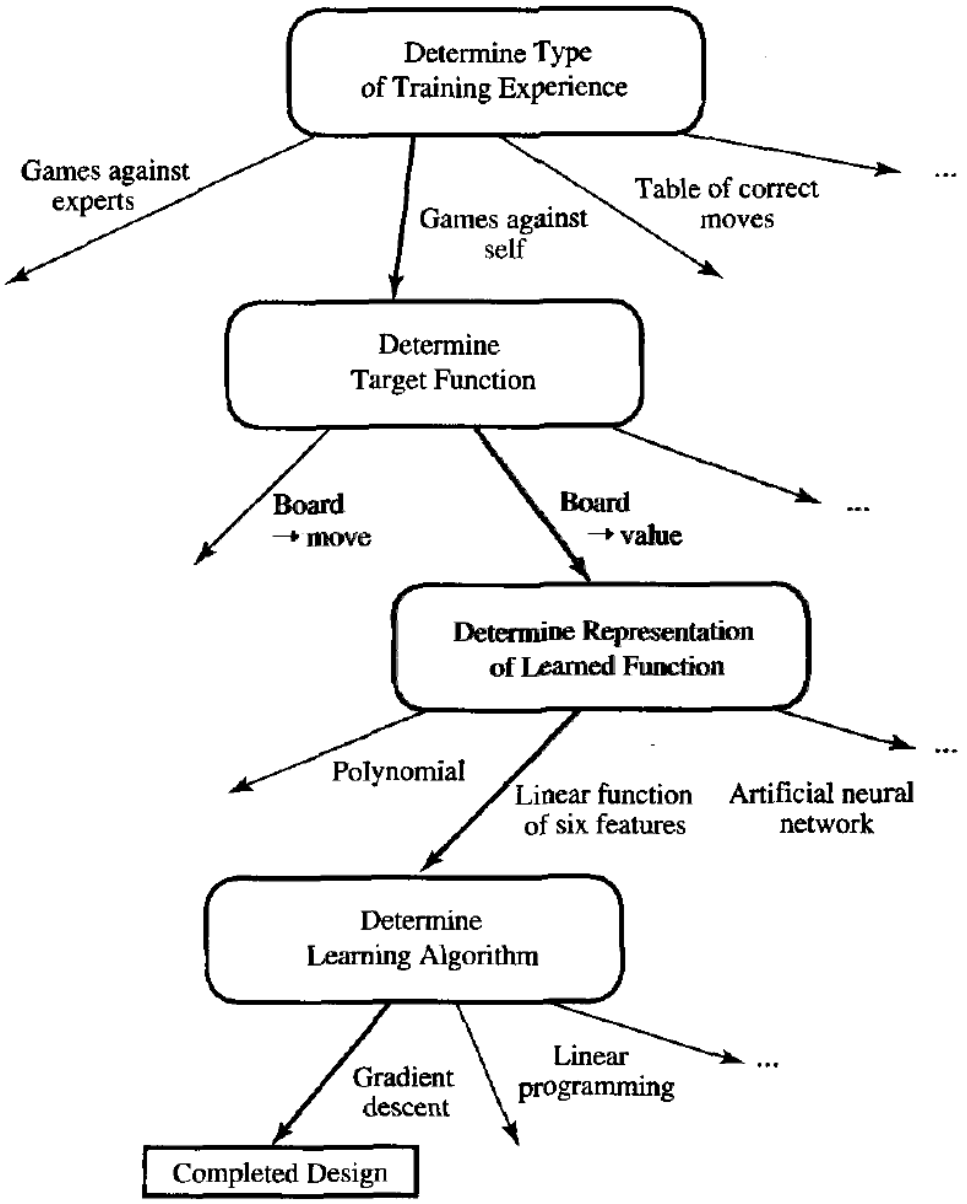
شکل ۱.۱ طراحی نهایی برنامه یادگیری چکرز.

انتخاب‌ها طراحی‌ای که برای طراحی برنامه‌ی بازی چکرز انجام دادیم ویژگی‌های دقیق سیستم‌های کارایی، کارشناس، تامیم دهنده و ایجاد کننده‌ی تجربه، را تعیین می‌کند. بسیاری از سیستم‌های یادگیری ماشین را می‌توان در فرم همین چهار قسمت بیان کرد.

ترتیب انتخاب گزینه های طراحی برای بازی چکرز در شکل ۱.۲ به طور خلاصه آورده شده است. این انتخاب‌های طراحی کار یادگیری را از چندین نظر محدود کرده است. برای مثال، نوع دانشی که ذخیره می‌شود را به تابع خطی محدود کرده‌ایم. علاوه بر این، تابع خطی را فقط تابع ۶ متغیر خاص از صفحه‌ی بازی فرض کرده‌ایم. اگر تابع هدف V را بتوان با ترکیبی خطی از ویژگی‌ها نمایش داد، برنامه‌ی ما با احتمال خوبی V را تخمین خواهد زد. اما اگر V را نتوان با ترکیب خطی این متغیرها نشان داد در بهترین حالت می‌توان از برنامه انتظار داشت که تقریب خوبی از آن‌را یاد بگیرد. زیرا که هیچ برنامه ای نمی‌تواند چیزی را که نمی‌تواند نمایش دهد یاد بگیرد.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net



شکل ۱.۲ خلاصه‌ی انتخاب‌های طراحی برنامه‌ی بازی چکرز.

بیایید فرض کنیم که تقریب خوبی از V را بتوان با این فرم نمایش داد. حال این سؤال مطرح خواهد بود که آیا این تکنیک‌های یادگیری تضمین می‌کنند که در صورت وجود این تقریب آن را پیدا کنند. فصل ۱۳ بررسی ی تئوری انجام می‌دهد که برای تحت شرایط محدود کننده ای، روشی مشابه این روش واقعاً به سمت تابع ارزیابی میل خواهد کرد. خوشبختانه در نتایج عملی مشاهده می‌شود که حتی هنگامی که از محدوده‌ی شرایط اثبات خارج می‌شویم معمولاً این روش برای یادگیری تابع ارزیابی موفقیت آمیز است.

آیا برنامه ای که ما طراحی می‌کنیم به اندازه‌ی کافی قوی خواهد بود تا بتواند بازیکنی جهانی را ببرد؟ احتمالاً خیر. این به خاطر این است که نمایش خطی تابع $\hat{V}$ بسیار ساده است و نمی‌تواند جزئیات بازی را تعیین کند. با این وجود، با در نظر گرفتن نمایشی پیچیده تر برای تابع هدف، این روش کلی می‌تواند بسیار موفقیت آمیز باشد. برای مثال (Tesauro 1992, 1995) طرحی برای برنامه ای که یاد می‌گیرد تخته نرد بازی کند را با یادگیری تابع ارزیابی مشابهی بر روی وضعیت‌ها را ارائه می‌کند. برنامه‌ی وی تخمین تابع یاد گرفته شده را با استفاده از شبکه ای

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

عصبی که ویژگی‌های کامل وضعیت صفحه را به جای زیرمجموعه ای از ویژگی‌های صفحه دریافت می‌کند نمایش می‌دهد. بعد از آموزش بر روی یک میلیون بازی آموزشی خودساخته[1] برنامه‌ی وی توانست در مقابل بازیکنان سطح بالای تخته نرد بازی کند.

البته می‌توانستیم الگوریتم‌های دیگری را برای کار یادگیری بازی چکرز طراحی کنیم. برای مثال، ممکن بود نمونه های آموزشی را ذخیره کرده، و در وضعیت‌های جدید در درون مجموعه‌ی ذخیره شده به دنبال نمونه های مشابه بگردیم (الگوریتم nearest-neighbor، فصل ۸). یا می‌توانستیم تعداد زیاد از برنامه های بازی چکرز را ایجاد کرده و اجازه‌ی بازی به آن‌ها بدهیم و موفق‌ترین آن‌ها را حفظ کرده و با این مجموعه با جهش[2] و ترکیب برنامه‌ها را تکامل دهیم (الگوریتم‌های ژنتیک، فصل ۹). به نظر می‌رسد انسان‌ها روشی متفاوت برای یادگیری استفاده می‌کنند، در این روش آن‌ها شرایط را بررسی کرده و توضیحات و دلایلی برای موفقیت یا شکست بازی ایجاد می‌کنند (یادگیری توضیحی، فصل ۱۱). طراحی ما یکی از طراحی‌های ممکن است و برای آشنایی با بحث و انتخاب‌های طراحی متد یادگیری دسته ای از مسائل آورده شده است.

۱.۳ دورنما و مشکلات یادگیری ماشین

یکی از نگاه های یادگیری ماشین، جستجویی میان فضای فرضیه ای نسبتاً بزرگ فرضیه های ممکن برای مشخص کردن بهترین فرضیه با توجه به داده های آموزشی موجود و دانش قبلی است. برای مثال، فضای فرضیه ای تمامی فرضیه‌هایی خروجی یادگیر بازی چکرز (که در بالا طراحی کردیم) را در نظر بگیرید. این فضای فرضیه ای شامل تمامی توابع ارزیابی‌ای می‌شود که می‌توان آن‌ها را در قالب w_0 تا w_6 بیان کرد. پس کار یادگیر جستجو در میان این فضای فرضیه ای وسیع برای پیدا کردن سازگارترین فرضیه با نمونه های آموزشی موجود است. الگوریتم LMS با تکرار تغییر وزن‌ها و تصحیح تخمین‌های اشتباه تابع در هر مرحله به این تابع ارزیابی دست پیدا می‌کند. این الگوریتم را می‌توان هنگامی که نمایش فرضیه یادگیر با پارامتر های پیوسته است کار برد.

بسیاری از فصول این کتاب الگوریتم‌هایی را ارائه می‌کنند که فضای فرضیه ای تعریف شده با استفاده از نمایشی خاص (مثل توابع خطی، توصیف منطقی، درخت تصمیم و شبکه های عصبی) را جستجو می‌کنند. این نمایش‌های متفاوت فرضیه‌ها برای یادگیری انواع مختلف توابع هدف است. در هر یک از این نمایش فرضیه‌ها، الگوریتم یادگیری مناسب از ساختاری نمایش برای ترتیب جستجو در فضای فرضیه ای کمک می‌گیرد.

در تمام طول این کتاب، از این نگاه به مسائل یادگیری برای دسته بندی متد های یادگیری بر اساس استراتژی‌های جستجو و ساختار فضای فرضیه ای مورد جستجو کمک می‌گیریم. همچنین این نگاه را برای بررسی رسمی روابط بین اندازه‌ی فضای فرضیه ای مورد جستجو، تعداد نمونه های آموزشی موجود و اطمینان تعمیم فرضیه نهایی بر روی داده های جدید کار می‌بریم.

۱.۳.۱ مشکلات یادگیری ماشین

مسئله‌ی مطرح شده چکرز سؤالات کلی‌ای درباره‌ی یادگیری ماشین ایجاد می‌کند. یادگیری ماشین و اکثر متن این کتاب برای جواب به چنین سؤالاتی است:

---

[1] self-generated
[2] mutate

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- چه الگوریتم‌هایی برای یادگیری کلی توابع هدف از نمونه های آموزشی خاص وجود دارد؟ در چه شرایطی یک الگوریتم خاص با داشتن نمونه های آموزشی کافی به تابع مورد نظر میل می‌کند؟ چه الگوریتم‌هایی برای چه نوع از مسائل و نمایش‌ها کارایی بهتری دارند؟
- چه میزان داده‌ی آموزشی کافی است؟ چه محدودیت‌های کلی‌ای را می‌توان برای رابطه‌ی اطمینان فرضیه‌ها، میزان تجربیات آموزشی و ویژگی‌های فضای فرضیه ای یادگیر بدست آورد؟
- در چه شرایطی و چگونه دانش قبلی یادگیر می‌تواند به فرآیند یادگیری کمک کند؟ آیا دانش قبلی زمانی که کاملاً درست نیست نیز می‌تواند به فرآیند یادگیری کمک کند؟
- بهترین روش انتخاب تجربه‌ی آموزشی بعدی چیست، و چگونه این انتخاب این روش پیچیدگی یادگیری مسئله را تغییر می‌دهد؟
- بهترین راه کاهش کار یادگیری به یک یا چند مسئله‌ی تابع تخمین چیست؟ به عبارت دیگر، چه توابع خاصی را باید هدف یادگیری قرار داد؟ آیا می‌توان خود این فرایند را خودکار[1] کرد؟
- چگونه یادگیر می‌تواند به طور خودکار نحوه‌ی نمایش را برای بهتر کردن قدرت نمایش و یادگیری تابع هدف تغییر دهد؟

۱.٤ این کتاب را چگونه بخوانیم

این کتاب شامل مقدمه ای بر الگوریتم‌ها و روش‌های ابتدایی یادگیری ماشین و نتیجه های تئوری از امکان[2] یادگیری کارهای مختلف و ظرفیت‌های الگوریتم‌های خاص و نمونه های کاربردی یادگیری ماشین در جهان واقعی است. فصل‌های این کتاب را می‌توان به هر ترتیب دلخواه خواند، با این وجود بعضی وابستگی‌ها بین فصول اجتناب ناپذیرند. اگر این کتاب را برای سیلابس درسی استفاده می‌کنید، واقعاً توصیه می‌شود که ابتدا به فصول ۱ و ۲ پرداخته شود. به جز این دو فصل بقیه فصول را می‌توان تقریباً به هر ترتیب دلخواه ممکن خواند. برای کلاسی که یک ترم خواهد بود متن ۷ فصل کافی خواهد بود، البته فصول اضافی می‌تواند برای مطالعه‌ی آزاد گذاشته شود (که از اهمیت قابل توجهی برخوردارند). در زیر خلاصه ای از آنچه در هر فصل آورده شده آمده:

- فصل ۲ به یادگیری مفهوم بر پایه‌ی نمایش نمادین[3] و نمایش منطقی است. همچنین در این فصل ترتیب کلی به جزئی فرضیه‌ها و بایاس استقرایی و اهمیتش بررسی شده است.
- فصل ۳ یادگیری درختی و مسئله‌ی overfit را بررسی می‌کند. همچنین تیغ occam قانونی که فرضیه های کوتاه‌تر را ترجیح می‌دهد– نیز آورده شده است.
- فصل ۴ به یادگیری شبکه های عصبی و الگوریتم Backpropagation و روش کلی شیب نزول می‌پردازد. این بخش شامل مثالی از پردازش اطلاعات تصویر (صورت انسان) نیز می‌شود. آدرس منابع داده‌ها و الگوریتم‌های اضافه نیز آورده شده است.
- فصل ۵ به مفاهیم پایه ای تئوری آمار و تخمین با تمرکز بر ارزیابی دقت فرضیه‌ها با استفاده از داده های محدود می‌پردازد. این فصل به بازه های اطمینان[4] برای تخمین دقت فرضیه‌ها و متد های مقایسه‌ی دقت متد های مختلف یادگیری می‌پردازد.

---

[1] automatic
[2] feasibility
[3] symbolic representation
[4] confidence interval

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- فصل ۶ به یادگیری بیزی در یادگیری ماشین می‌پردازد، این فصل به کاربرد بررسی بیزی هم برای بررسی الگوریتم‌های یادگیری ی غیر بیزی و هم برای الگوریتم‌های یادگیری بیزی، که احتمال فرضیه های را محاسبه می‌کنند، می‌پردازد. این فصل همچنین شامل مثالی از به کار بردن دسته بندی کننده‌ی ساده‌ی بیز در مسئله‌ی دسته بندی متون، با استفاده از برنامه و داده های اینترنت می‌شود.
- فصل ۷ نظریه‌ی یادگیری محاسباتی را پوشش می‌دهد. در این فصل به مدل یادگیری تقریباً درست (probably approximately correct (PAC)) و مدل یادگیری مرز خطا[1] خواهیم پرداخت. علاوه بر این در این فصل به الگوریتم رای گیری وزن دار[2] نیز خواهیم پرداخت که روشی برای ترکیب الگوریتم‌های یادگیری است.
- فصل ۸ به متد های یادگیری مبتنی بر نمونه‌ها می‌پردازد، این متدها شامل یادگیری نزدیک‌ترین همسایه‌ها، برازش وزن دار محلی و case-based reasoning می‌شود.
- فصل ۹ الگوریتم‌های یادگیری که با الهام از تکامل زیستی ساخته شده‌اند را بررسی می‌کند. این الگوریتم‌ها شامل الگوریتم‌های ژنتیک و برنامه نویسی ژنتیک می‌شوند.
- فصل ۱۰ الگوریتم‌هایی که برای یادگیری دسته قوانین، شامل روش‌های برنامه نویسی منطقی استقرایی برای horn clause های درجه اول، را پوشش می‌دهد.
- فصل ۱۱ به یادگیری توضیحی، متدی یادگیری که از دانش قبلی برای توضیح نمونه های آموزشی استفاده می‌کند و با این توضیحات بر روی نمونه های آموزشی تعمیم می‌دهد می‌پردازد.
- فصل ۱۲ روش‌های بهبود دقت فرضیه یادگیری با ترکیب دانش قبلی و نمونه های آموزشی را بحث خواهد کرد. در این فصل هم از الگوریتم‌های نمادی و هم از شبکه های عصبی استفاده خواهد شد.
- فصل ۱۳ به یادگیری تقویتی می‌پردازد، روشی که سیستم باید کارایی خود را طبق پاداش‌هایی که دریافت می‌کند (چه آنی چه با تأخیر) به عنوان اطلاعات آموزشی حداکثر کند. بازی چکرز که پیش‌تر در فصل ۱ بررسی کردیم نمونه ای از همین نوع مسئله است.

۱.۵ خلاصه و منابع برای مطالعه‌ی بیشتر

یادگیری ماشین به سؤالاتی نظیر چگونگی ساخت برنامه های کامپیوتری‌ای که بتوانند کارایی‌شان را در انجام کاری بعد از تجربه افزایش دهند می‌پردازد. نکات کلیدی این فصل شامل موارد زیر می‌شود:

- اثبات شده که الگوریتم‌های یادگیری ماشین ارزش عملی زیادی در بسیاری از زمینه های کاربردی دارند. این الگوریتم‌ها به طور خاص در (a) مسائل کاوش داده در پایگاه‌های داده ای که ممکن است ترتیب‌های محض خاصی را داشته باشد که به صورت اتوماتیک قابل تشخیص هستند (برای مثال، برای بررسی حاصل یک درمان پزشکی بر روی پایگاه داده‌ی بیماران یا یادگیری قوانین کلی بازگشت سرمایه بر روی پایگاه داده‌ی اطلاعاتی)؛ (b) قلمروهایی که انسان دانش کافی برای درک و ارائه‌ی الگوریتم‌های موثر در آن‌ها را ندارند (نظیر تشخیص چهره در عکس) و (c) قلمروهایی که برنامه‌ها باید تطبیق پذیر با محیط در حال تغییر باشند (نظیر فرایند تولید با انبار محدود منابع و یا تشخیص علاقه به مطالب برای فردی که علایق متغیری دارد) کاربرد های زیادی دارند.

---

[1] mistake-bound
[2] Weighted majority

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- یادگیری ماشین به سمت ایده ای از مجموعه‌ی متنوعی از قوانینی شامل هوش مصنوعی، احتمال، آمار، پیچیدگی محاسباتی، تئوری اطلاعات، روانشناسی، تئوری کنترل و فلسفه می‌رود.
- یک مسئله‌ی یادگیری خوش تعریف نیاز به هدف، معیار عملکرد و منبع تجربیات آموزشی ی دقیق تعریف شده[1] دارد.
- طراحی یک روش یادگیری ماشین شامل تعدادی انتخاب‌های طراحی نظیر تعیین نوع تجربیات آموزشی، تابع هدف یادگیری، نمایشی برای این تابع هدف، و الگوریتمی برای یادگیری تابع هدف از تجربیات آموزشی می‌شود.
- یادگیری به نگاه جستجو: جستجو میان فضایی از فرضیه های ممکن برای پیدا کردن فرضیه ای که بهترین تطابق را با نمونه های آموزشی و قیود و دانش اولیه داشته باشد. در اکثر فصول این کتاب بر متد های یادگیری مختلفی است که فضاهای فرضیه ای مختلفی را جستجو می‌کنند تاکید می‌کنیم (برای مثال، فضای توابع عددی یا شبکه های عصبی یا درخت‌های تصمیم یا قوانین نمادین یا ...) و نتایج تئوری‌ای در مورد شرایط همگرایی این متدها به فرضیه‌ی بهینه را بررسی خواهیم کرد.

منابع خوبی برای مطالعه درباره‌ی آخرین تحقیقات در یادگیری ماشین وجود دارد. مجله های مربوط شامل Machine Learning، Neural Computation، Neural Networks، Journal of the American Statistical Association، IEEE Transacations on Pattern Analysis و Machine Intelligence می‌شوند. همچنین همایش‌های سالانه ای که جنبه های مختلفی از یادگیری ماشین بر گزار می‌شود، این همایش‌ها نظیر International Conference on Machine Learning، Neural Information Processing Systems، Conference on Computation Learning Theory، International Conference on Genetic Algorithms، International Conference on Knowledge Discovery and Data Mining، European Conference on Machine Learning و دیگر همایش‌ها هستند.

تمارین

۱.۱. به سه کامپیوتر کاربردهایی دهید که به نظر می‌رسد یادگیری ماشین در آن‌ها غیر کاراست و سه کاربردی که به نظر می‌رسد یادگیری ماشین مناسب است. کاربردهایی را انتخاب کنید که در فصل ذکر نشده‌اند و توجیه کوتاهی برای هر کدام بیاورید.

۱.۲. چند کار یادگیری که در فصل آورده نشده‌اند را انتخاب کنید. به طور غیر رسمی این کارها را در پاراگرافی توصیف کنید. حال این کار را با استفاده از تعیین تا حد ممکن دقیق کار، معیار کارایی و تجربیات آموزشی انجام دهید. تابع هدف و نمایشی برای آن تعیین کنید تا بتوان تابع را یاد گرفت. معیار های اساسی‌ای را که در توصیف دقیق این کار به‌کاربرده این را تعیین کنید.

۱.۳. اثبات کنید که قانون تغییر وزن LMS که در فصل آورده شد از شیب نزول برای رسیدن به مینیمم خطای مربعی استفاده می‌کند. در حالت خاص خطای مربعی E مشابه آنچه در متن فصل آورده شد تعریف می‌شود. حال مشتق E را نسبت به وزن w_i با فرض اینکه $\hat{V}(b)$ تابع خطی تعریف شده در متن درس است حساب کنید. شیب نزول با تغییر هر وزن در جهت $-\frac{\partial E}{\partial w_i}$ است. بنابراین شما باید نشان دهید که قانون آموزش LMS برای هر نمونه‌ی آموزشی وزن‌ها را در این جهت تغییر می‌دهد.

۱.۴. استراتژی‌های مختلفی برای سازنده‌ی تجربه‌ی شکل ۱.۲ در نظر بگیرید. در کل، استراتژی‌هایی را در نظر بگیرید که این سازنده‌ی تجربه چینش‌های صفحه ای را ارائه دهد که:

- چینش صفحه تصادفی باشد (اما ممکن[1] باشد)

[1] well-specified

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

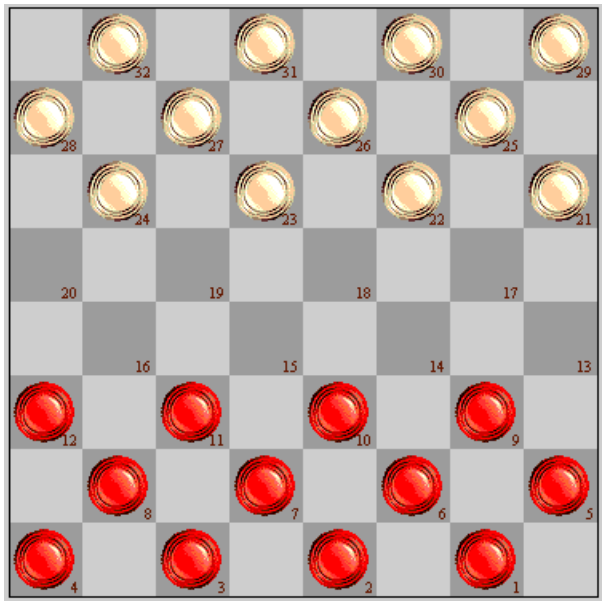
- چینش صفحه با انتخاب یک چینش صفحه از بازی قبل و اعمال چند حرکت که در آن بازی انجام نشده باشد
- استراتژی دلخواه خودتان

معیار های این استراتژی‌ها را با هم مقایسه و بررسی کنید. اگر می‌خواستید یکی از این استراتژی‌ها را برای یادگیری با تعداد تجربه‌ی آموزشی ثابت با معیار عملکرد تعداد بیشتر برد در مسابقات جهانی کدام عملکرد بهتری خواهد داشت؟

۱.۵. الگوریتمی مشابه الگوریتم مطرح شده برای مسئله‌ی بازی چکرز را برای بازی دوز[2] ارائه دهید. تابع یاد گرفته شده‌ی $\hat{V}$ را ترکیب خطی ویژگی‌های دلخواه صفحه در نظر بگیرید. برای آموزش مسئله‌ی خود، از بازی برنامه در مقابل نسخه‌ی دومی از همان برنامه که از معیار عملکرد ثابتی که دست نویس خودتان باشد استفاده کنید. نمودار درصد بازی‌های برده را بر حسب تعداد بازی انجام شده در مرحله‌ی آموزش را رسم کنید.

۱.٦ بازی چکرز (برای خوانندگانی که با این بازی آشنایی ندارند) (اضافه شده توسط مترجم)

بازی چکرز یک بازی دو نفره است که دو طرف، روی یک صفحه‌ی مربع با طول و عرض ۸ خانه و به نوبت با همدیگر بازی می‌کنند. در شکل زیر صفحه ای صفحه‌ی بازی چکرز را مشاهده می‌کنید:



علائم و نشانه‌هایی از این بازی را در آثار به جا مانده از مصر باستان مربوط به ۱۶۰۰ سال قبل از میلاد مسیح را دید. بازی‌هایی به این سبک و در شکل‌های گوناگون از زمین، را می‌توان در نقاط گوناگون از دنیا مشاهده کرد.

---

[1] legal
[2] tic-tac-toe

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

برخی از قوانین ابتدایی این بازی به این شکل است:

- هر بازیکن دارای ۱۲ مهره است که در خانه های تیره چیده شده‌اند.
- بازیکنان هر تیم فقط می‌توانند بر روی خانه های تیره حرکت کنند.
- حرکات به صورت ضربدی انجام می‌شود.
- اگر مهره‌ی حریف در مسیر حرکت قرار گیرد اگر خانه‌ی بعدی خالی باشد می‌توان با پرش از روی مهره‌ی حریف آن مهره را زد. اگر بازیکنی بتواند مهره‌ی حریف را بزند نمی‌تواند حرکت دیگری انجام دهد.
- می‌توان در یک حرکت بیش از یک مهره‌ی حریف را به صورت پشت سر هم زد.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

experience	تجربه
recogonize handwritten words	تشخیص دستخط
well-posed	خوش وضع
performance	کارایی
learning	یادگیری
Task	کار

www.farabar.net

فصل دوم: یادگیری مفهوم و ترتیب کل به جزء

مسئله‌ی استقرای توابع کلی با داشتن تعدادی نمونه خاص، هدف اصلی یادگیری است. در این فصل به یادگیری مفهوم[1] (پی بردن به تعریف یک رسته[2] از اشیا یا اتفاقات با داشتن تعداد محدودی نمونه مثبت و منفی) می‌پردازیم. یادگیری مفهوم را می‌توان جستجو میان فرضیه های از پیش تعریف شده برای پیدا کردن مطابق‌ترین فرضیه با نمونه‌ها دانست. در بسیاری موارد این جستجو را می‌توان با بهره گیری از خاصیتی ذاتی در فضای فرضیه ای (ترتیب جز به کل فرضیه‌ها) سازمان دهی کرد. این فصل الگوریتم‌های یادگیری بسیاری را معرفی خواهد کرد و همچنین شرایطی را که این جواب این الگوریتم‌ها به فرضیه‌ی درست میل می‌کنند را بررسی می‌کند. در ادامه به یادگیری استقرایی می‌پردازیم و توجیه اینکه چگونه ممکن است برنامه‌ها بتوانند داده‌ها را برای نمونه های دیگر تعمیم دهند را بررسی خواهیم کرد.

۲.۱ مقدمه

قسمت عمده ای از یادگیری منوط به یادگیری مفهومی کلی از روی نمونه های آموزشی محدود است. مردم، به عنوان مثال، مفاهیم کلی‌ای مثل "پرنده"، "ماشین" و "وضعیتی که برای قبولی نیاز به بیشتر درس خواندن دارم" و ... را یاد می‌گیرند. هر مفهوم را می‌توان به عنوان زیر مجموعه ای از یک مفهوم کلی‌تر از اشیا یا اتفاقات در نظر گرفت. (برای مثال، مجموعه‌ی پرندگان زیر مجموعه‌ی حیوانات قرار می‌گیرد). همچنین، هر مفهوم را می‌توان به عنوان تابعی منطقی مقدار بر روی مجموعه‌ی بزرگ‌تر در نظر گرفت (برای مثال، روی مجموعه‌ی حیوانات مقادیر تابع برای پرندگان درست و برای دیگر حیوانات غلط است).

در این فصل، به استنتاج تعریف کلی یک مفهوم با استفاده از نمونه‌های موجود (که بعضی عضو مفهومند و بعضی دیگر عضو مفهوم نیستند) می‌پردازیم. به این کار در حالت کلی **یادگیری مفهوم** یا تخمین تابع منطقی از نمونه‌ها می‌گویند.

[1] concept
[2] category

یادگیری مفهوم: استنتاج مقادیر تابع منطقی با یادگیری از نمونه‌هایی از ورودی و خروجی تابع.

۲.۲ کار یادگیری مفهوم

برای درک بهتر، مفهوم "روزهایی که الدو از آبتنی لذت می‌برد" را در نظر بگیرید. جدول ۲.۱ چند روز مختلف را با ویژگی‌هایشان[1] نشان می‌دهد. ویژگی EnjoySport اینکه الدو از آبتنی در آن روز از آبتنی لذت برده یا خیر را مشخص می‌کند. هدف یادگیری پیش‌بینی مقدار ویژگی EnjoySport با دانستن دیگر ویژگی‌ها یک روز است.

در این یادگیری یادگیر از چه نمایشی برای فرضیه‌ها باید استفاده کند؟ بیایید برای فرضیه‌ها یک نمایش ساده در نظر بگیریم؛ فرض می‌کنیم که هر فرضیه عطفی از قید روی چندین ویژگی از ویژگی‌های موجود باشد. به طور دقیق‌تر ویژگی‌ها را به صورت شش تایی مرتبی در نظر می‌گیریم که مقادیر شش ویژگی ما را معلوم کنند. این شش ویژگی به ترتیب: Sky، AirTemp، Humidity، Wind، Water و Forecast هستند. هر یک ویژگی‌ها می‌توانند یکی از حالات ممکن زیر را داشته باشد:

مقدار "?" : یعنی هر مقداری را می‌تواند داشته باشد

مقدار مشخص خاصی داشته باشد (برای مثال: AirTemp ممکن است Warm باشد)

مقدار "0" : یعنی هیچ مقداری برای این ویژگی قابل قبول نیست.

اگر نمونه x تمام قیود فرضیه‌ی h را تأمین کند، می‌توان گفت h، x را به عنوان یک نمونه مثبت دسته بندی می‌کند (h(x) = 1). برای درک بهتر، فرض اینکه الدو فقط در روزهای Cold و مرطوب از آبتنی لذت می‌برد (مجزا از اینکه بقیه ویژگی‌ها چه باشند) به صورت زیر نمایش داده می‌شود:

<?,Cold,High,?,?,?>

شماره‌ی نمونه	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
۱	Sunny	Warm	Normal	Strong	Warm	Same	Yes
۲	Sunny	Warm	High	Strong	Warm	Same	Yes
۳	Rainy	Cold	High	Strong	Warm	Change	No
۴	Sunny	Warm	High	Strong	Cool	Change	Yes

جدول ۲.۱ نمونه‌های مثبت و منفی یادگیری مفهوم EnjoySport.

کلی‌ترین فرضیه، اینکه وی روز از آبتنی لذت می‌برد، به صورت زیر نمایش داده می‌شود:

<?,?,?,?,?,?>

[1] attribute

و در نقطه‌ی مقابل فرضیه‌ی اینکه در روزی مقدار EnjoySport بله نیست به صورت زیر نمایش داده می‌شود:

<0,0,0,0,0,0>

به طور خلاصه، عمل یادگیری مفهوم EnjoySport پیدا کردن و توصیف روزهایی (در قالب گفته شده) که EnjoySport = Yes است. در کل، هر گونه مسئله‌ی یادگیری مفهوم را می‌توان با نمایی که تابع هدف بر روی آن‌ها تعریف شده است، تابع هدف، دسته ای از فرضیه های موجود که یادگیر[1] در نظر می‌گیرد و مجموعه‌ی نمونه های آموزشی موجود مشخص کرد. مسئله‌ی کلی یادگیری مفهوم EnjoySport با این شکل کلی مذکور در جدول ۲.۲ آمده است.

۲.۲.۱ نماد گذاری

در تمام کتاب، زمانی که بحث، مسایل یادگیری مفهوم است، از نماد گذاری‌ای که توضیح داده می‌شود استفاده خواهد شد. دسته اجسامی که مفهوم روی آن تعریف می‌شود "مجموعه‌ی نمونه‌ها"[2] نامیده می‌شود و با نماد X مشخص می‌شود. در مثال مذکور، X تمام‌های ممکن است با ویژگی‌های Sky، AirTemp، Humidity، Wind، Water، Forecast مشخص می‌شود. مفهومی که به دنبال یادگیری آن هستیم "مفهوم هدف"[3] نامیده و با c نمایش داده می‌شود. در کل، c هر مقدار منطقی است که X به عنوان خروجی می‌دهد و به زبان ریاضی داریم: $c:X \to \{0,1\}$. در مثال مذکور مقدار c همان مقدار EnjoySport است (c(x)=1 اگر مقدار EnjoySport=Yes باشد و c(x)=0 اگر مقدارEnjoySport=No باشد).

- معلومات:
  - نمونه های X: تمامی حالت‌های روزهای ممکن
    - Sky (مقادیر ممکن: Sunny، Cloudy و Rainy)
    - AirTemp (مقادیر ممکن: Warm و Cold)
    - Humidity (مقادیر ممکن: Normal و High)
    - Wind (مقادیر ممکن: Strong و Weak)
    - Water (مقادیر ممکن: Warm و Cool)
    - Forecast (مقادیر ممکن: Same و Change)
  - مجموعه فرضیه های H: هر فرضیه با یک شش تایی مرتب ازم تغییر های Sky، AirTemp، Humidity، Wind، Water، Forecast توصیف می‌شود. مقدار هر متغیر می‌تواند "?" (هر مقدار ممکن)، "0" (مقداری ممکن نیست) و یا یک مقدار خاص باشد.
  - مفهوم هدف: EnjoySport c : $: X \to \{0,1\}$
  - نمونه های یادگیری: نمونه های مثبت و منفی تابع هدف. (جدول ۲.۱)

---

[1] learner
[2] set of instances
[3] target concept

- مجهولات:
  - فرضیه‌ی h عضوی از H است اگر که برای تمامی x ما داشته باشیم h(x)=c(x)

جدول ۲.۲ کار یادگیری مفهوم EnjoySport

در هنگام یادگیری، به یادگیر مجموعه ای از نمونه های آموزشی با مقدار تابع هدفشان ارائه می‌شود (x و c(x)) (جدول ۲.۱) که x ها عضو X هستند. نمونه‌هایی که در آن‌ها مقدار c(x)=1 نمونه مثبت[1] یا عضو مفهوم هدف[2] نامیده می‌شود. در مقابل، نمونه‌هایی که در آن‌ها مقدار c(x)=0 نمونه منفی[3] یا غیر عضو مفهوم هدف[4] نامیده می‌شود. گاهی برای راحتی کار از زوج مرتب <x,c(x)> برای نمایش نمونه های آموزشی استفاده می‌شود. مجموعه‌ی نمونه های آموزشی را با حرف D نشان می‌دهیم.

از یادگیر انتظار می‌رود که با داشتن نمونه‌هایی از عملکرد c که آن‌را فرضیه سازی[5] کند یا تخمین بزند. مجموعه‌ی تمامی فرضیه های ممکن را با حرف H نشان می‌دهیم. H معمولاً توسط کاربر انسانی و در انتخاب نوع نمایش فرضیه تعیین می‌گردد. در کل، هر فرضیه‌ی h در H یک تابع منطقی مقدار است که $h: X \rightarrow \{0,1\}$. هدف یادگیر پیدا کردن h ای است که برای تمام مقادیر x در X، h(x)=c(x).

۲.۲.۲ یادگیری استقرایی فرضیه

باید توجه داشت که کار یادگیری پیدا کردن فرضیه h ای است که برای تمامی x های X مشابه مفهوم هدف کار کند در حالی که تنها اطلاعات موجود در مورد c فقط تعداد محدودی نمونه است که در اختیار یادگیر قرار گرفته می‌شود. بنابراین استفاده از الگوریتم‌های یادگیری استقرایی حداکثر تضمین می‌کنند که در نمونه های آموزشی مقدار فرضیه با مقدار تابع هدف یکی است. کمبود اطلاعات باعث می‌شود که فرض کنیم بهترین فرضیه همان فرضیه است که به بهترین شکل با نمونه های موجود مطابقت دارد[6]. این فرض، فرض اساسی یادگیری استقرایی است، در ادامه کتاب مفصلاً درباره‌ی این فرض بحث خواهیم کرد. فعلاً در این قسمت به طور غیر رسمی این فرض را تعریف می‌کنیم اما در فصل‌های ۵، ۶ و ۷ رسمی‌تر این فرض را بررسی می‌کنیم.

یادگیری استقرایی فرضیه: هر فرضیه ای که در مجموعه ای به اندازه‌ی کافی بزرگ از نمونه های آموزشی تابع هدف را خوب تخمین بزند می‌تواند در نقاط دیگر نیز تابع هدف را خوب تخمین می‌زند.

۲.۳ یادگیری مفهوم با دید جستجو

از نظری می‌توان یادگیری مفهوم را جستجویی بین تمام فرضیه های موجود (H) دانست. هدف از این جستجو پیدا کردن فرضیه ای است که به بهترین وجه ممکن رفتار تابع هدف را در نمونه های موجود تخمین بزند. مهم است که بدانیم که طراح با انتخاب نوع نمایش فرضیه، تمام فرضیه های موجود (H)، فرضیه‌هایی را که الگوریتم می‌تواند نمایش بدهد و در کل یاد بگیرد را نیز محدود می‌کند. دوباره به مثال EnjoySport برمی گردیم، تمام فرضیه های موجود H و دسته نمونه های ممکن X را در نظر بگیرید. Sky ۳ حالت ممکن و بقیه ویژگی‌ها

[1] positive example
[2] member of targent concept
[3] negative example
[4] nonmember of targent concept
[5] hypothesize
[6] best fit

هر کدام دو حالت ممکن دارند، پس در کل تعداد حالات ممکن برای X، 3x2x2x2x2x2=96 خواهد بود. با محاسباتی ساده می‌توان (با در نظر گرفتن "?" و "0") گفت که کل تعداد فرضیات ممکن 5x4x4x4x4x4x4=5120 خواهد بود. اما می‌دانیم فرضیاتی که یک یا چند "0" دارند کلی هستند چون همه‌ی مقادیر را 0 پیش بینی می‌کنند ("0" بودن یک مقدار به معنی این است که در هیچ حالتی از این ویژگی مقدار تابع ۱ نمی‌شود). پس تعداد واقعی کل فرضیات برابر است با 1+(4x3x3x3x3x3)=973. توجه داشته باشید که مثال EnjoySport، یک مثال بسیار ساده از یادگیری مفهوم است و متناسباً تعداد کل فرضیات ممکن کمی نیز دارد در حالی که در اکثر مثال‌های واقعی این تعداد بسیار زیاد و گاهی اوقات نیز نامتناهی است.

با دید جستجو به مسئله، طبیعی است که مطالعه بر روی الگوریتم‌های یادگیری مفهوم به مطالعه بر روی الگوریتم‌های جستجو بر روی فرضیه‌ها تبدیل شود. علاقه‌ی ما به الگوریتم‌هایی خواهد بود که به طور موثر و سریع بتوانند تعداد زیاد و حتی نامتناهی از فرضیه‌ها را بررسی کنند تا بهترین فرضیه را برای نمونه های موجود پیدا کنند.

۲.۳.۱ ترتیب کل به جزء فرضیه‌ها

بسیاری از الگوریتم‌های یادگیری مفهوم جستجوی بین فرضیه‌ها را با یک ساختار مفید سازمان دهی می‌کنند. این سازمان دهی برای تمام مسایل یادگیری مفهوم به کار می‌رود: ترتیب کل به جزء فرضیه‌ها. با استفاده از این ساختار طبیعی می‌توان الگوریتم‌هایی طراحی کرد که بدون بررسی تک تک فرضیه‌ها می‌توانند تمام فرضیه‌ها را بررسی کنند. با این نوع الگوریتم‌ها می‌توان حتی هنگامی که اندازه‌ی H نامتناهی است به جواب رسید. برای مثال دو فرضیه زیر را در نظر بگیرید:

$$h_1 = < \text{Sunny}, ?, ?, \text{Strong}, ?, ? >$$

$$h_2 = < \text{Sunny}, ?, ?, ?, ?, ? >$$

حال نمونه های درون این دو فرضیه در نظر بگیرید. چون h_2 قیود کمتری دارد پس تعداد بیشتری نمونه درون آن وجود خواهد داشت در واقع هر نمونه ای که در h_1 باشد در h_2 نیز هست. بنابراین می‌گوییم h_2 از h_1 کلی‌تر[1] است.

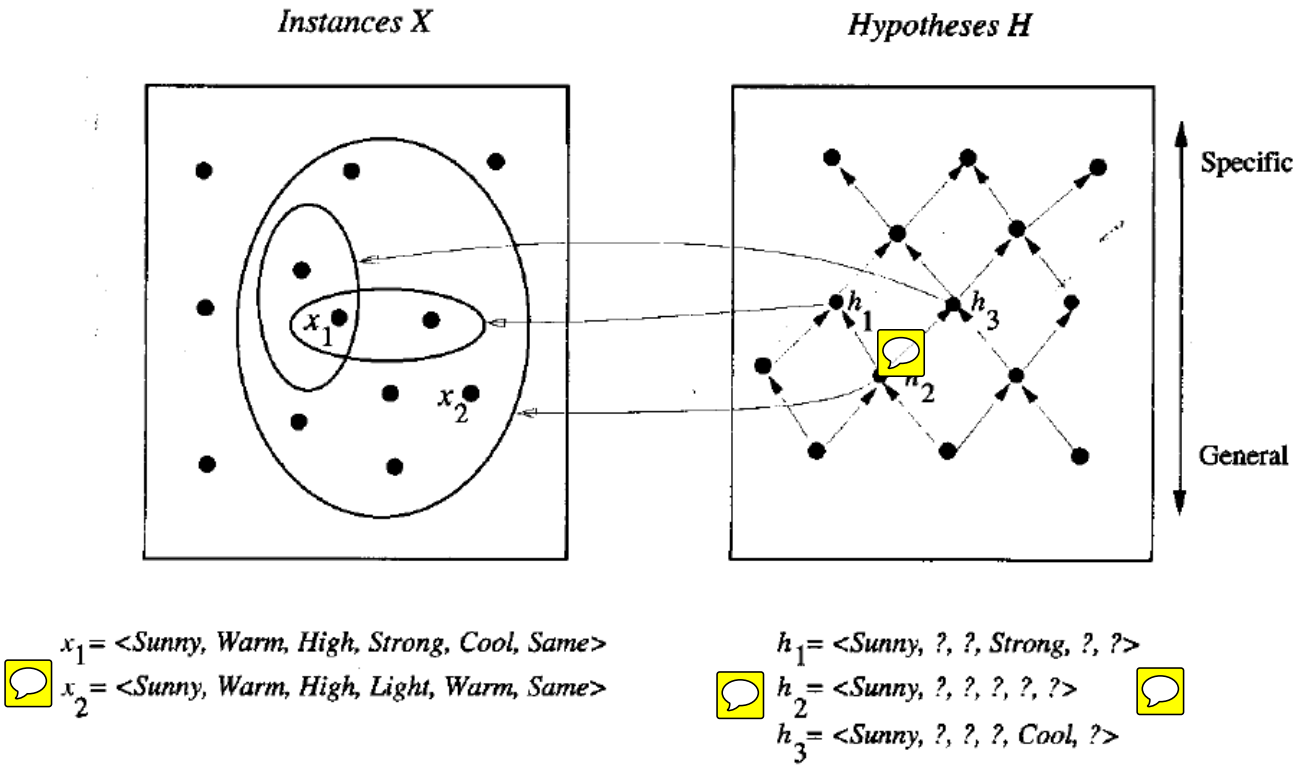
رابطه‌ی ذاتی "کلی‌تر یا مساوی بودن" بین فرضیه‌ها را می‌توان به صورت دقیق‌تر نیز تعریف کرد. اول، برای هر نمونه x در X و هر فرضیه‌ی h در H می‌گوییم x، h را راضی می‌کند اگر و فقط اگر h(x)=1. حال رابطه‌ی "کلی‌تر یا مساوی بودن" را بر اساس نمونه‌هایی که فرضیه‌ها را راضی می‌کنند تعریف می‌کنیم. برای دو فرضیه‌ی h_j و h_k داریم: h_j کلی‌تر از h_k است اگر و فقط اگر هر نمونه ای که h_k را راضی کرد h_j را نیز راضی کند.

تعریف: اگر h_j و h_k دو تابع منطقی مقدار تعریف شده روی X باشند h_j کلی‌تر یا مساوی است با h_k اگر و فقط اگر $(h_j \geq_g h_k)$

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

[1] more general

بعضی مواقع لازم است فرضیه ای به طور اکید از فرضیه دیگر کلی‌تر باشد. فرضیه‌ی h_j (اکیداً) کلی‌تر است از h_k $(h_j >_g h_k)$ اگر و فقط اگر $(h_j \geq_g h_k) \wedge (h_k \ngeq_g h_j)$. بعضی مواقع نیز لازم است برعکس بالا بگوییم که h_k خاص‌تر[1] از h_j است اگر و تنها اگر $(h_j >_g h_k)$.



شکل ۲.۱ نمونه‌ها و فرضیه‌ها و رابطه‌ی کلی‌تری.

مربع سمت چپ نشان دهنده‌ی X یا همان تمامی نمونه‌هاست. و مربع سمت راست نشان دهنده‌ی H یا همان تمامی فرضیه‌هاست. هر فرضیه متناسب با زیر مجموعه ای از X است (همان زیر مجموعه ای که آن را راضی می‌کند). فلش‌های بین فرضیه‌ها رابطه‌ی خاص‌تر بودن را نشان می‌دهد (سر انتهایی فلش‌ها خاص‌ترند). توجه داشته باشید که مجموعه‌ی متناسب با $\mathbf{h_2}$، مجموعه‌ی متناسب با $\mathbf{h_1}$ را شامل می‌شود پس فرضیه‌ی $\mathbf{h_2}$ از فرضیه‌ی $\mathbf{h_1}$ کلی‌تر است.

برای درک بهتر، سه فرضیه‌ی h_1، h_2 و h_3 را در همان مثال EnjoySport در نظر بگیرید (شکل ۲.۱). رابطه‌ی $\geq_g$ بین این سه فرضیه چگونه است؟ همان طور که قبلاً نیز گفته شد h_2 از h_1 کلی‌تر است زیرا هر نمونه ای که h_2 را راضی کند h_1 را نیز راضی می‌کند. به طور مشابه h_2 از h_3 نیز کلی‌تر است. توجه داشته باشید که هیچ کدام از فرضیه های h_1 و h_3 کلی‌تر از دیگری نیست. با وجود اینکه در نمونه‌هایی اشتراک دارند اما هیچ کدام دیگری را شامل نمی‌شود. توجه داشته باشید که دو رابطه‌ی $\geq_g$ و $>_g$ مستقل از اینکه مفهوم هدف چه باشد تعریف شده‌اند و فقط بر اساس اینکه کدام نمونه‌ها در درون فرضیه قرار می‌گیرند تعریف شده‌اند و نه بر اساس تابع هدف. به طور رسمی، رابطه‌ی $\geq_g$ ترتیب خاصی را در درون فضای فرضیه‌ها H ایجاد می‌کند (این رابطه، بازتابی، پاد متقارن و انتقالی است). به طور غیر رسمی،

[1] more specific

زمانی که می‌گوییم یک ساختار جزئی مرتب[1] (در مقابل کلی[2]) است، منظورمان این است که ممکن است جفت فرضیه‌هایی مثل h_1 و h_3 وجود داشته باشند که $h_3 \not\geq_g h_1$ و $h_1 \not\geq_g h_3$.

اهمیت رابطه‌ی $\geq_g$ در این است که ساختار مفیدی برای هر مسئله‌ی یادگیری مفهوم بر روی فضای فرضیه‌ها (H) ایجاد می‌کند. قسمت بعدی به الگوریتمی که با استفاده از این ساختار جستجو را سازمان دهی می‌کند، می‌پردازد.

---

۱. h را خاص‌ترین فرضیه‌ی H در نظر بگیر
۲. برای هر نمونه مثبت x
   - برای هر ویژگی a_i در h
     اگر x، a_i را راضی می‌کند کاری انجام نده
     در غیر این صورت در h از خاصیت a_i به سمت کلی‌تر شدن برو (قید کلی‌تری که نمونه را شامل می‌شود را در این ویژگی جایگزین کن).
۳. فرضیه h را خروجی بده.

جدول ۲.۳ الگوریتم FIND-S

۲.٤ FIND-S : پیدا کردن خاص‌ترین فرضیه

چگونه می‌توان از رابطه‌ی کلی‌تری برای سازمان دهی جستجوی بین فرضیه‌ها استفاده کرد؟ یک راه شروع کردن از خاص‌ترین فرضیه درون فضای فرضیه‌ها H و کلی‌تر کردن آن در مراحلی که نمی‌تواند نمونه‌ها را بپوشاند است. (زمانی که یک فرضیه یک نمونه مثبت را می‌پوشاند که آن‌را شامل شود). برای بهتر روشن شدن این مطلب الگوریتم FIND-S (جدول ۲.۳) در نظر بگیرید.

برای تصور بهتر فرض کنیم که به یادگیر مقادیر جدول ۲.۱ داده شده تا مفهوم EnjoySport را یاد بگیرد. گام اول الگوریتم مقدار دهی اولیه‌ی h با خاص‌ترین فرضیه است.

h ← <0,0,0,0,0,0>

بعد به سراغ اولین داده‌ی جدول ۲.۱ می‌رود، چون این داده مقدار مثبت این فرضیه نیست معلوم می‌شود که فرضیه بیش از حد خاص است. چون نمونه هیچ یک از مقادیر "0" فرضیه را راضی نمی‌کند پس این مقادیر با مقادیر کلی‌تری جایگزین می‌شوند.

h ← <Same,Warm,High, Normal, Warm,Sunny>

---

[1] partial order
[2] total

www.farabar.net

اما با این حال این h بیش از حد خاص است زیرا جز به همان نمونه اول، نمونه مثبت دیگری ندارد. زمانی که به نمونه آموزشی دوم می‌رسد (که در اینجا یک نمونه مثبت است)، دوباره الگوریتم مجبور می‌شود که h را تعمیم‌تر کند و ویژگی سوم را "?" قرار دهد (ویژگی‌ای که در این نمونه پوشانده نشده بود) پس:

h ← <Same,Warm,High,?, Warm,Sunny>

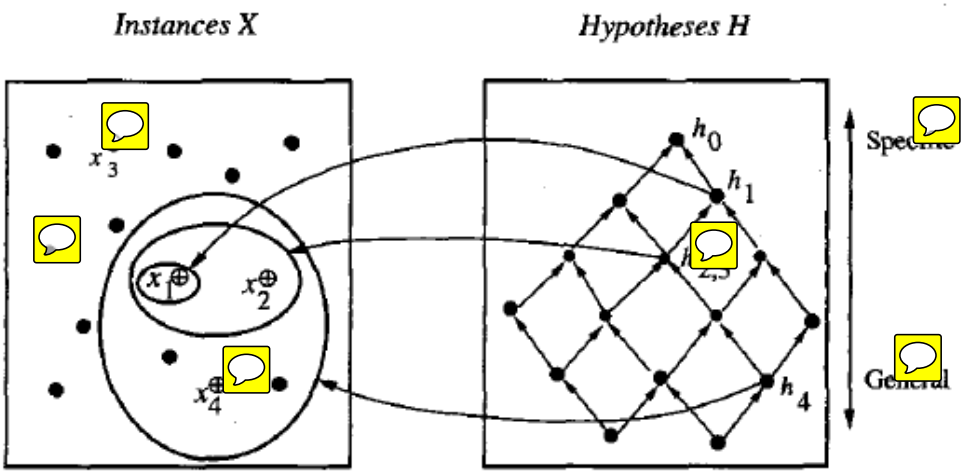
با رسیدن به نمونه سوم (که یک نمونه منفی است) نیاز نیست که تغییر خاصی به h بدهد. در واقع الگوریتم FIND-S هیچ کاری در قبال نمونه های منفی نمی‌کند! با وجود اینکه این رفتار الگوریتم کمی عجیب به نظر می‌آید توجه داشته باشید که الآن این نمونه از h جواب منفی می‌گیرد (h این نمونه را درست دسته بندی کرده) پس به هر حال تغییری لازم نیست. در حالت کلی، تا زمانی که فرض کنیم در فضای فرضیه‌ها (H)، فرضیه ای وجود دارد که c را به طور کامل توصیف می‌کند و نمونه های آموزشی درست هستند نیازی به تغییر در نمونه های منفی نیست. دلیل این امر آن است که فرضیه‌ی h خاص‌ترین فرضیه‌ی ممکن در H است که بر اساس نمونه های مثبت ساخته شده، و چون فرض می‌کنیم که c نیز در H وجود دارد و نمونه های مثبت را نیز در بر دارد پس مطمئناً c کلی‌تر یا مساوی h است. و چون c با نمونه‌ی منفی‌ای را شامل نمی‌شود، h نیز آن‌را شامل نخواهد شد. پس هیچ نیازی به تغییر در نمونه های منفی نداریم.

برای کامل کردن الگوریتم FIND-S چهارمین داده را نیز بررسی می‌کنیم و داریم که:

h ← {"؟" ,"؟",High,"؟" , Warm,Sunny}

الگوریتم FIND-S نمونه ای از استفاده‌ی از ویژگی کلی‌تری برای جستجوی فضای فرضیه‌ها در پیدا کردن فرضیه مطلوب است. جستجو از فرضیه ای خیلی خاص شروع و با کلی‌تر کردن فرضیه‌ها در یک زنجیره‌ی کلی‌تر شدن ادامه پیدا می‌کند. شکل ۲.۲ این جستجو را در درون فضای فرضیه ای و فضای نمونه‌ها نشان می‌دهد. در هر مرحله فقط به اندازه‌ی لازم برای پوشش نمونه جدید فرضیه کلی‌تر می‌شود. پس در هر مرحله فرضیه‌ی h خاص‌ترین فرضیه‌ی ساخته شده بر روی نمونه های مثبت قبلی است. (S در نام FIND-S نیز از همان کلمه‌ی specific گرفته شده). تحقیقات یادگیری مفهوم پر از الگوریتم‌هایی مشابهی است که ترتیب کلی‌تری را به سبکی برای سازمان دهی جستجو استفاده کرده‌اند. تعدادی از این الگوریتم‌ها را در این فصل و تعدادی دیگر را در فصل ۱۰ بررسی می‌کنیم.

www.farabar.net



x_1 = <Sunny Warm Normal Strong Warm Same>, +
x_2 = <Sunny Warm High Strong Warm Same>, +
x_3 = <Rainy Cold High Strong Warm Change>, -
x_4 = <Sunny Warm High Strong Cool Change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny Warm Normal Strong Warm Same>
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

شکل ۲.۲ فرضیه‌هایی که در طی مراحل الگوریتم FIND-S به دست آمد.

جستجو از فرضیه‌ی $\boldsymbol{h_0}$ شروع می‌شود که خاص‌ترین فرضیه در H است سپس پله به پله تحت تاثیر نمونه‌ها کلی‌تر می‌شود (از $\boldsymbol{h_1}$ تا $\boldsymbol{h_4}$). در نمودار نمونه‌ها (سمت چپ) نمونه های مثبت با علامت "+"، نمونه های منفی با علامت "-" و نمونه‌هایی که جزو های آموزشی نبوده‌اند با دایره های توپر نشان داده شده‌اند.

ویژگی کلیدی الگوریتم FIND-S این است که در میان فضای فرضیه‌ها (H) تضمین می‌کند که خاص‌ترین فرضیه را بر اساس نمونه های مثبت ارائه دهد. با فرض اینکه نمونه‌ها درست باشند و c نیز در H موجود باشد، خروجی الگوریتم FIND-S برای نمونه های منفی مقدار صفر می‌دهد. با این وجود تعدادی از سؤالات موجود بی جواب می‌مانند:

- آیا یادگیر به سمت مفهوم هدف همگرا شده؟ با وجود اینکه الگوریتم FIND-S فرضیه ای را پیدا می‌کند که با تمام نمونه های آموزشی مطابقت داشته باشد، اما تضمین نمی‌کند که فرضیه پیدا شده یکتا باشد و ممکن است فرضیه های دیگری در H موجود باشند که با نمونه‌ها مطابقت داشته باشند. ترجیح ما بر این است که از الگوریتم‌هایی استفاده کنیم که مشخص کنند آیا به فرضیه مشخصی همگرا شده‌اند و اگر نه، مشخص کنند که میزان عدم قطعیت چقدر و چگونه است.
- چرا دنبال خاص‌ترین فرضیه هستیم؟ زمانی که به الگوریتم FIND-S نمونه‌هایی داده می‌شود خروجی خاص‌ترین فرضیه‌ی ممکن خواهد بود. معلوم نیست که چرا دنبال کلی‌ترین فرضیه یا چیزی بینابین نمی‌گردیم و فقط دنبال خاص‌ترین فرضیه هستیم.
- آیا همیشه نمونه های یادگیری بدون خطا هستند؟ در بسیاری از مسایل یادگیری مفهوم امکان وجود خطا یا نویز[1] در نمونه های یادگیری وجود دارد. نمونه‌هایی که خطا دارند کاملاً FIND-S را به اشتباه می‌اندازند، مخصوصاً اینکه FIND-

[1] noise

S در قبال نمونه های منفی هیچ عکس‌العملی انجام نمی‌دهد. ما ترجیح می‌دهیم از الگوریتم‌هایی استفاده کنیم که خطا داشتن نمونه‌ها را تشخیص بدهند و ترجیحاً بتوانند خود را با این خطاها تطبیق دهند.

- اگر خاص‌ترین فرضیه یکتا نبود چه؟ در مثال EnjoySport همیشه خاص‌ترین فرضیه‌ی ساخته شده روی نمونه‌ها یکتا بود. با این حال، در فضا های فرضیه ای دیگر (که بعداً درباره‌ی آن‌ها بحث خواهیم کرد) ممکن است خاص‌ترین فرضیه یکتا نباشد. در چنین شرایطی، الگوریتم FIND-S باید تصحیح شود تا بتواند گزینه های دیگر موجود در کلی‌تر سازی و احتمال این را که آیا می‌شود از شاخه ای دیگر از روند کلی سازی به مفهوم هدف رسید بررسی کند. در آینده، فضا های فرضیه ای را معرفی خواهیم کرد که در آن‌ها همیشه خاص‌ترین فرضیه‌ی موجود یکتا نیست، با این حال این نوع فضا های فرضیه ای بیشتر تئوری‌اند تا عملی.

۲.۵ فضا های ویژه و الگوریتم Candidate-Elimination

در این بخش به الگوریتم دیگری در یادگیری مفهوم به نام الگوریتم Candidate-Elimination می‌پردازیم که ضعف‌های FIND-S را ندارند. توجه داشته باشید که خروجی الگوریتم FIND-S فقط یکی از فرضیه های داخل H است که با نمونه‌ها مطابقت دارد. نکته‌ی کلیدی الگوریتم Candidate-Elimination هم همین است. این الگوریتم توصیفی از تمامی فرضیه های مطابق با نمونه‌ها می‌دهد. نکته‌ی جالب‌تر این است که الگوریتم Candidate-Elimination برای پیدا کردن دسته فرضیه‌ی مطابق با نمونه‌ها تمامی فرضیه‌ها را بررسی نمی‌کند. این کار بر اساس همان ترتیب کلی‌تری و با استفاده از توصیفی برای مجموعه‌ی شامل تمامی فرضیه های سازگار با نمونه‌ها انجام می‌گردد.

الگوریتم Candidate-Elimination در گذشته برای پیدا کردن رابطه‌ی بین طیف سنجی جرمی در شیمی (Mitchell 1979) و در یادگیری قوانین جستجوی اکتشافی[1] (Mitchell 1983) استفاده شده است. اما با این حال در کاربرد های واقعی، به دلیل اینکه هر دو الگوریتم FIND-S و Candidate-Elimination در مواجهه با نمونه‌هایی که خطا و نویز دارند عملکرد ضعیفی دارند، کاربرد زیادی ندارند. از آن مهم‌تر، برای هدف ما ، الگوریتم Candidate-Elimination قالبی مفهومی را برای معرفی بسیاری از مطالب پایه ای در یادگیری ماشین معرفی می‌کند. در ادامه به این الگوریتم و این مطالب خواهیم پرداخت. در فصل‌های بعدی به این الگوریتم‌هایی که با داده های نویز دار نیز درست کار می‌کنند خواهیم پرداخت.

۲.۵.۱ معرفی

الگوریتم Candidate-Elimination تمامی فرضیه های قابل توصیف که در نمونه‌ها صدق می‌کنند را پیدا می‌کند. برای تعریف دقیق الگوریتم، با چند تعریف اولیه شروع می‌کنیم. اول، یک فرضیه با نمونه های آموزشی سازگار است، اگر آن نمونه‌ها را به درستی دسته بندی کند.

تعریف: فرضیه‌ی h با نمونه های D سازگار[2] است اگر و فقط اگر برای هر زوج مرتب <x,c(x)> در D داشته باشیم h(x)=c(x).

$$Consistent(h, D) \equiv (\forall < x, c(x) > \in D)\ h(x) = c(x)$$

[1] heuristic search
[2] consistent

توجه داشته باشید راضی کردن و سازگاری یکی نیستند. برای مثال: x فرضیه‌ی h را راضی می‌کند اگر h(x)=1 در حالی که فرقی نمی‌کند که x یک نمونه مثبت یا منفی باشد. در حالی که نمونه x زمانی با h سازگار است که h(x)=c(x) باشد.

الگوریتم Candidate-Elimination تمامی دسته فرضیه های سازگار با نمونه های آموزشی را خروجی می‌دهد. این دسته فرضیه‌ها فضای ویژه[1] نامیده می‌شود چون تمامی نسخه های قابل قبول مفهوم هدف را شامل می‌شود. فضای ویژه وابسته به فضای فرضیه‌ها (H) و نمونه های آموزشی (D) است.

تعریف: فضای ویژه ، که با $VS_{H.D}$ نمایش داده می‌شود، با توجه به فضای فرضیه‌ها (H) و نمونه های آموزشی (D)، مجموعه‌ی فرضیه‌هایی از H است که با مثال‌های D سازگار است.

$$\boldsymbol{VS_{D.H} \equiv \{h \in H | Consistent(h, D)\}}$$

۲.۵.۲ الگوریتم List-Then-Eliminate

ساده‌ترین راه ممکن برای معرفی فضای ویژه معرفی تک تک عضو های آن است. این نوع معرفی به یک الگوریتم به نام List-Then-Eliminate می‌انجامد (جدول ۲.۴).

الگوریتم List-Then-Eliminate ابتدا فرض می‌کند که تمامی فرضیه‌ها سازگار با نمونه‌ها هستند. یعنی فضای ویژه را با H مقدار دهی اولیه می‌کند. سپس هر فرضیه ای را که با مثال‌ها سازگاری نداشته باشد حذف می‌کند. با بررسی سازگاری تک تک فرضیه‌ها با تک تک نمونه‌ها، فرضیه‌ها از فضای ویژه حذف می‌شوند و در آخر فقط یک فرضیه در فضای ویژه باقی می‌ماند (که همان مفهوم هدف است). اگر تعداد داده‌ها کافی نباشد، در فضای ویژه بیشتر از یک عضو باقی می‌ماند و آن هم دسته فرضیه های سازگار با نمونه‌هاست.

اصولاً، فقط زمانی که H متناهی[2] است می‌توان از الگوریتم List-Then-Eliminate استفاده کرد. این الگوریتم مزیت‌های بسیاری شامل تضمین اینکه که تمامی فرضیه های سازگار با نمونه‌ها را پیدا کند دارد . اما در مقابل بسیار زمان‌گیر است چون باید سازگاری تمامی اعضای H را با تمامی داده‌ها بررسی کرد که جز در فضاهای فرضیه ای بسیار ساده شرطی غیر عملی است.

۲.۵.۳ نمایش فشرده تری از فضاهای ویژه

الگوریتم Candidate-Elimination مشابه الگوریتم List-Then-Eliminate عمل می‌کند. با این تفاوت که از نمایشی دیگر برای فضای ویژه استفاده می‌کند. در این نمایش فضای ویژه با کلی‌ترین و خاص‌ترین فرضیه‌هایش نمایش داده می‌شود. این اعضای مرزی فضای ویژه، نشان دهنده‌ی مکان فضای ویژه در ترتیب کلی‌تری هستند.

الگوریتم List-Then-Eliminate

۱. تمامی فرضیه های H ← VersionSpace
۲. برای هر نمونه آموزشی <x,c(x)>

---

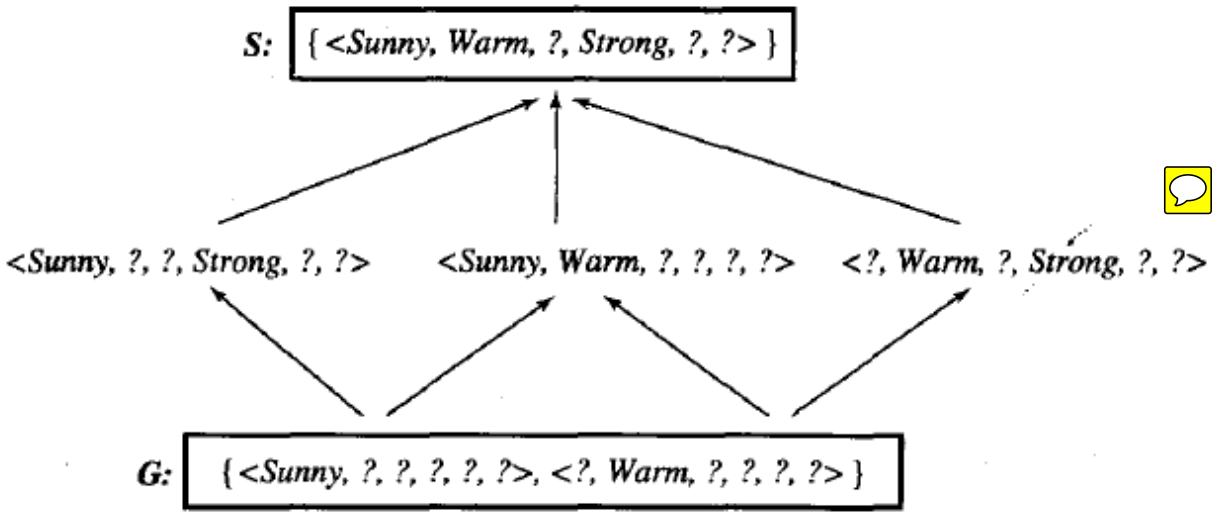
[1] version space
[2] finite

Telegram: @Farabar_BI
www.farabar.net

هر فرضیه ای در VersionSpace مثل h که h(x)≠c(x) را از VersionSpace حذف کن

۳. لیست باقی مانده در VersionSpace را چاپ کن

جدول ۲.۴ الگوریتم List-Then-Eliminate



شکل ۲.۳ یک فضای ویژه با مرزهای خاص‌ترین و کلی‌ترین فرضیه‌ها.
فضای ویژه‌ی فوق هر چهار فرضیه‌ی نشان داده شده را در بر می‌گیرد اما به طور خیلی ساده تر می‌توان آن‌را فقط با S و G نشان داده شده در شکل نمایش داد. فلش‌های شکل فرایند خاص‌تر شدن را نشان می‌دهد (پیکان فلش‌ها به سمت فرضیه های خاص‌تر است). این فضای ویژه برای مفهوم EnjoySport است و نمونه های آموزشی نیز همان نمونه های جدول ۲.۱ است.

برای تصور بهتر از این نمایش جدید فضاهای ویژه دوباره به سراغ مسئله‌ی EnjoySport می‌رویم (جدول ۲.۲). الگوریتم FIND-S برای این مسئله خروجی زیر را داده است:

h = <Sunny,Warm,? ,Strong,?,?>

در واقع این فرضیه فقط یکی از ۶ فرضیه‌ی سازگار موجود در H است. تمامی این ۶ فرضیه در شکل ۲.۳ نشان داده شده‌اند. مجموعه‌ی ۶ فرضیه‌ی سازگار با این نمونه‌ها فضای ویژه است. فلش‌های شکل فرایند خاص‌تر شدن را نشان می‌دهد (پیکان فلش‌ها به سمت فرضیه های خاص‌تر است). الگوریتم Candidate-Elimination فضای ویژه را با تشخیص مرزهای کلی‌تر (که در شکل با حرف G مشخص شده) و مرز های خاص‌تر (که در شکل با حرف S مشخص شده) مشخص می‌کند. با داشتن این دو دسته فرضیه می‌توان تمامی فرضیه های فضای ویژه را با ترتیب کلی‌تری مشخص کرد.

بدیهی است که می‌توان هر فضای ویژه را با خاص‌ترین و کلی‌ترین عضوهایش مشخص کرد. در ادامه دو مرز کلی‌تر و خاص‌تر را تعریف کرده و ثابت می‌کنیم که می‌توان فقط با استفاده از این دو مرز کل فضای ویژه را مشخص کرد.

تعریف: مرز کلی G، با توجه به فضای فرضیه‌ها (H) و نمونه های آموزشی (D) مجموعه کلی‌ترین فرضیه‌ها در H و سازگار با D است.

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[g' >_g g) \wedge Consistent(g', D)]\}$$

www.farabar.net

تعریف: مرز خاص S، با توجه به فضای فرضیه‌ها (H) و نمونه های آموزشی (D) مجموعه خاص‌ترین فرضیه‌ها در H و سازگار با D است.

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[s >_g s') \wedge Consistent(s', D)]\}$$

تا زمانی که دو دسته‌ی G وS خوش تعریف[1] باشند (تمرین ۲.۷)، فضای ویژه را به درستی و کاملاً مشخص می‌کنند. به عبارت دیگر، می‌توان نشان داد که فضای ویژه اجتماع سه مجموعه‌ی G، S و مجموعه‌ی بین آن‌ها در ترتیب کلی‌تری است. این اثبات به طور کامل در قضیه‌ی ۲.۱ آمده.

قضیه‌ی ۲.۱. قضیه‌ی نمایش فضای ویژه. اگر X تمام نمونه‌ها، H تمام فرضیه های روی X، $c: X \rightarrow \{0,1\}$ یک مفهوم هدف دلخواه روی X، D نمونه های آموزشی موجود باشد {<x,c(x)>} و S و G خوش تعریف باشند داریم:

$$VS_{H.D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

اثبات. برای اثبات این قضیه کافی است نشان دهیم که (۱) هر h که در قسمت سمت راست تساوی بالا صدق می‌کند عضو $VS_{H.D}$ است و (۲) هر عضو $VS_{H.D}$ در سمت راست تساوی صدق می‌کند. اثبات قسمت (۱):

فرض کنیم g عضوی از G و s عضوی از S و h عضوی از H باشد به صورتی که $g \geq_g h \geq_g s$. از روی تعریف S داریم که s توسط تمامی نمونه های D را راضی می‌شود و طبق فرض $h \geq_g s$ پس h نیز توسط تمامی نمونه های D را راضی می‌شود.

به طور مشابه طبق تعریف G، g توسط هیچ یک از نمونه های منفی ی D را راضی نمی‌شود و طبق فرض $g \geq_g h$ پس h نیز توسط هیچ یک از نمونه های منفی ی D راضی نمی‌شود.

با توجه به دو قسمت بالا پس h با D سازگار است پس h نیز عضو $VS_{H.D}$ است.

اثبات قسمت دوم کمی پیچیده تر است. باید از برهان خلف استفاده کرد و فرض کرد که فرضیه ای در مثل h در $VS_{H.D}$ وجود دارد که در قسمت سمت راست تساوی صدق نمی‌کند و به تناقض رسید (تمرین ۲.۶).

۲.۵.٤ الگوریتم یادگیری Candidate-Elimination

الگوریتم یادگیری Candidate-Elimination فضای ویژه ای را محاسبه می‌کند که با تمامی نمونه های آموزشی موجود سازگار باشد. مثل الگوریتم List-Then-Eliminate در ابتدای این الگوریتم فضای ویژه را کل H در نظر می‌گیریم. پس G را کلی‌ترین فرضیه در نظر می‌گیریم:

$$G_0 \leftarrow \{<?,?,?,?,?,?,?>\}$$

در مقابل نیز S را خاص‌ترین فرضیه در نظر می‌گیریم:

$$S_0 \leftarrow \{< 0,0,0,0,0,0,0 >\}$$

[1] Well-defined

دو مرز تعیین شده تمامی فرضیه های موجود در H را در بر می‌گیرند، چون تمامی فرضیه‌ها از S_0 کلی‌تر و از G_0 خاص‌ترند. سپس تک تک نمونه‌ها بررسی می‌شوند و S و G، کلی‌تر و خاص‌تر می‌گردند تا فرضیه های ناسازگار با نمونه‌ها را از فضای ویژه حذف کنند. بعد از بررسی کل نمونه‌ها فضای ویژه مشخص می‌شود. خلاصه‌ی این الگوریتم در جدول ۲.۵ آمده.

مقدار اولیه‌ی G را کلی‌ترین فرضیه در H قرار بده

مقدار اولیه‌ی S را خاص‌ترین فرضیه در H قرار بده

برای هر نمونه آموزشی d مراحل زیر را انجام بده

- اگر d نمونه ای مثبت بود
  - هر فرضیه ای که در G با d مطابقت نداشت را حذف کن
  - برای هر فرضیه‌ی s در S که سازگار با d نیست
    - s را از S حذف کن
    - تمامی خاص‌ترین کلی سازی h از s را که در شرط زیر صدق می‌کنند به S اضافه کن
      - h در آن با d سازگار است و حداقل یکی از اعضای G از آن کلی‌تر است
    - هر فرضیه ای که از فرضیه‌ی دیگری در S کلی‌تر بود از آن حذف کن
- اگر d نمونه ای منفی بود
  - هر مثال ناسازگار با d در S را حذف کن
  - برای هر فرضیه‌ی g در G که با d سازگار نیست
    - g را از G حذف کن
    - تمامی کلی‌ترین خاص سازی h از g را که در شرط زیر صدق می‌کنند به G اضافه کن
      - h در آن با d سازگار است و حداقل یکی از اعضای S از آن خاص‌تر است
    - هر فرضیه ای که از فرضیه‌ی دیگری در G خاص‌تر بود از آن حذف کن

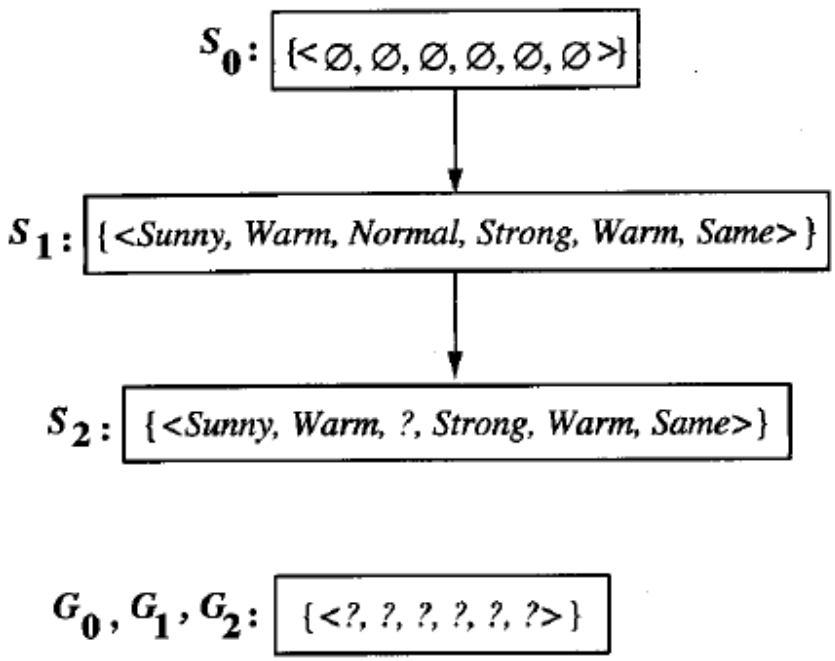
جدول ۲.۵ الگوریتم Candidate-Elimination با استفاده از فضای ویژه.
توجه داشته باشید که نمونه های مثبت و منفی به روش‌های متفاوتی روی S و G اثر می‌گذارند (به طور قرینه).

توجه داشته باشید که در شرح الگوریتم از عملیات‌هایی از جمله پیدا کردن خاص‌ترین کلی سازی ، کلی‌ترین خاص سازی، خاص‌ترین، کلی‌ترین استفاده شده است. اطلاعات جزیی تر در مورد این عملیات به نمونه‌ها و فرضیه‌ها بستگی دارد. با این وجود، خود الگوریتم را می‌توان برای هر مسئله‌ی یادگیری مفهوم و هر فضای فرضیه ای که این عملیات‌ها را بشود رویش تعریف کرد، به کار برد. در ادامه دوباره همان مسئله‌ی EnjoySport را این بار از دید این الگوریتم بررسی می‌کنیم.

۲.۵.۵ یک مثال شهودی

در شکل ۲.۴ حاصل اجرا کردن الگوریتم Candidate-Elimination برای دو مثال اول جدول ۲.۱ آمده است. همان طور که پیش‌تر نیز اشاره شد، در ابتدا فرضیه های مرزی ابتدایی G_0 و S_0 به ترتیب کلی‌ترین و خاص‌ترین فرضیه‌ها هستند.

زمانی که الگوریتم به نمونه اول می‌رسد (یک نمونه مثبت)، الگوریتم S را چک می‌کند و متوجه می‌شود که S بیش از حد خاص است (مثال را پوشش نمی‌دهد). پس مرز تا خاص‌ترین فرضیه ای که نمونه را پوشش دهد کلی می‌شود. این تغییر مرز در شکل ۲.۴ با S_1 نشان داده شده. در این مرحله هیچ تغییری در G لازم نیست، چون G_0 مثال را پوشش می‌دهد. در نمونه آموزشی بعدی (باز هم یک نمونه مثبت)، دوباره S تغییر می‌کند و به S_2 تبدیل می‌شود در حالی که G همچنان بدون تغییر می‌ماند. توجه می‌کنید که تأثیر این دو نمونه‌ی مثبت در الگوریتم Candidate-Elimination مشابه تأثیر آن‌ها در دو مرحله‌ی اول در الگوریتم FIND-S است.

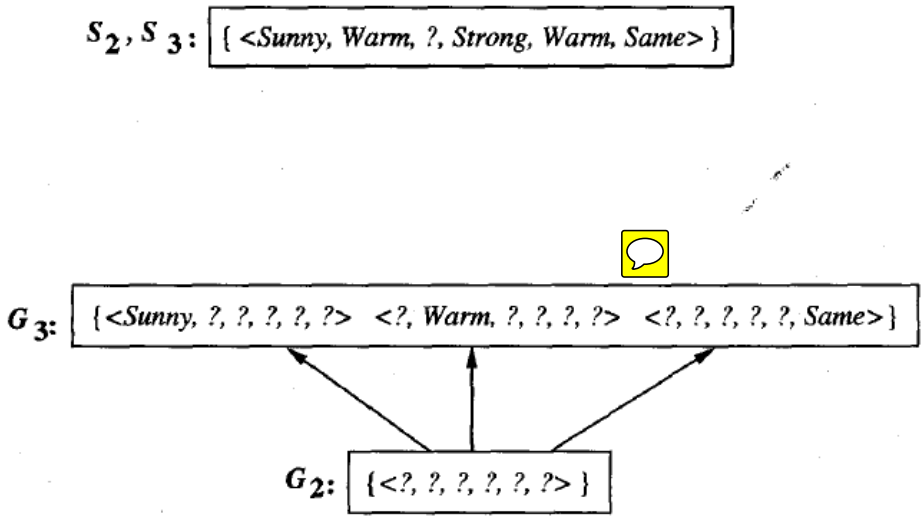


شکل ۲.۴ عملکرد الگوریتم Candidate-Elimination در دو گام اول.
در ابتدا G_0 و S_0 دو مقدار اولیه‌ی G و S به ترتیب کلی‌ترین و خاص‌ترین فرضیه‌ها هستند. دو نمونه اول S را وادار می‌کنند که کلی‌تر شود (درست مثل الگوریتم FIND-S). این دو مثال تأثیری بر G ندارند.

همان طور که در شکل نیز معلوم است دو نمونه اول مرز S را وادار می‌کنند که کلی‌تر شود و فضای ویژه را مشخص‌تر کند. در نقطه‌ی مقابل مثال‌های منفی G را مجبور می‌کنند که تا جای لازم خاص شود. نمونه آموزشی سوم را در نظر بگیرید (شکل ۲.۵). نمونه منفی نشان می‌دهد که G بیش از حد کلی است. چون بدون این خاص سازی پیش بینی می‌شود که این مثال یک نمونه مثبت است! پس باید G تا جایی که این نمونه را درست تشخیص دهد خاص‌تر شود. همان طور که در شکل ۲.۵ نیز نشان داده شده کلی‌ترین خاص سازی‌های بسیاری وجود دارد. همه این فرضیه‌ها عضو مرز جدید G_3 خواهند بود.

چرا با این که می‌دانیم ۶ کلی‌ترین خاص سازی برای G وجود داشت اما با این حال چرا فقط ۳ فرضیه به G_3 اضافه شد؟ برای مثال h = <?,?,Normal,?,?,?> که نمونه منفی را به درستی منفی می‌داند و یک کلی‌ترین خاص سازی است اضافه نشده؟ جواب در این نهفته است که این فرضیه با نمونه های مثبت قبلی سازگار نیست. الگوریتم این نکته را با مقایسه‌ی کلی‌ترین خاص سازی‌ها با S می‌فهمد زیرا که S خلاصه ای از نمونه های مثبت قبلی را در خود ذخیره کرده و می‌توان با آن مشخص کرد که آیا کلی‌ترین خاص سازی‌ها با نمونه های مثبت

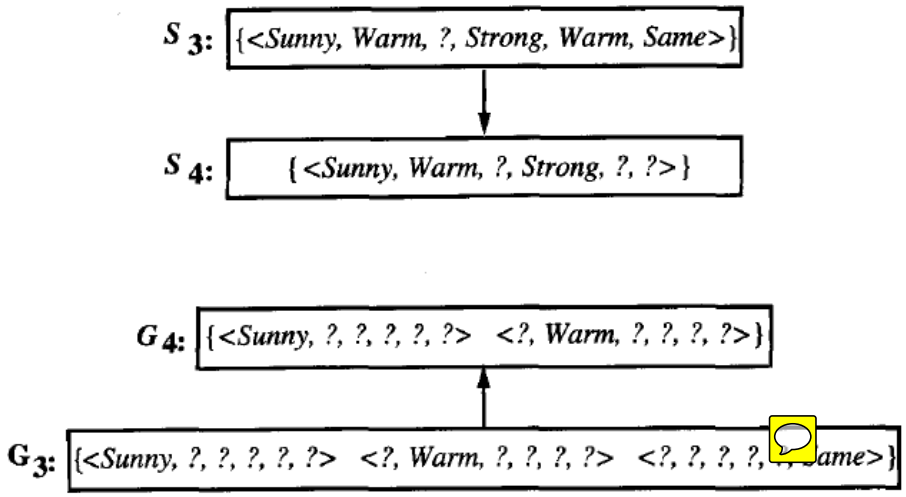
قبلی سازگار هستند یا نه. هر فرضیه ای که کلی‌تر از S باشد، طبق تعریف، با هر نمونه مثبت قبلی‌ای سازگار است. در نقطه‌ی مقابل نیز G خلاصه ای از نمونه های منفی را ذخیره می‌کند. و هر فرضیه ای که از G خاص‌تر باشد با نمونه های منفی قبلی سازگار نیست. چون طبق تعریف چنین فرضیه ای نمونه های منفی G را به عنوان نمونه منفی در خود دارد.



شکل ۲.۵ عملیات الگوریتم Candidate-Elimination در نمونه سوم.

نمونه سوم یک نمونه منفی است و باعث می‌شود که G_2 به G_3 تغییر کند. توجه داشته باشید که کلی‌ترین خاص سازی‌ها در G_3 قرار داده شده.

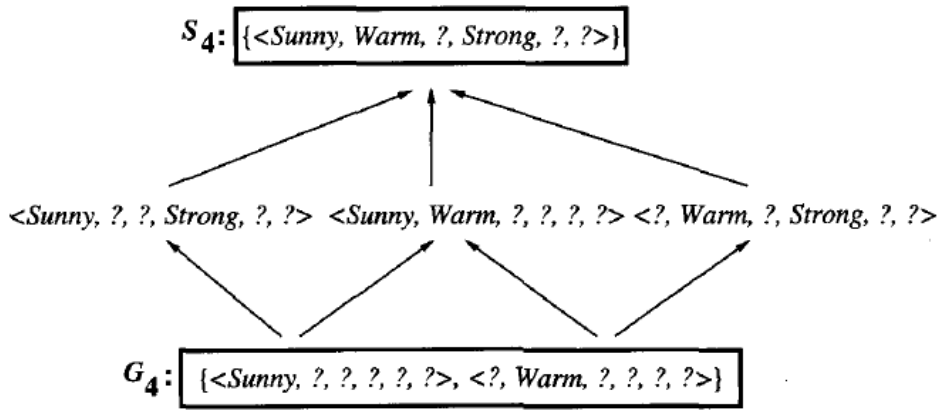
در نمونه آموزشی پنجم (شکل ۲.۶) باز S کلی‌تر می‌شود و در طرف دیگر نیز یکی از اعضای G بخاطر نپوشاندن این نمونه جدید حذف می‌گردد. این حذف در خط اول الگوریتم جدول ۲.۵ نوشته شده. این سؤال به جا است که چرا باید فرضیه های اضافی حذف شود. جواب در این نکته است که خاص‌تر کردن باعث پوشش نمونه نخواهد شد. کلی‌تر کردن نیز باعث می‌شود که فرضیه با حداقل یکی از نمونه های منفی قبلی نا سازگار شود (طبق تعریف: هر فرضیه کلی‌تر حداقل یکی از نمونه های منفی را داراست). پس، ناچار فرضیه‌ی مذکور از G حذف می‌شود و با این حذف شدن شاخه ای از فضای ویژه کم می‌گردد.



Telegram: @Farabar_BI
www.farabar.net

شکل ۲.۶ قدم سوم در الگوریتم Candidate-Elimination.
نمونه سوم (نمونه مثبت) مرز S_3 را به S_4 تبدیل می‌کند. و همچنین یکی از اعضای G_3 در این فرایند حذف می‌شود، زیرا که کلی‌تر از S_4 نیست.

بعد از انجام مراحل برای هر چهار نمونه، دو مرز S_4 و G_4 فضای ویژه و متعاقباً تمامی فرضیه سازگار با آن‌ها چهار نمونه را شامل می‌شود. کل فضای ویژه در شکل ۲.۷ نشان داده شده است. فضای ویژه‌ی پیدا شده به ترتیب نمونه‌ها وابسته نیست (زیرا که در انتها تمامی فرضیه های سازگار با نمونه‌ها را در بر خواهد گرفت). با بیشتر شدن تعداد نمونه‌ها کم کم دو مرز S و G به سوی یکدیگر می‌روند و فضای ویژه‌ی کوچک‌تری را تشکیل می‌دهند.



شکل ۲.۷ فضای ویژه‌ی مشخص شده برای مفهوم EnjoySport برای نمونه های آموزشی داده شده.

۲.٦ نکاتی چند در مورد فضای ویژه و الگوریتم Candidate-Elimination

۲.٦.۱ آیا الگوریتم Candidate-Elimination به سمت فرضیه‌ی درست می‌رود؟

اگر (۱) خطایی در فرضیه‌ها نباشد و (۲) فرضیه ای که درست مفهوم هدف را توصیف کند در H باشد، فضای ویژه‌ی خروجی الگوریتم Candidate-Elimination به سمت فرضیه ای میل می‌کند که مفهوم هدف را به درستی توصیف می‌کند. در واقع، می‌توان بعد از هر مثال بررسی کرد که آیا تعداد نمونه های آموزشی کافی بوده (فضای ویژه به فرضیه‌ی خاصی میل کرده؟ و ابهامات را در مورد مفهوم هدف از بین برده؟). زمانی که دو مرز S و G به یک مجموعه‌ی واحد و یکی برسند فرضیه هدف به طور کامل یاد گرفته شده است.

اما اگر نمونه های آموزشی خطا داشته باشد چه اتفاقی می‌افتد؟ برای مثال فرض کنیم که نمونه دوم در EnjoySport به جای نمونه مثبت به عنوان نمونه منفی ارائه شده بود. متأسفانه در چنین مثال‌هایی الگوریتم، مفهوم هدف را از فضای ویژه حذف می‌کرد! زیرا که به محض مواجهه با نمونه دوم تمام فرضیه‌هایی که آن‌را شامل می‌شد را حذف می‌کرد! البته در چنین شرایطی با دادن نمونه های کافی معلوم می‌شد که S وG به جایی میل می‌کنند که فضای ویژه تهی می‌شود. و این به این معناست که هیچ فرضیه ای در H وجود ندارد که با تمامی نمونه های آموزشی مطابقت داشته باشد. یکی از حالات ممکن این است که نمونه درست باشند اما مفهوم هدف در H وجود نداشته باشد (مثلاً مواقعی که تابع هدف یک تابع فصلی است و ما H را به صورت توابع عطفی در نظر گرفته‌ایم). چنین احتمالاتی را بعداً مفصلاً بررسی خواهیم کرد. اما در حال حاضر بنا بر این فرض است که تمامی مثال‌ها درستند و مفهوم هدف نیز در H وجود دارد.

www.farabar.net

۲.۶.۲ یادگیر چه مثال‌هایی را باید در خواست کند؟

تا الآن فرض می‌کردیم که مثال‌ها از معلم به یادگیر داده می‌شود و یادگیر هیچ حق انتخابی ندارد. حال فرض کنیم که یادگیر حق انتخاب داشته باشد. یعنی بتواند ویژگی‌هایی را در نظر بگیرد و از طریقی (آزمایش، طبیعت و یا معلم) مقدار تابع هدف را برای آن ویژگی‌ها پیدا کند. در چنین شرایطی وضع یادگیر بسیار متفاوت‌تر خواهد بود و یادگیر می‌تواند به انجام آزمایش بپردازد (مثلاً برای مفهوم "استحکام پل" می‌تواند پل‌های جدیدی با خواص دلخواه خود بسازد و از روی طبیعت بفهمد که پل مستحکم هست یا نه؟) یا زمانی که معلمی حاضر است می‌تواند از معلم سؤال کند (پلی طراحی کند و از معلم سؤال کند که آیا پل مستحکم است یا نه؟). به چنین نمونه‌هایی که ویژگی‌هایشان را یادگیر تعیین می‌کند آزمایش[1] می‌گوییم.

دوباره فضای ویژه‌ی بدست آمده برای مثال EnjoySport را در نظر بگیرید (شکل ۲.۳). حال فرض کنیم می‌توانیم آزمایش کنیم. چه آزمایشی، آزمایش خوبی محسوب می‌شود؟ در کل با چه استراتژی کلی‌ای باید آزمایش کنیم؟ واضح است که یادگیر باید آزمایش‌هایی را امتحان کند که تفاوت بین فرضیه‌ها را نشان بدهند. چنین آزمایش‌هایی، آزمایش‌هایی هستند که در بعضی فرضیه های در فضای ویژه، نمونه مثبت و در بعضی دیگر نمونه منفی باشند. برای مثال:

<Sunny,Warm,Normal,Light,Warm,Same>

توجه می‌کنید که این نمونه سه فرضیه از ۶ فرضیه‌ی موجود (شکل ۲.۳) را راضی می‌کند. حال اگر این نمونه با مفهوم هدف سازگار بود S را کلی‌تر می‌کنیم. در مقابل اگر این مثال با مفهوم هدف ناسازگار بود مرز G را خاص‌تر می‌کنیم. در هر صورت اطلاعات مفیدی در مورد مفهوم هدف بدست می‌آید و تعداد اعضای فضای ویژه نیز نصف می‌شود.

در حالت کلی آزمایش بهینه، آزمایشی است که با نصف اعضای فضای ویژه سازگار و با نصف دیگر ناسازگار باشد. اگر آزمایش‌ها چنین شرایطی را داشته باشند در هر آزمایش تعداد اعضای فضای ویژه نصف می‌شود تا در آخر در $\lceil \log_2 |VS| \rceil$ آزمایش مفهوم هدف معلوم می‌شود. درست مثل بازی ۲۰ سؤالی[2] که هر دفعه یک دسته فرضیه رد می‌شوند. در بازی ۲۰ سؤالی نیز بهترین استراتژی بازی پرسیدن سؤال‌هایی است که دقیقاً نصف فرضیه‌ها را در بر بگیرد است. از طرفی دیگر، همان طور که در شکل ۲.۳ نیز نشان داده شده می‌توان آزمایش‌هایی را ایجاد کرد که دقیقاً با نصف فضای ویژه سازگار باشد. در کل ممکن است این کار ممکن نباشد، در چنین شرایطی تعداد آزمایش‌ها متعاقباً بیش از $\lceil \log_2 |VS| \rceil$ خواهد شد.

۲.۶.۲ چگونه می‌توان از فضای ویژه برای تشخیص سازگاری نمونه‌ها با مفهوم هدف استفاده کرد؟

فرض کنید که فقط همان چهار مثال در اختیار یادگیر قرار گرفته و یادگیر هیچ گونه حق آزمایشی نیز ندارد، اما از یادگیر انتظار می‌رود که پیش بینی کند کدام مثال‌ها با مفهوم هدف سازگارند. با وجود اینکه فضای ویژه‌ی بدست آمده هنوز مفهوم هدف را دقیقاً نمی‌توان مشخص کرد. اما با این وجود می‌توان با استفاده از آن قطعاً گفت که مفهوم هدف بعضی نمونه‌ها را چگونه دسته بندی می‌کند. برای مثال فرض کنید که از یادگیر خواسته می‌شود تا مثال‌های جدول ۲.۶ را دسته بندی کند.

---

[1] query
[2] Twenty questions

توجه می‌کنید که با وجود اینکه نمونه A جزو نمونه های آموزشی نبوده اما هر ۶ فرضیه‌ی موجود در فضای ویژه آن‌را مثبت دسته بندی می‌کنند (شکل ۲.۳). و چون تمامی فرضیه‌ها به اتفاق این نمونه را مثبت دسته بندی می‌کنند، یادگیر با اطمینان می‌تواند پیش بینی کند که این نمونه برای مفهوم هدف مثبت است (چون مفهوم هدف یکی از همین فرضیه‌هاست). پس جدا از اینکه کدام فرضیه مفهوم هدف است، نمونه A برای آن مثبت است. توجه دارید که لازم نیست برای تشخیص این که A برای تمامی فرضیه‌ها مثبت است آن‌را با تک تک فرضیه‌ها تطبیق دهیم. فقط کافی است که نمونه A در تک تک اعضای S مثبت باشد (چرا؟). دلیل اینجاست که همه‌ی اعضای فضای ویژه حداقل از یکی از اعضای S کلی‌تر هستند و طبق تعریف کلی‌تری هر نمونه ای که اعضای S را راضی کند، تمامی فرضیه‌ها را راضی خواهد کرد.

نمونه	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Lihgt	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

جدول ۲.۶ نمونه های جدید

به طور مشابه نمونه B یک نمونه منفی دسته بندی می‌شود زیرا که همه‌ی فرضیه های فضای ویژه آن‌را نمونه منفی دسته بندی می‌کنند. و دوباره به طور مشابه کافی است فقط نمونه منفی اعضای G باشد (چرا؟).

در مورد نمونه C قضیه کاملاً متفاوت است. نیمی از فرضیه های فضای ویژه آن‌را نمونه مثبت و نیمی دیگر آن‌را نمونه منفی دسته بندی می‌کنند، پس تا زمانی که یادگیر با نمونه های آموزشی بیشتری مواجه نشده دسته بندی این مثال ممکن نیست. توجه دارید که نمونه C یک نمونه خوب برای آزمایش است. این اتفاق کاملاً قابل پیش بینی بود زیرا نمونه‌هایی که دسته بندیشان غیرممکن است حاوی اطلاعاتی هستند که ما نداریم (و برای همین نمی‌توانیم دسته بندیشان کنیم).

نمونه D توسط ۲ فرضیه نمونه مثبت و توسط ۴ فرضیه‌ی دیگر نمونه منفی دسته بندی می‌شود. در این حالت بر خلاف مثال‌های A و B نمی‌توان با اطمینان نمونه D را دسته بندی کرد. یک روش پذیرفتن رای اکثریت و دسته بندی D به عنوان نمونه منفی با درصد اطمینان (میزان رای اکثریت به رای کل) است. اگر همان طور که در فصل ۶ هم بحث شده احتمال درست بودن هر فرضیه را مساوی در نظر بگیریم چنین دسته بندی‌ای بهترین خروجی خواهد بود. با این حال همچنان احتمال مثبت بودن نمونه وجود دارد.

۲.۷ بایاس استقرایی

همان طور که قبلاً نیز بحث شد الگوریتم Candidate-Elimination در صورت وجود تابع هدف در H و درست بودن نمونه‌ها به تابع هدف میل می‌کند. اما اگر تابع هدف در H نبود چه؟ آیا می‌توان با در نظر گرفتن H به صورت تمامی فرضیه‌ها از این مشکل پرهیز کرد؟ تأثیر اندازه‌ی فضای فرضیه‌ها بر توانایی الگوریتم برای دسته بندی نمونه های جدید چگونه است؟ تأثیر اندازه‌ی فضای فرضیه‌ها بر تعداد نمونه های آموزشی لازم چگونه است؟ این سؤال‌های اساسی در مورد استقرا در کل مطرح هستند. در اینجا این سؤال‌ها را در متن Candidate-

Elimination بررسی خواهیم کرد. اما با این حال، می‌بینید که نتایج بدست آمده از این بررسی برای تمامی سیستم‌های یادگیری مفهوم قابل اجراست.

۲.۷.۱ فضای فرضیه ای بایاس دار

فرض کنید که می‌خواهیم فضای فرضیه‌ها، مفهوم هدف که نا معلوم است را در بر بگیرد. اولین و ساده‌ترین راه این است که تمامی فرضیه های ممکن را در فضای فرضیه ای قرار دهیم. دوباره مثال EnjoySport را در نظر بگیرید. ما در آنجا فضای فرضیه ای را تمام فرضیه های عطفی ممکن فرض کردیم. بخاطر این محدودیت، فضای فرضیه ای فرضیه‌ی بسیار ساده‌ی غیر عطفی "Sky=Sunny or Sky=Cloudy" را در بر نمی‌گیرد. در واقع در فضای فرضیه ای فعلی با دادن سه نمونه زیر به عنوان نمونه های آموزشی، خروجی الگوریتم بـرای فضـای ویـژه تهـی می‌شود.

مثال	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
۱	Sunny	Warm	Normal	Strong	Cool	Change	Yes
۲	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
۳	Rainy	Warm	Normal	Strong	Cool	Change	No

برای این که ثابت شود که فرضیه ای در H وجود ندارد که با سه نمونه بالا سازگار باشد، به این توجه کنید که خاص‌ترین فرضیه سازگار با دو و فرضیه‌ی اول به شکل زیر است:

$$S_2:<?,Warm,Normal,Strong,Cool,Change>$$

با این که این فرضیه خاص‌ترین فرضیه ممکن در H، اما با این حال بیش از حد کلی است زیرا که نمونه سوم را مثبـت دسـته بنـدی مـی‌کنـد. مشکل در این جاست که ما H را با بایاس[1] (محدود) کرده‌ایم و فقط ترکیب‌های عطفی را در نظر گرفته‌ایم. برای حل این مشکل لازم است که از فضای فرضیه ای شامل‌تری استفاده کنیم.

۲.۷.۲ یادگیر بدون بایاس

راه حل این مشکل که ممکن است مفهوم هدف در H نباشد این است که H را مجموعه‌ی تمامی فرضیه های قابل یادگیری در نظر بگیـریم. و این به معنای پیدا کردن تمام زیر مجموعه های مجموعه‌ی X است. در کل، به مجموعه ای که تمامی ایـن زیـر مجموعـه‌هـا را در بـر بگیـرد مجموعه‌ی توانی[2] X می‌گوییم.

در مثال EnjoySport تمامی حالت‌های یک روز که توسط شش ویژگی آن مشخص می‌شد ۹۶ حالت است. چند مفهوم روی ایـن مجموعـه می‌توان تعریف کرد؟ به عبارت دیگر تعداد اعضای مجموعه‌ی توانی X چقدر است؟ در کل تعداد زیر مجموعه های مجموعه‌ی X که |X| عضو

[1] bias
[2] Power set

دارد $2^{|X|}$ است (اندازه‌ی مجموعه‌ی توانی X). 2^{96} مفهوم یا حدود 10^{28} مفهوم می‌توان روی X تعریف کرد. با توجه به آنچه در قسمت ۲.۳ گفته شد فضای فرضیه ای عطفی فقط ۹۷۳ عضو دارد، که واقعاً فضای بایاس داری بوده!

بیایید دوباره عمل یادگیری مفهوم EnjoySport را این دفعه با تعریف فضای فرضیه ای بدون بایاس H′ (که H′ همان مجموعه‌ی توانی X است) انجام دهیم. یک راه برای تعریف چنین H′ ی اضافه کردن نقیض، عطف و فصل به تعداد دلخواه فرضیه‌هایی که قبلی به H است. برای مثال مفهوم هدف "Sky=Sunny or Sky=Cloudy" را می‌توان به صورت زیر نشان داد:

<Sunny,?,?,?,?,?,?>v<Cloudy,?,?,?,?,?,?>

با اضافه کردن چنین فرضیه‌هایی، می‌توانیم با اطمینان به سراغ الگوریتم Candidate-Elimination برویم و مطمئن باشیم که فضای فرضیه‌ها مفهوم هدف را در بر می‌گیرد. با این وجود حل کردن مشکل بدین شکل خود یک مشکل دیگر به وجود می‌آورد، با این فضای فرضیه ای پیش بینی نمونه های جدید غیرممکن می‌شود! برای معلوم شدن دلیل این مشکل فرض کنید که ما سه نمونه مثبت (x_1, x_2, x_3) و دو نمونه منفی (x_4, x_5) را به یادگیر می‌دهیم. بعد از بررسی هر پنج مثال مرز S فصل بین سه نمونه مثبت خواهد بود:

S:{ $(x_1 \vee x_2 \vee x_3)$}

زیرا که این خاص‌ترین فرضیه‌ی ممکن سازگار با سه مثال است. به طور مشابه نیز G به صورت زیر خواهد بود (کلی‌ترین فرضیه‌ی سازگار با نمونه های منفی)

G:{ $\neg(x_4 \vee x_5)$}

مشکل اینجاست که با این فضای فرضیه ای شامل، مرز S همواره فصل نمونه های مثبت و مرز G همواره نقیض فصل نمونه های منفی خواهد بود. پس تنها مثال‌هایی که توسط S و G حذف می‌شوند خود نمونه های آموزشی خواهند بود و برای میل کردن به یک فرضیه باید روی تک تک اعضای X آزمایش انجام دهیم.

در ابتدا ممکن است به نظر برسد که می‌توان با استفاده از آنچه در قسمت ۲.۶.۳ گفته شد با رای گیری روی کلیه‌ی فرضیه های فضای ویژه حداقل به درصدی قابل قبول رسید، اما متأسفانه فقط برای نمونه های آموزشی نتیجه قطعی خواهد بود و برای تمامی مثال‌های غیر آموزشی این درصد ۵۰-۵۰ خواهد بود (نیمی از فرضیه‌ها نمونه را مثبت و نیمی دیگر نمونه را منفی دسته بندی می‌کنند(چرا؟)). برای معلوم شدن دلیل این امر توجه داشته باشید که زمانی که H مجموعه‌ی توانی X است و x نیز یک نمونه غیر آموزشی، در مقابل هر فرضیه ای در H که x را می‌پوشاند فرضیه ای دیگر مثل h′ در مجموعه توانی وجود دارد که در روی تمامی اعضای X مشابه h است و تنها x را نمی‌پوشاند. و البته اگر h در فضای ویژه باشد مسلماً h′ نیز در فضای ویژه خواهد بود، زیرا که در تمامی نمونه های آموزشی مشابه h رفتار می‌کند.

۲.۷.۳ بیهودگی یادگیری بدون بایاس

بحث بالا یک خاصیت بنیادی یادگیری استقرایی را مشخص کرد: یادگیری که هیچ پیش قضاوتی در مورد ماهیت مفهوم هدف نمی‌کند نمی‌تواند نمونه های جدید را دسته بندی کند. در واقع تنها دلیلی که باعث می‌شود الگوریتم Candidate-Elimination در مثال EnjoySport بتواند نمونه های جدید را دسته بندی کند این بود که در تعیین H بایاس شده است و فرض شده تابع هدف به صورت عطف ویژگی‌های موجود بیان شود. در واقع زمانی که این فرض درست است (و نمونه‌ها نیز خطا ندارند) این الگوریتم می‌تواند نمونه های جدید را

درست نیز دسته بندی کند. و زمانی که این فرض غلط باشد، الگوریتم Candidate-Elimination حداقل برای تعدادی از اعضای X دسته بندی اشتباه انجام می‌دهد.

چون یادگیری استقرایی نیاز به حداقل نوعی پیش فرض، بایاس در استقرا (بایاس در استقرا نباید با بایاس تخمینی که در فصل ۵ آمده اشتباه گرفته شود) دارد، دسته بندی ی روش‌های مختلف بایاس مفید خواهد بود. نکته‌ی مهم اینجا این است که چگونه یادگیر نمونه های آموزشی را برای دسته بندی نمونه های دیگر تعمیم می‌دهد. اگر فرض کنیم که به الگوریتم یادگیری L، نمونه های آموزشی دلخواه $D_c = \{< x, c(x) >\}$ برای یادگیری مفهوم دلخواه c داده شود و بعد از آموزش از L خواسته می‌شود تا نمونه جدید x_i را دسته بندی کند و $L(x_i, D_c)$ تشخیص الگوریتم L درباره‌ی نمونه جدید باشد (مثبت یا منفی) می‌توان این استنباط L را توسط عبارت زیر توصیف کنیم:

$$(D_c \wedge x_i) \succ L(x_i, D_c)$$

عبارت $y \succ z$ بدین معناست که z از روی y استنباط[1] شده است. برای مثال اگر L همان الگوریتم Candidate-Elimination باشد و D_c نیز همان نمونه های جدول ۲.۱ باشد و x_i نیز همان نمونه اول جدول ۲.۶ باشد پس خواهیم داشت که $L(x_i, D_c)$ =(EnjoySport=Yes).

چون L یک الگوریتم یادگیری استقرایی است، نمی‌توان ثابت کرد که حکم $L(x_i, D_c)$ درست یا غلط است. این فقط استنباطی است که L از نمونه های آموزشی D_c در مورد نمونه جدید x_i می‌کند. با این حال این سؤال جالب است که بپرسیم که چه پیش فرض‌های دیگری را می‌توان به $D_c \wedge x_i$ اضافه کرد تا همچنان $L(x_i, D_c)$ درست بماند. این مجموعه پیش فرض‌های جدید را بایاس استقرایی ی L[2] می‌نامیم. به طور دقیق‌تر بایاس استقرایی L را به صورت دسته پیش فرض‌های B تعریف می‌کنند که برای تمامی نمونه های x_i داشته باشیم:

$$(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)$$

عبارت $y \vdash z$ به این معناست که z قابل نتیجه گیری از y است[3] (مثلاً:z از y اثبات می‌شود). پس بایاس استقرایی را به صورت دسته پیش فرض‌های کافی B تعریف می‌کنیم تا بتوان استقرا را با استنتاج (نتیجه گیری) توجیه کرد. به طور خلاصه،

تعریف: فرض کنید که الگوریتم یادگیری مفهوم L برای فضای نمونه های X تعریف شده است و c نیز یک مفهوم دلخواه روی X و $D_c = \{< x, c(x) >\}$ نیز نمونه های آموزشی دلخواهی از c باشند. اگر $L(x_i, D_c)$ تشخیص استقرایی L از x_i بر اساس D_c باشد. بایاس استقرایی L هر دسته فرضیاتی مثل B است که برای هر مفهوم هدف c و نمونه های آموزشی مربوطه D_c داشته باشیم:

$$(\forall x_i \epsilon X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \qquad (2.1)$$

پس با این تعریف، بایاس استقرایی الگوریتم Candidate-Elimination چیست؟ برای جواب این سؤال، باید اول $L(x_i, D_c)$ را دقیقاً برای این الگوریتم مشخص کنیم: برای هر دسته نمونه آموزشی D_c الگوریتم Candidate-Elimination اول فضای ویژه‌ی متناسب با آن $VS_{H.D_c}$ را محاسبه می‌کند و سپس x_i را از رای گیری روی آن دسته بندی می‌کند. بیاید فرض کنیم که الگوریتم فقط زمانی که کل

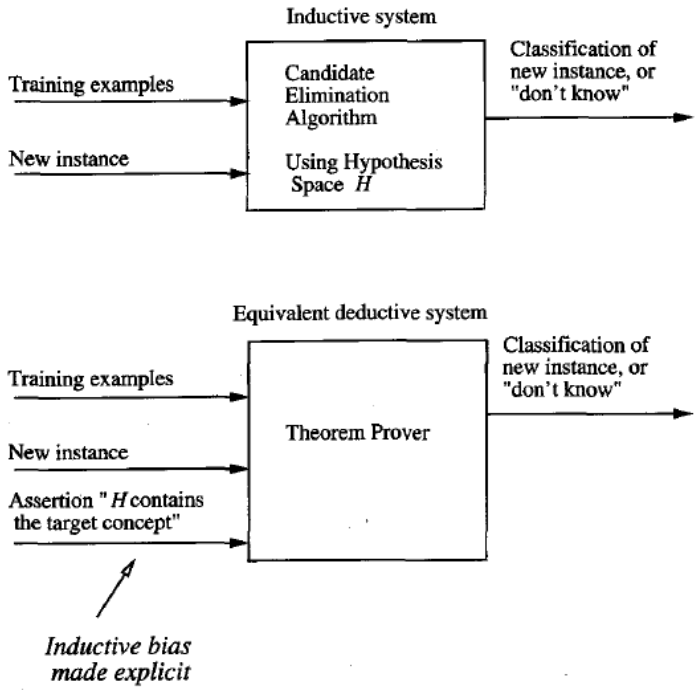
[1] infer
[2] inductive bias
[3] Follows deductively

فرضیه های فضای ویژه به طور صد در صدی رای دهند خروجی بدهد. حال با معلوم شدن تعریف $L(x_i, D_c)$ برای الگوریتم Candidate-Elimination، بایاس استقرایی آن چیست؟ جواب واضح است، طبق تعریف پیش فرض $c \in H$ است. با داشتن این پیش فرض هر استنباط استقرایی توسط Candidate-Elimination درست خواهد بود.

برای روشن شدن دلیل اینکه چرا از D_c و مشخصات x_i و اینکه $B = \{c \in H\}$ دسته بندی $L(x_i, D_c)$ نتیجه گیری می‌شود، به بحث زیر توجه کنید. اول اینکه، توجه داشته باشید که پیش فرض $c \in H$ نتیجه می‌دهد که $c \in VS_{H.D_c}$ (زیرا که $c \in H$ و از تعریف فضای ویژه به عنوان مجموعه‌ی شامل تمام اعضای H که با نمونه های D_c مطابقت دارد و تعریف $D_c = \{< x, c(x) >\}$ به عنوان نمونه های آموزشی‌ای که با c سازگار است). دوم اینکه، توجه داشته باشید که فرض کردیم $L(x_i, D_c)$ رای صد درصدی اعضای فضای ویژه است. بنا برین، اگر L، $L(x_i, D_c)$ را به عنوان خروجی بدهد، حتما تمامی فرضیه های $VS_{H.D_c}$ نیز همان تشخیص را داشته اند و چون $c \in VS_{H.D_c}$. بنابرین $c(x_i) = L(x_i, D_c)$. به طور خلاصه الگوریتم Candidate-Elimination تعریف شده با بایاس زیر مطابقت دارد:

بایاس استقرایی الگوریتم Candidate-Elimination. باید فضای فرضیه‌ها H مفهوم هدف c را شامل شود.

شکل ۲.۸ خلاصه‌ی وضع را به صورت نموداری نمایش می‌دهد. الگوریتم استقرایی Candidate-Elimination در بالای شکل دو ورودی دارد: نمونه های آموزشی و نمونه جدید. در پایین شکل ثابت کننده‌ی قضیه با همان دو ورودی و یک فرض سوم، "مفهوم هدف در H موجود است"، نشان داده شده است. به طور کلی، این دو سیستم برای هر دسته نمونه آموزشی و هر نمونه جدید خروجی یکسان می‌دهند. البته بایاس استقرایی که ورودی سوم ثابت کننده‌ی قضیه است به طور بالقوه در کد الگوریتم Candidate-Elimination وجود دارد. به عبارتی، این فقط در شکل نشان داده شده و تأثیری دیگر ندارد. از طرف دیگر این ورودی به ما اطمینان می‌دهد که خروجی حتماً درست است.



شکل ۲.۸ مدل سازی سیستم‌های استقرایی با سیستم‌های استقرایی هم ارز.
رفتار ورودی–خروجی الگوریتم Candidate-Elimination با استفاده از فضای فرضیه ای H درست مشابه رفتار استقرایی ثابت کننده‌ی قضیه ای است که از پیش فرض "مفهوم هدف در فضای فرضیه ای وجود دارد" استفاده می‌کند. از این رو پیش فرض اضافه شده، "مفهوم هدف در فضای فرضیه ای وجود دارد"، بایاس استقرایی Candidate-Elimination نامیده می‌شود. دسته بندی سیستم‌های استقرایی با بایاس‌های استقرایی‌شان با ما این اجازه را می‌دهد که آن‌ها را با سیستم‌های استقرایی نظیر مد سازی کنیم. این مدل سازی باعث می‌شود تا بتوانیم سیستم‌های استقرایی را بر اساس نحوه‌ی استقرا روی نمونه های آموزشی مقایسه کنیم.

یکی از مزیت‌های نگاه بایاس استقرایی به سیستم‌های استنباط استقرایی، پیدا کردن نحوه‌ی تامیم نمونه های آموزشی برای نمونه های جدید، بدون این که با مراحل الگوریتمشان درگیر شویم است. مزیت دوم اینکه، می‌توانیم قدرت یادگیر های الگوریتم‌های مختلف را مقایسه کنیم. برای مثال سه الگوریتم زیر به ترتیب ضعیف‌ترین به قوی‌ترین بایاس ترتیب شده‌اند:

۱. Rote-Learner: هر نمونه آموزشی را در حافظه ذخیره سازی می‌کند. برای دسته بندی نمونه های جدید، فقط در حافظه جستجو می‌کنند، اگر مثال میان نمونه های آموزشی بود جواب را خروجی می‌دهد، در غیر این صورت خروجی نمی‌دهد!

۲. الگوریتم Candidate-Elimination: همان طور که قبلاً نیز توضیح داده شد، این الگوریتم خاص‌ترین و کلی‌ترین فرضیه های سازگار با نمونه های آموزشی را پیدا می‌کند و با استفاده از فضای ویژه (رای صد درصدی) نمونه های جدید را دسته بندی می‌کند.

۳. FIND-S: این الگوریتم خاص‌ترین فرضیه‌ی ممکن سازگار با نمونه های آموزشی را پیدا کرده و از آن برای دسته بندی نمونه های جدید استفاده می‌کند.

الگوریتم Rote-Learner هیچ بایاس استقرایی‌ای ندارد و نمونه های جدید را فقط با توجه به نمونه های آموزشی دسته بندی می‌کند و هیچ فرض اضافه ای نمی‌کند. Candidate-Elimination بایاس استقرایی قوی‌تری دارد: مفهوم هدف در میان اعضای فضای فرضیه‌ها H موجود است. چون Candidate-Elimination بایاس استقرایی قوی‌تری نسبت به Rote-learner دارد مثال‌هایی را دسته بندی می‌کند که Rote-Learner از دسته بندی آن‌ها عاجز است. البته توجه دارید که درستی این دسته بندی‌ها کاملاً به درستی بایاس استقرایی وابسته است. بایاس استقرایی FIND-S حتی از این هم قوی‌تر است: علاوه بر این که فرض می‌کند که مفهوم هدف یکی از اعضای فضای فرضیه‌هاست، فرض می‌کند که تمامی نمونه های جدید نمونه منفی هستند مگر اینکه خلافش ثابت شود (با اطلاعات قبلی ثابت شده باشد که نمونه مثبت است).

بد نیست که در مواجهه با دیگر متد های استنباط استقرایی بایاس استقرایی و قدرت آن‌را نیز بررسی کنیم. متدهایی که بایاس استقرایی قوی‌تری دارند قدرت بیشتری نیز در دسته بندی نمونه های جدید دارند و نمونه های جدید بیشتری را می‌توانند دسته بندی کنند. بعضی بایاس‌های استقرایی پیش فرض‌های دسته بندی کننده ای هستند که دسته ای از فرضیه‌ها را به کلی کنار می‌گذارند، مثل این بایاس: "فضای فرضیه‌ها H باید شامل مفهوم هدف باشد". بعضی دیگر از بایاس‌ها فقط ترتیبی برای اولویت بین فرضیه‌ها می‌گذارند، مثل: "فرضیه های خاص‌تر بر فرضیه‌های کلی‌تر ارجحیت دارند". بعضی از بایاس‌ها نیز به طور بالقوه در یادگیر قرار داده شده‌اند، بایاس‌هایی که ذکر شد. در فصول ۱۱ و ۱۲ سیستم‌های دیگری را مورد بحث قرار خواهیم داد که بنا به خواست یادگیر بایاس تغییر می‌کند.

۲.۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل:

- می‌توان به مسایل یادگیری مفهوم به دید جستجوی در میان تمامی فرضیه های ممکن نگاه کرد.
- ترتیب کلی‌تری فرضیه‌ها می‌تواند به این جستجو سازمان ببخشد، این سازماندهی در جستجوی فضای فرضیه‌ها را می‌توان برای هر مسئله‌ی یادگیری مفهوم به کار برد.
- الگوریتم FIND-S با استفاده از ترتیب کلی‌تری، یک جستجو در میان فرضیه‌ها را در یک شاخه از ترتیب کلی‌تری، ترتیب می‌دهد، تا در آخر خاص‌ترین فرضیه‌ی مطابق با نمونه های آموزشی را پیدا کند.
- الگوریتم Candidate-Elimination با استفاده از ترتیب کلی‌تری فضای ویژه را (تمامی فرضیه های سازگار با نمونه های آموزشی) با استفاده از محاسبه‌ی خاص‌ترین (S) و کلی‌ترین (G) فرضیه های سازگار پیدا می‌کند.
- چون S وG تمامی فرضیه های سازگار با نمونه‌ها را محدود می‌کنند، به یادگیر اطلاعاتی قطعی در مورد مفهوم هدف می‌دهند. این فضای ویژه دو کاربرد دارد: اول اینکه می‌توان بررسی کرد که آیا فرضیه های ممکن همگرا شده‌اند، تا مشخص شود که آیا نمونه های آموزشی کافی بوده و چه آزمایش‌هایی برای همگرا سازی فضای ویژه مناسب است. دوم اینکه می‌توان با استفاده از آن برای دسته بندی نمونه های جدید استفاده کرد.
- با وجود اینکه فضای ویژه و الگوریتم Candidate-Elimination محیطی ادراکی برای درک یادگیری مفهوم ایجاد می‌کنند، دو مشکل عمده دارند: اول اینکه در مقابل خطا و داده های نویز دار کاملاً آسیب پذیرند و دوم اینکه اگر مفهوم هدف در H نباشد نمی‌توانند آن‌را پیدا کنند. در فصل ۱۰ با الگوریتم‌هایی که از ترتیب کلی‌تری استفاده می‌کنند و در مقابل نویز نیز مقاومند می‌پردازیم.
- الگوریتم‌های استقرایی فقط زمانی می‌توانند نمونه های جدید را دسته بندی کنند که بایاسی داشته باشند. بایاس استقرایی باعث می‌شود که این الگوریتم‌ها فرضیه ای را از فرضیه‌ی دیگر مقدم‌تر بدانند و با آن نمونه های جدید را دسته بندی کنند. بایاس استقرایی در نظر گرفته شده در الگوریتم Candidate-Elimination این است که فرض شده مفهوم هدف در فضای فرضیه‌ها موجود است $(c \in H)$. استنباط‌هایی که این الگوریتم انجام می‌دهد مبتنی بر نمونه های آموزشی و این بایاس استقرایی است.
- اگر فضای فرضیه‌ها همه‌ی فرضیه های ممکن را داشته باشد (مجموعه‌ی توانی مثال‌ها) بایاس استقرایی Candidate-Elimination بر طرف می‌شود. اما متأسفانه هرگونه دسته بندی نمونه های جدید با این عمل از بین می‌رود. یادگیر بدون بایاس نمی تواند با استقرا نمونه های جدید را دسته بندی کند.

ایده‌ی یادگیری مفهوم و ترتیب کلی‌تری خیلی جدید نیست. (Burner et al. 1957) اولین مطالعات را در مورد یادگیری مفهوم روی انسان‌ها انجام داد، (Hunt and Hovland 1963) (1963) اولین تلاش‌ها را برای الگوریتمیک کردن آن انجام دادند. تز دکترای معروف (Wiston) که یادگیری مفهوم را به صورت جستجویی با استفاده از عملیات‌های کلی سازی و جزیی سازی تعریف کرد. (Plotkin) (1970,1971) فرمولی اولیه از رابطه‌ی کلی‌تری وابسته به جانشینی θ [1] (که در فصل ۱۰ مطرح شده) بدست آورد. (Simon and Lea 1973) یادگیری مفهوم را به معنای جستجو در فضای فرضیه ای را مطرح کردند. بقیه‌ی سیستم‌های یادگیری مفهوم اولیه، (Popplestone 1969) ، (Michalski 1973)، (Buchanan 1974)، (Vere 1975) و (Hayes-Roth 1974) هستند. تعداد زیادی از الگوریتم‌هایی که تا به حال برای یادگیری مفهوم طراحی شده مبتنی بر نمایش نمادین[2] بوده است. در فصل ۱۰، تعداد بسیار دیگری از الگوریتم‌های یادگیری مفهوم از جمله الگوریتم‌هایی که از منطق مرتبه اول استفاده می‌کنند، الگوریتم‌هایی که نسبت به نویز و خطا مقاوم هستند و الگوریتم‌هایی که حتی اگر مفهوم هدف در میان فضای فرضیه‌ها نباشد باز هم درست کار می‌کنند را بررسی خواهیم کرد.

---

[1] θ-sumsumption
[2] symbolic representation

فضا های ویژه و الگوریتم Candidate-Elimination توسط میشل (Mitchell 1977,1982) طراحی و معرفی شد. کاربرد این الگوریتم برای استنباط قوانین طیف بینی جرمی نیز توسط وی انجام شد (1979). و همچنین کاربرد آن برای یادگیری قوانین کنترل جستجو نیز توسط وی در سال ۱۹۸۳ انجام شد. Haussler (1988) نشان داد که مرز کلی متناسب با تعداد نمونه های آموزشی می‌تواند، حتی زمانی که فضای فرضیه ای شامل عطف ویژگی‌های نمونه‌هاست، به صورت نمایی افزایش یابد. (Smith and Rosenbloom 1990) نشان دادند که تغییر کوچکی در نمایش مجموعه‌ی G می‌تواند در بعضی موارد پیچیدگی را بهبود بخشد و (Hirsh 1992) نشان داد که یادگیری می‌تواند در بعضی موارد که G مرتب[1] متناسب با چند جمله ای از تعداد نمونه‌ها باشد. (Subramanian and Feigenbaum 1986) متدی را که می‌تواند نمونه های موثری در بعضی موارد با فاکتور گیری از فضای ویژه[2] انجام دهد. یکی از بزرگ‌ترین محدودیت‌های عملی الگوریتم Candidate-Elimination نیاز آن به داده های آموزشی بدون خطاست. (Mitchell 1979) تعمیمی از این الگوریتم را ارائه می‌کند که می‌تواند با خطای دسته بندی محدود و از پیش تعیین شده کار کند و (Hirsh 1990,1994) تعمیم زیبایی برای کار با خطای محدود در ویژگی‌های حقیقی مقدار که نمونه های آموزشی را توصیف می‌کنند را ارائه می‌کند. (Hirsh 1990) الگوریتم Incremental Version Space Merging را که تعمیمی از الگوریتم Candidate-Elimination است را برای حالاتی که اطلاعات آموزشی حالات مختلفی از قیود توسط فضای ویژه بیان شده است را ارائه داد. اطلاعات هر یک از قیود توسط فضای ویژه ای بیان شده و سپس قیود با تقسیم فضای ویژه ترکیب می‌شوند. (Sebag 1994, 1996) روش یادگیری‌ای را که فضای ویژه‌ی فصلی می‌نامد را برای یادگیری فصل مفاهیم از داده های خطا دار ارائه می‌کند. در این روش، فضای ویژه‌ی مجزایی برای هر نمونه‌ی آموزشی یاد گرفته شده و نمونه های جدید با رای گیری میان این فضاهای ویژه دسته بندی می‌شوند. وی تحقیقات زیادی را در زمینه های مختلف انجام داده و برتری نتایج الگوریتم خود را نسبت به الگوریتم‌های مشابه مانند درخت تصمیم و k-nearest neighbor نشان می‌دهد.

تمارین

۲.۱ توضیح دهید که چرا اندازه‌ی فضای فرضیه‌ها در مثال EnjoySport ۹۷۳ شد. با اضافه کردن ویژگی دیگری مثل WaterCurrent که سه حالت Light، Moderate و Strong را داشته باشد، تعداد مثال‌ها و فرضیه های ممکن چگونه تغییر می‌کرد؟ در حالت کلی اگر ویژگی A را که k حالت دارد را اضافه کنیم تعداد مثال‌ها و فرضیه های ممکن چگونه تغییر خواهد کرد؟

۲.۲ برای الگوریتم Candidate-Elimination دو مرز S و G را برای جدول ۲.۱ با ترتیب عکس حساب کنید. با وجود اینکه فضای ویژه‌ی بدست آمده در انتها یکی است (چرا؟)، اما در مراحل میانی S و G متفاوتی بدست می‌آید. آیا می‌توانید با عوض کردن ترتیب کاری کنید که مجموع تعداد اعضای S و G در تمام مراحل کمینه شود؟

۲.۳ دوباره مسئله‌ی EnjoySport را با فضای فرضیه ای قسمت ۲.۲ در نظر بگیرید. حال اگر فضای فرضیه ای H′ را تمام ترکیب‌های دو تایی فصلی H در نظر بگیریم الگوریتم Candidate-Elimination را برای این فضای فرضیه ای جدید و نمونه های جدول ۲.۱ انجام دهید.(سری S ها و G های مراحل را بدست آورید).

یک نمونه بسیار ساده از H′:

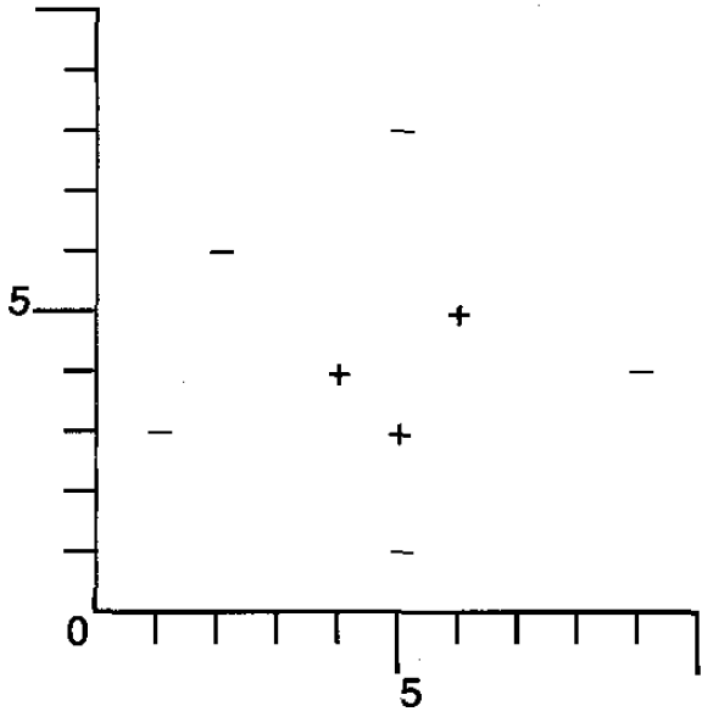
---

[1] sorted
[2] Factoring version space

<?,Cold,High,?,?,?>V<Sunny,?,High,?,?,Same>

۲.۴ اگر فضای نمونه ای نقاطی با x و y صحیح باشند و فضای فرضیه‌ها نیز مستطیل‌ها باشند. به عبارت دقیق‌تر فرضیه‌ها به فرم $a \le x \le b$ و $c \le y \le d$ هستند که a,b,c,d در آن اعداد صحیحند.

(a) فضای ویژه ای که با نمونه های آموزشی شکل زیر مطابقت دارد را در نظر بگیرید. مرز S در این فضای ویـژه چیسـت؟ آن‌را بنویسـید و در شکل نیز مشخص کنید.



(b) G را برای فضای ویژه‌ی مربوطه چیست؟ آن‌را بنویسید و در شکل نیز مشخص کنید.

(c) فرض کنید که حالا یادگیر حق آزمایش دارد. آزمایشی را که اندازه‌ی فضای ویژه را کم می‌کند پیدا کنید. یک آزمایش که اندازه‌ی فضـای ویژه را کم نمی‌کند نیز پیدا کنید.

(d) حال فرض کنید که مفهوم هدف خاصی را برای تعلیم در نظر گرفته‌ایم مثل $3 \le x \le 5$ و $2 \le y \le 9$ کمترین تعداد نمونه آموزشی‌ای که لازم است به یادگیر الگوریتم Candidate-Elimination بدهیم تا این مفهوم را یاد بگیرد چقدر است؟

۲.۵ نمونه های آموزشی مثبت و منفی زیر را که برای آموزش مفهوم "جفت‌هایی که با هم در یـک خانـه زنـدگی مـی‌کننـد" در نظـر بگیریـد. ویژگی‌ها به ترتیب جنسیت، رنگ مو (سیاه، قهوه ای و بور)، قد (بلند، متوسط و کوتاه) و ملیت (امریکایی، فرانسـوی، المـانی، ایرلنـدی، هنـدی، ژاپنی، پرتقالی) هستند.

+ (<مرد،قهوه ای،بلند،امریکایی>،<زن،سیاه،کوتاه،امریکایی>)

+ (<مرد،قهوه ای،کوتاه،فرانسوی>،<زن،سیاه،کوتاه،امریکایی>)

− (<زن،قهوه ای،بلند،المانی>،<زن،سیاه،کوتاه،هندی>)

+ (<مرد،قهوه ای،بلند،ایرلندی>،<زن،قهوه ای،کوتاه،ایرلندی>)

فرض کنید فضای فرضیه ای بر روی این مثال‌ها به صورت زیر تعریف شده که هر یک از ویژگی‌ها می‌تواند ؟ یا ۰ یا یک مقدار باشد (همیشه ۴ ویژگی معلوم است). برای مثال:

(<مرد،؟،بلند،؟>،<زن،؟،؟،ژاپنی>)

که این مثال تمامی زوج‌هایی را در بر می‌گیرد که نفر اول مردی قد بلند (از هر ملیتی با هر رنگ مویی) است و نفر دوم زنی ژاپنی (از هر قدی و هر رنگ مویی) است را در بر می‌گیرد

(a) فرایند الگوریتم Candidate-Elimination را برای مثال‌های فوق طی کنید و مرز های S و G را برای فضای ویژه بعد از هر مثال بیابید.

(b) چند تا از فرضیه های فضای فرضیه ای تعریف شده با مثال زیر سازگارند؟

+ (<مرد،سیاه،کوتاه،پرتقالی>،<زن،بور،بلند،هندی>)

(c) فرض کنید که فقط نمونه مثبت قسمت b را به عنوان نمونه آموزشی داریم و حالا به یادگیر اجازه داده می‌شود که آزمایش انجام دهد. سری‌ای از آزمایش‌ها را ترتیب دهید که در هر صورت ما را به فرضیه‌ی درست برساند (با فرض اینکه مفهوم هدف در فضای فرضیه ای وجود دارد). کوتاه‌ترین سری آزمایش‌ها را انتخاب کنید. طول این سری چه ربطی به فرضیه جواب قسمت b دارد؟

(d) با توجه به این که فضای فرضیه های تعریف شده تمامی مفهوم‌های هدف قابل تعریف روی فضای مثال‌ها را در بر نمی‌گیرد، اگر H را طوری تعریف می‌کردیم که تمامی مفاهیم هدف ممکن را در بر بگیرد، جواب قسمت c چه تغییری می‌کرد؟

۲.۶ اثبات قضیه‌ی ۲.۱ (ارایه فضای ویژه) را کامل کنید

۲.۷ مسئله‌ی یادگیری مفهومی را در نظر بگیرید که در آن هر مثال یک عدد حقیقی است و هر فرضیه نیز بازه ای روی اعداد حقیقی است. به طور دقیق‌تر، فضای فرضیه‌ها H به صورت a<x<b در نظر گرفته می‌شود که در آن a,b اعداد حقیقی‌اند. برای مثال 4.5<x<6.1 تمامی اعداد بین ۴.۵ و ۶.۱ را مثبت و بقیه‌ی اعداد حقیقی را منفی دسته بندی می‌کند. غیر رسمی، توضیح دهید که چرا برای تعدادی نمونه مثبت خاص‌ترین فرضیه موجود نیست. تغییری را در فضای فرضیه‌ها پیشنهاد کنید که این چنین فرضیه‌هایی موجود باشد.

۲.۸ در این فصل فضای فرضیه ای بدون بایاس (مجموعه‌ی توانی مثال‌ها) معرفی شد و گفته شد که با استفاده از آن دقیقاً نصف فرضیه‌ها با هر مثال را مثبت و نصفی دیگر منفی دسته بندی می‌کنند. این گزاره را اثبات کنید. به عبارت دیگر ثابت کنید برای هر فضای مثال‌های X و نمونه های آموزشی D و هر نمونه جدید c ، اگر H مجموعه‌ی توانی X باشد دقیقاً نیمی از فرضیه های $VS_{H,D}$، x را مثبت و نیمی دیگر x را منفی دسته بندی می کنند.

۲.۹ مسئله ای از یادگیری مفهوم را در نظر بگیرید که هر مثال عطفی از n ویژگی منطقی $a_1, \dots, a_n$ باشد. یک مثال ممکن است این باشد:

$$(a_1 = T) \wedge (a_2 = F) \wedge \dots \wedge (a_n = T)$$

حال فرض کنید که فضای فرضیه‌ها به صورت فصلی از ویژگی‌ها تعریف شود. مثال:

$$(a_1 = T) \vee (a_5 = F) \vee (a_7 = T)$$

الگوریتمی پیشنهاد کنید که نمونه های آموزشی را بگیرد و اگر فرضیه‌ی سازگاری با آن‌ها وجود داشت آن‌را خروجی دهد. الگوریتم باید در زمان متناسب با چند جمله ای از n و تعداد نمونه های آموزشی اجرا شود.

۲.۱۰ برنامه ای برای الگوریتم FIND-S بنویسید و آن‌را را برای مثال EnjoySport اجرا کنید و نشان دهید که مراحل همان مراحل قسمت ۲.۴ است. با این برنامه تعیین کنید که چند نمونه تصادفی آموزشی برای تعیین دقیق مفهوم هدف لازم است. برنامه ای بنویسید که نمونه هـای آموزشی تصادفی متناسب با مفهوم زیر ایجاد کند:

<?،?،?،?،Warm،Sunny>

فرض کنید که خروجی برنامه‌ی ایجاد نمونه آموزشی را به برنامه‌ی اول بدهیم، آیا می‌توانید حدس بزنید که به طور متوسط چند نمونه آموزشـی لازم است تا برنامه به مفهوم هدف پی ببرد؟ این کار را حداقل ۲۰ بار انجام دهید و متوسط آن‌را حساب کنید. فکر می‌کنید با عوض کردن تعداد ؟ های مفهوم هدف این تعداد چگونه تغییر می‌کند؟ تعداد ویژگی‌ها چه تأثیری بر این مقدار (تعداد نمونه های آموزشی لازم) دارد؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

سازگار	Consistent
فضای ویژه	space version
فرضیه	Hypothesis
خاص	Specific
کلی	General
نمونه آموزشی	training example
کلی‌ترین خاص سازی‌ها	minimal specialization
خاص‌ترین کلی سازی‌ها	maximal generalization
مفهوم هدف	target concept
دسته بندی	Classify
بایاس	Bias
نقیض	Negation
شامل	Expressive
مثال غیر آموزشی	unobserved instance
بایاس استقرایی	inductive bias

Telegram: @Farabar_BI
www.farabar.net

فصل سوم: یادگیری درخت تصمیم گیری

یادگیری درخت[1] تصمیم گیری یکی از پرکاربردترین و کارامدترین متد های یادگیری استقرایی است. این متد در یادگیری توابع گسسته مقدار با داده های خطا دار به کار می‌رود. در این فصل به خانواده ای از الگوریتم‌های یادگیری درختی، مثل الگوریتم‌های ID3، ASSISTANT و C4.5 می‌پردازیم. این متد های یادگیری درختی فضای فرضیه ای کاملی را جستجو می‌کنند و مشکل محدودیت فضای فرضیه ای را ندارند. بایاس‌های استقرایی این الگوریتم‌ها این است که همیشه درخت‌های کوچک‌تر را بر درخت‌های بزرگ‌تر ترجیح می‌دهند (اصل تیغ Occam).

۳.۱ مقدمه

یادگیری درختی متدی برای تخمین توابع هدف گسسته مقدار است، در یادگیری درختی تابع تخمین زده شده با یک درخت تصمیم گیری مشخص می‌شود. درخت‌های بدست آمده را نیز می‌توان به صورت دسته ای از دستور های if-then نیز نمایش داد تا بررسی آن برای انسان راحت‌تر گردد. این متدها از جمله متداول‌ترین متدها در یادگیری‌های استقرایی هستند و در حوزه‌ی وسیعی از کارهای یادگیری، از یادگیری تشخیص موارد پزشکی گرفته تا تشخیص میزان ریسک وام، مورد استفاده قرار گرفته‌اند.

۳.۲ نمایش درخت تصمیم گیری

درخت تصمیم گیری با ترتیب کردن نمونه‌ها از ریشه به سمت برگ‌های درخت، نمونه‌ها را دسته بندی می‌کند. در این درخت هر گره ویژگی‌ای را در مورد نمونه و هر شاخه (که از آن گره خارج می‌شود) مقادیر مربوطه‌ی آن ویژگی را مشخص می‌کند. برای دسته بندی هر نمونه ابتدا از ریشه شروع می‌کنیم، به هر ویژگی که می‌رسیم از شاخه ای از درخت که ویژگی نمونه با آن مطابق است پایین می‌رویم. این فرایند برای زیر درخت‌ها نیز ادامه می‌یابد تا به دسته بندی نمونه برسیم.

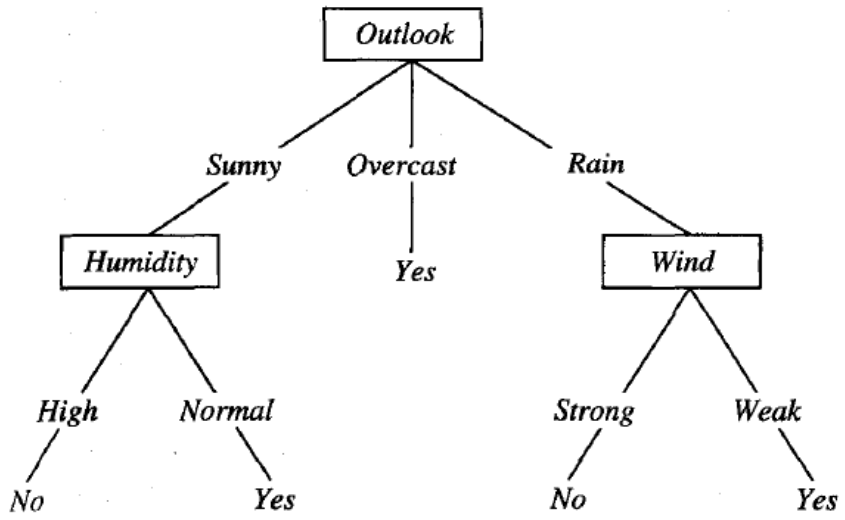
[1] tree

www.farabar.net

شکل ۳.۱ یک مثال از درخت تصمیم گیری را نشان می‌دهد. این درخت تصمیم گیری نشان می‌دهد که مقدار هدف PlayTennis را نشان می‌دهد. برای مثال، نمونه‌ی

<Outlook=Sunny,Temperature=Hot,Humidity=High,Wind=Strong>

در چپ‌ترین گوشه‌ی پایین درخت قرار می‌گیرد، پس بنابراین این نمونه منفی دسته بندی خواهد شد (درخت پیش بینی می‌کند برای این مقادیر PlayTennis مقدار No را داشته باشد). درخت و نمونه‌های آمده در جدول ۲.۳ که برای توضیح الگوریتم یادگیری ID3 مورد استفاده قرار گرفته‌اند از (Quinlar 1986) گرفته شده‌اند.



شکل ۳.۱ درختی تصمیم گیری برای مفهوم PlayTennis.
نمونه‌ها با ترتیب شدن بین شاخه های درخت دسته بندی می‌شوند و در انتها مقدار برگ را بر می‌گردانند (در این مثال مقادیر Yes یا No). این درخت نمونه‌ی مذکور را برای مفهوم PlayTennis منفی دسته بندی خواهد کرد.

در کل درخت‌های تصمیم گیری روابط فصلی‌ای از عطف شروط را برای دسته بندی نمونه‌ها به کار می‌برند. هر مسیر از ریشه‌ی درخت به سمت برگ‌ها عطفی از روابط در مورد ویژگی‌هاست و کل درخت نیز فصلی از این عطف‌هاست. برای مثال، شکل ۳.۱ متناظر با رابطه‌ی زیر است:

$$(Outlook = Sunny \wedge Humidity = Normal)$$

$$\vee (Outlook = Overcast)$$

$$\vee (Outlook = Rain \wedge Wind = Weak)$$

۳.۳ مسائل مناسب برای درخت تصمیم گیری

با وجود اینکه متد های یادگیری درختی زیادی با نیازها و قابلیت‌های متفاوت ارائه شده است، اما اغلب یادگیری درخت تصمیم گیری برای مسائلی با ویژگی‌های زیر مناسب است:

- نمونه‌ها با زوج مرتب دسته ویژگی‌ها و مقدار تابع هدف مشخص شوند. نمونه‌ها در این مسائل با دسته ای از ویژگی‌های ثابت (مثلاً Temperature و ...) و مقادیرشان (مثل Hot) مشخص شوند. راحت‌ترین وضعیت برای یادگیری درخت تصمیم گیری حالتی است که هر ویژگی تعداد کمی از مقادیر را بتواند بگیرد (مثلاً فقط Hot، Mild و Cold). با این وجود، با الحاقتی به الگوریتم‌های اصلی، که در بخش ۳.۷.۲ بحث خواهد شد، می‌توان ویژگی‌ها را از گسسته مقدار به حقیقی مقدار تغییر داد (مثلاً Temperature را با درجه مشخص کرد).
- تابع هدف مقادیر، خروجی گسسته داشته باشد. درخت تصمیم گیری شکل ۳.۱ مقادیر منطقی را به هر یک از نمونه‌ها نسبت می‌دهد. متد های یادگیری درختی با افزایش تعداد مقادیر تابع هدف به راحتی به الگوریتم‌های یادگیری توابع گسسته مقدار تعمیم پیدا می‌کنند. با تعمیمی قابل توجه تر می‌توان توابعی با مقادیر حقیقی را با این متدها یاد گرفت، با این وجود استفاده از یادگیری درختی در یادگیری توابع حقیقی متداول نیست.
- زمانی که هدف یادگیری توضیحات فصلی است. همان طور که پیش‌تر نیز گفته شد، یادگیری درختی ذاتاً روابط فصلی را یاد می‌گیرد.
- داده های آموزشی می‌توانند خطا داشته باشند. متد های یادگیری درختی می‌توانند خود را با خطای موجود در داده های آموزشی وفق دهند، فرقی ندارد که مقدار تابع هدف نمونه بوده یا یکی از ویژگی‌ها اشتباه گزارش شده باشد.
- نمونه های آموزشی می‌توانند ویژگی‌های مجهول داشته باشند. یادگیری درختی را حتی زمانی که نمونه های آموزشی ویژگی‌های مجهول دارند می‌توان به کار برد (مثلاً اگر ویژگی Humidity برای بعضی از روزها معلوم نباشد). این حالت در بخش ۳.۷.۴ بررسی خواهد شد.

بسیاری از مسائل کاربردی دارای ویژگی‌هایی فوقند، به همین خاطر یادگیری درخت تصمیم گیری بسیار پر کاربرد شده است تا جایی که در مسایلی نظیر تشخیص موارد پزشکی، تشخیص دلیل خرابی تجهیزات و تشخیص ریسک وام بر اساس عقب افتادگی قسطها به کار می‌روند. به چنین مسائلی، که هدف از یادگیری دسته بندی نمونه‌ها در یکی از دسته های موجود است، مسائل دسته بندی[1] می‌گویند.

در ادامه‌ی این فصل بدین ترتیب بحث را پی می‌گیریم: در بخش ۳.۴ الگوریتم اساسی ID3 را در یادگیری درختی و نحوه‌ی کار آن را توضیح خواهیم داد. در قسمت ۳.۵ جستجوی این الگوریتم در فضای فرضیه ای را بررسی و آن را با الگوریتم‌های فصل ۲ مقایسه خواهیم کرد. در بخش ۳.۶ بایاس‌های استقرایی این الگوریتم یادگیری درختی را بررسی خواهیم کرد و با بایاسی کلی‌تر به نام تیغ Occam[2] آشنا خواهیم شد که ترجیح درخت‌های کوچک‌تر و ساده تر را در میان فضای فرضیه ای توجیه می‌کند. در بخش ۳.۷ پدیده‌ی overfit را بررسی خواهیم کرد و استراتژی‌های هرس[3] را برای حل این مسئله بیان خواهیم کرد. در این قسمت در مورد مباحث پیشرفته تر دیگری نیز مثل چگونگی تعمیم یادگیری درختی برای یادگیری توابع حقیقی مقدار، یادگیری با ویژگی‌های مجهول و ویژگی‌های غیر هم هزینه نیز بحث شده است.

[1] classification problems
[2] Occam's razor
[3] post-pruning

۳.٤ الگوریتم اساسی یادگیری درختی

اکثر الگوریتم‌هایی که برای یادگیری درختی ایجاد شده نسخه های مختلف یک الگوریتم اساسی هستند که از جستجویی حریصانه[1] و بالا به پایین[2] برای جستجوی فضای درخت‌های تصمیم گیری ممکن استفاده می‌کند. این روش الگوریتم ID3 نام دارد (Quinlan 1986) و تکامل یافته‌ی این الگوریتم نیز C4.5 نامیده می‌شود (Quinlan 1993). این دو الگوریتم موضوع بحث این بخش هستند. در این بخش الگوریتم پایه ای یادگیری درختی را معرفی می‌کنیم، این الگوریتم تقریباً همان ID3 است. در قسمت ۳.۷ نیز تعدادی از تعمیم‌های این الگوریتم، تعمیم‌های مربوط به الگوریتم C4.5 و چند الگوریتم دیگر، را توضیح خواهیم داد.

الگوریتم اساسی ما، یا همان ID3، درخت تصمیم گیری متناسب را با جستجویی بالا به پایین پیدا می‌کند، این جستجو با طرح این سؤال آغاز می‌شود "چه ویژگی‌ای باید در ریشه‌ی درخت بررسی شود؟" برای جواب دادن به این سؤال، تمامی ویژگی‌ها در تمامی نمونه‌ها توسط یک بررسی آماری بررسی می‌شود تا معلوم گردد تا کدام ویژگی به تنهایی تأثیر بیشتری بر دسته بندی نمونه‌ها دارد. سپس بهترین ویژگی انتخاب می‌شود و به عنوان گره ریشه‌ی درخت قرار می‌گیرد. برای هر مقدار این ویژگی انتخابی در ریشه‌ی درخت نمونه های آموزشی ترتیب می‌شوند و با توجه به این گره مسئله به مسئله های کوچک‌تر تبدیل می‌شود (هر نمونه‌ی آموزشی از طرف شاخه ای پایین می‌رود که مقدار آن با مقدار ویژگی نظیرش مطابق باشد). این فرایند برای زیر شاخه‌ها آنقدر اجرا می‌شود تا بالاخره هر نمونه درست دسته بندی شود، در هر تکرار همیشه ویژگی انتخابی برای گره ویژگی‌ای است که مهم‌ترین اثر را در دسته بندی دارد. آن طور که شرح داده شد، با یک فرایند حریصانه به جستجوی بهترین درخت ممکن می‌پردازیم، یعنی اینکه هیچ وقت الگوریتم به انتخاب‌هایی که قبلاً کرده بازنگری نمی‌کند. ساده شده‌ی این الگوریتم (برای یادگیری توابع منطقی مقدار، یا همان یادگیری مفهوم) در جدول ۳.۱ آمده است.

ID3 (Examples, Target_attribute, Attributes)

Examples مجموعه‌ی تمامی نمونه های آموزشی است. Target_attribute ویژگی‌ای است که مقادیرش توسط درخت پیش بینی می‌شود. Attributes لیستی از دیگر ویژگی‌هایی است که ممکن است توسط درخت بررسی شود. این الگوریتم درخت تصمیم گیری‌ای را که به درستی نمونه های داده شده را دسته بندی می‌کند بر می‌گرداند.

- گره ای برای ریشه‌ی درخت ایجاد کن
- اگر تمامی نمونه‌های Examples، نمونه‌ی مثبتند ریشه را با + علامت گذاری[3] کن و درخت را خروجی بده.
- اگر تمامی نمونه‌های Examples، نمونه‌ی منفی‌اند ریشه را با – علامت گذاری کن و درخت را خروجی بده.
- اگر مجموعه‌ی Attributes تهی است، ریشه را با متداول‌ترین دسته بندی نمونه‌ها علامت گذاری کن و درخت را خروجی بده.
- در غیر این صورت :
  - *A را ویژگی‌ای قرار بده که Examples را بهترین دسته بندی می‌کنند.*
  - *ویژگی متناسب با گره ریشه را A قرار بده.*
  - *برای هر مقدار v_i از A*
    - *یک شاخه‌ی جدید در زیر ریشه متناسب با مقدار v_i اضافه کن.*

---

[1] greedy
[2] top-down
[3] label

- $Examples_{v_i}$ را زیر مجموعه ای از Examples قرار بده که مقدار v_i را برای ویژگی A دارند
- اگر $Examples_{v_i}$ تهی بود،
  - در زیر شاخه‌ی جدید گره برگی اضافه کن و آن را با متداول‌ترین مقدار Target_attribute در Examples علامت گذاری کن.
  - در غیر این صورت، در زیر این شاخه‌ی جدید زیر درخت $ID3(Examples_{v_i}, Target_{attribute}, Attributes - \{A\})$ را اضافه کن.
- درخت ایجاد شده را برگردان.

*: بهترین ویژگی‌ای است که بالاترین مقدار بهره‌ی اطلاعات (که در رابطه‌ی ۳.۴ آمده) را داشته باشد.

جدول ۳.۱ خلاصه‌ی الگوریتم ID3 برای یادگیری توابع منطقی مقدار.

ID3 یک الگوریتم حریصانه است که درخت را از بالا به پایین رشد می‌دهد، در هر گره ویژگی‌ای انتخاب می‌شود که نمونه های آموزشی در آن ناحیه را بهتر دسته بندی کند. این فرایند آنقدر ادامه پیدا می‌کند تا درخت به طور کامل تمام نمونه های آموزشی را درست دسته بندی نماید، یا اینکه تمامی ویژگی‌ها استفاده شوند.

۳.٤.۱ کدام ویژگی بیشترین نقش را در دسته بندی دارد؟

مهم‌ترین انتخابی که در الگوریتم ID3 انجام می‌گیرد انتخاب ویژگی‌ای که در هر گره از درخت بررسی می‌شود است. ما ترجیح می‌دهیم که این ویژگی، ویژگی‌ای باشد که بیشترین تأثیر را در دسته بندی نمونه‌ها دارد. چه معیاری را می‌توان معیار خوبی برای برتری یک ویژگی دانست؟ در اینجا خاصیتی آماری به نام بهره‌ی اطلاعات[1] را تعریف می‌کنیم که میزان تأثیر یک ویژگی را بر دسته بندی نمونه‌ها بر اساس دسته بندی تابع هدفشان اندازه گیری می‌کند. ID3 از بهره‌ی اطلاعات برای انتخاب ویژگی در هر مرحله از رشد درخت استفاده می‌کند.

۳.٤.۱.۱ آنتروپی، معیار یکدستی نمونه‌ها

برای تعریف دقیق بهره‌ی اطلاعات از تعریف معیار دیگری به نام آنتروپی[2]، که در تئوری اطلاعات[3] کاربرد بسیار دارد، شروع می‌کنیم. این معیار یکدستی[4] و عدم یکدستی یک دسته‌ی دلخواه از نمونه‌ها را مشخص می‌کند. با داشتن دسته‌ی S از نمونه‌های مثبت و منفی مفهوم هدف، آنتروپی دسته‌ی S متناسب با این دسته بندی منطقی به صورت زیر تعریف می‌شود:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \qquad (3.1)$$

در این رابطه $p_{\oplus}$ نسبت تعداد نمونه های مثبت به تعداد کل نمونه‌ها و $p_{\ominus}$ نیز نسبت تعداد نمونه های منفی به تعداد کل نمونه‌هاست. همیشه در محاسبه‌ی آنتروپی فرض می‌کنیم که 0log0، صفر است.

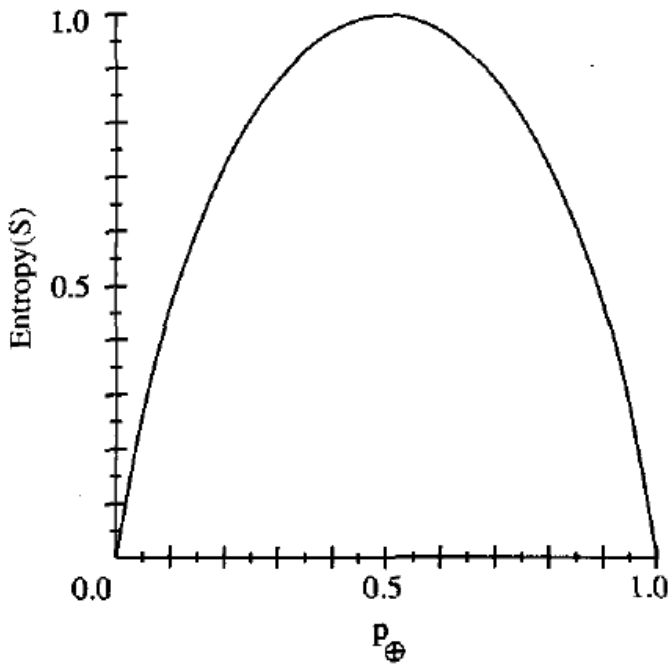
برای درک بهتر، فرض کنید که مجموعه‌ی S شامل ۱۴ نمونه از مفهومی منطقی باشد، از این ۱۴ نمونه ۹ نمونه مثبت و ۵ نمونه منفی هستند (برای خلاصه سازی به طور خلاصه می‌نویسیم [9+,5-]). آنتروپی مربوط به این مجموعه‌ی زیر خواهد بود:

$$Entropy([9+,5-]) = -\left(\frac{9}{14}\right)\log_2\frac{9}{14} - \left(\frac{5}{14}\right)\log_2\frac{5}{14}$$

[1] information gain
[2] entropy
[3] information theory
[4] Homogeneity

$$= 0.940 \qquad (3.2)$$

توجه داشته باشید که زمانی آنتروپی صفر است که تمامی اعضای S از یک نوع دسته بندی باشند. برای مثال اگر تمامی نمونه‌ها مثبت باشند $(p_{\oplus} = 1)$ پس $p_{\ominus}$ صفر خواهد بود و داریم که $Entropy(S) = -1\log_2 1 - 0\log_2 0 = -1 * 0 - 0\log_2 0 = 0$. همچنین توجه دارید که زمانی آنتروپی ۱ است که تعداد نمونه های مثبت و منفی مساوی باشد. همیشه مقدار آنتروپی مقداری بین ۰ و ۱ است. شکل ۳.۲ شکل تابع آنتروپی را برای یک تابع منطقی مقدار بر حسب $p_{\oplus}$ نشان می دهد.



شکل ۳.۲ میزان آنتروپی برای دسته بندی منطقی، برچسب مقدار نسبی $p_{\oplus}$.

یکی از تفسیر های آنتروپی که در تئوری اطلاعات مطرح می‌شود حداقل تعداد بیت‌های لازم برای کد کردن یکی از اعضای دلخواه S است (برای مثال یک عضو تصادفی با احتمال یکنواخت). مثلاً اگر $p_{\oplus}$ یک باشد، دریافت کننده‌ی اطلاعات می‌داند که نمونه‌ی انتخابی حتماً مثبت خواهد بود، پس نیازی به ارسال داده ای نیست، آنتروپی صفر است. از سوی دیگر، اگر $p_{\oplus}$ ۰.۵ باشد، برای ارسال هر نمونه دقیقاً ۱ بیت لازم خواهد بود تا برای دریافت کننده معلوم گردد که نمونه مثبت بوده یا منفی. و اگر $p_{\oplus}$ ۰.۸ باشد، مجموعه ای از اعضا را می‌توان با متوسط کمتر از ۱ بیت برای هر عضو کد کرد، در این کد، برای اعضای مثبت کد کوتاه‌تر و برای اعضای منفی کد بلند تری مورد استفاده قرار می‌گیرد.

تعریف بالا تعریف آنتروپی برای توابع هدف منطقی است، در حالت کلی‌تر اگر ویژگی هدف بتواند c مقدار متفاوت داشته باشد، آنتروپی S برای این دسته بندی c حالتی به صورت زیر تعریف می‌شود:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \qquad (3.3)$$

در این رابطه p_i نسبتی از S است که مقدار i را دارد. توجه دارید که پایه‌ی لگاریتم همچنان ۲ باقی می‌ماند زیرا که آنتروپی متوسط تعداد بیت‌های لازم برای ارسال اطلاعات را حساب می‌کند. همچنین توجه داشته باشید که اگر ویژگی هدف c حالت ممکن داشته باشد، مقدار آنتروپی حداکثر $\log_2 c$ خواهد بود.

۳.٤.۱.۲ بهره‌ی اطلاعات، معیار کاهش انتظاری آنتروپی

با داشتن آنتروپی به عنوان معیاری برای میزان یکدستی مجموعه ای از نمونه های آموزشی، حال می‌توانیم معیاری برای تأثیر گذاری یک ویژگی در دسته بندی نمونه های آموزشی ارائه دهیم. همان طور که گفته شد این معیار بهره‌ی اطلاعات نامیده می‌شود. بهره‌ی اطلاعات میزان کاهش انتظاری آنتروپی از دسته بندی بر اساس ویژگی خاص است. به عبارت دقیق‌تر، بهره‌ی اطلاعات ویژگی A بر روی مجموعه‌ی S، Gain(S,A) را بر حسب مجموعه‌ی نمونه های موجود به شکل زیر تعریف می‌کنیم:

$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \qquad (3.4)$$

در این رابطه مقدار Values(A) دسته تمام مقادیر ممکن برای ویژگی A است و S_v زیر تعداد نمونه‌هایی از S هستند که برای ویژگی A مقدار v را دارند ($S_v = \{s \in S | A(s) = v\}$). توجه داشته باشید که جمله‌ی اول در رابطه‌ی ۳.۴ فقط خود آنتروپی مجموعه‌ی S است و جمله‌ی دوم میانگین آنتروپی بعد از تقسیم S با ویژگی A است. میانگین یا امید آنتروپی‌ای که در این جمله آمده است همان مجموع آنتروپی برای تمامی S_v هاست که در نسبت نمونه‌ها $\frac{|S_v|}{|S|}$ ضرب شده است. پس بنابراین Gain(S,A) امید کاهش آنتروپی با تقسیم بندی بر اساس ویژگی A است. به عبارت دیگر، Gain(S,A) میزان اطلاعاتی است که در مورد مقدار تابع هدف با داشتن مقدار ویژگی A بدست می‌آوریم. مقدار Gain(S,A) تعداد بیت‌هایی است که در کد کردن مقدار تابع هدف بر روی اعضای دلخواه S با داشتن مقدار ویژگی A آن‌ها صرفه جویی می‌شود.

برای مثال، فرض کنید که S مجموعه‌ی نمونه های آموزشی روزها باشد که توسط ویژگی Wind با مقادیر Strong و Weak توصیف می‌شود. همان طور که قبلاً هم داشتیم این مجموعه‌ی S، ۱۴ نمونه دارد، [9+, 5-]. از این ۱۴ نمونه برای مقدار "Wind = Weak" ۶ نمونه‌ی مثبت و ۲ نمونه‌ی منفی داریم و بقیه‌ی نمونه‌ها برای مقدار "Wind = Strong" است. بهره‌ی اطلاعات با توجه به این ۱۴ نمونه را می‌توان به شکل زیر محاسبه کرد:

Values (Wind) = Strong, Weak

S = [9+, 5-]

$$S_{\text{Weak}} \leftarrow [6+, 2-]$$

$$S_{\text{Strong}} \leftarrow [3+, 3-]$$

$$Gain(S, \text{Wind}) = Entropy(S) - \sum_{v \in \{\text{Weak,Strong}\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

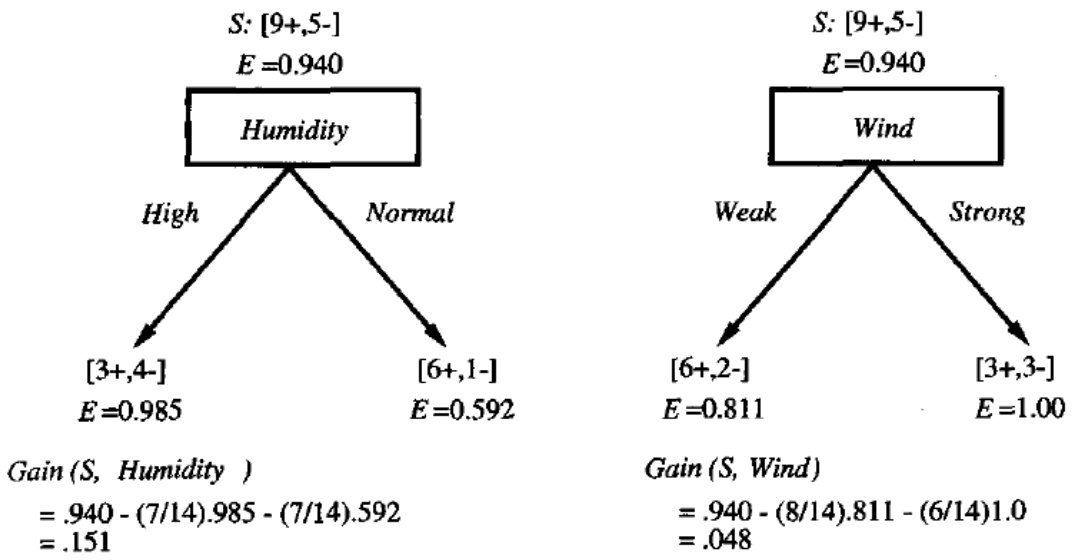
$$= Entropy(S) - \left(\frac{8}{14}\right) Entropy(S_{\text{Weak}})$$

$$- \left(\frac{6}{14}\right) Entropy(S_{\text{Strong}})$$

$$= 0.940 - \left(\frac{8}{14}\right) 0.811 - \left(\frac{6}{14}\right) 1.00$$

$$= 0.048$$

بهره‌ی اطلاعات دقیقاً معیاری است که در ID3 برای انتخاب بهترین ویژگی در هر مرحله از رشد درخت استفاده می‌شود. نمونه‌ای از استفاده از بهره‌ی اطلاعات برای تخمین میزان ارتباط ویژگی‌ها با تابع هدف در شکل ۳.۳ آمده است. در این شکل بهره‌ی اطلاعات برای دو ویژگی مختلف Humidity و Wind محاسبه شده تا معلوم گردد که کدام ویژگی، ویژگی بهتری برای دسته بندی نمونه‌های آموزشی آمده در جدول ۳.۲ است.



شکل ۳.۳ ویژگی Humidity بهره‌ی اطلاعاتی بیشتری برای دسته بندی نسبت به Wind دارد.
در این شکل E نماد آنتروپی است و S مجموعه‌ی اولیه‌ی نمونه های آموزشی است. با داشتن مجموعه‌ی اولیه‌ی S [9+,5-]، با استفاده از ویژگی Humidity دو زیر مجموعه‌ی [3+,4-] (برای Humidity=High) و [6+,1-] (برای Humidity=Normal) بدست می‌آید. بهره‌ی اطلاعات این تقسیم بندی 0.151 است که از مقدار نظیر در تقسیم بندی بر اساس (0.048) باد بیشتر است.

۳.٤.۲ یک مثال

برای تصور بهتر از عملکرد ID3، کار یادگیری که توسط نمونه های آموزشی جدول ۳.۲ بیان شده است را در نظر بگیرید. در اینجا ویژگی هدف ویژگی PlayTennis است، که مقادیر Yes و No دارد. مرحله‌ی اول الگوریتم را در نظر بگیرید، در این مرحله بالاترین قسمت درخت تشکیل می‌شود. چه ویژگی‌ای باید در این قسمت بررسی شود؟ ID3 بهره‌ی اطلاعات را برای تمامی ویژگی‌ها (Outlook،

Temperature، Humidity و Wind) تعیین می‌کند، سپس ویژگی‌ای را که بالاترین بهره‌ی اطلاعات را دارد بر می‌گزینـد. محاسـبه‌ی لازم برای دو مورد از این ویژگی‌ها در شکل ۳.۳ آمده است. مقادیر بهره‌ی اطلاعات محاسبه شده برای تمامی ویژگی‌ها به شرح زیر است:

Gain(S, Sky) = 0.246

Gain(S, Humidity) = 0.151

Gain(S, Wind) = 0.048

Gain(S, AirTemp) = 0.029

در این روابط S همان مجموعه‌ی نمونه‌های جدول ۳.۲ است.

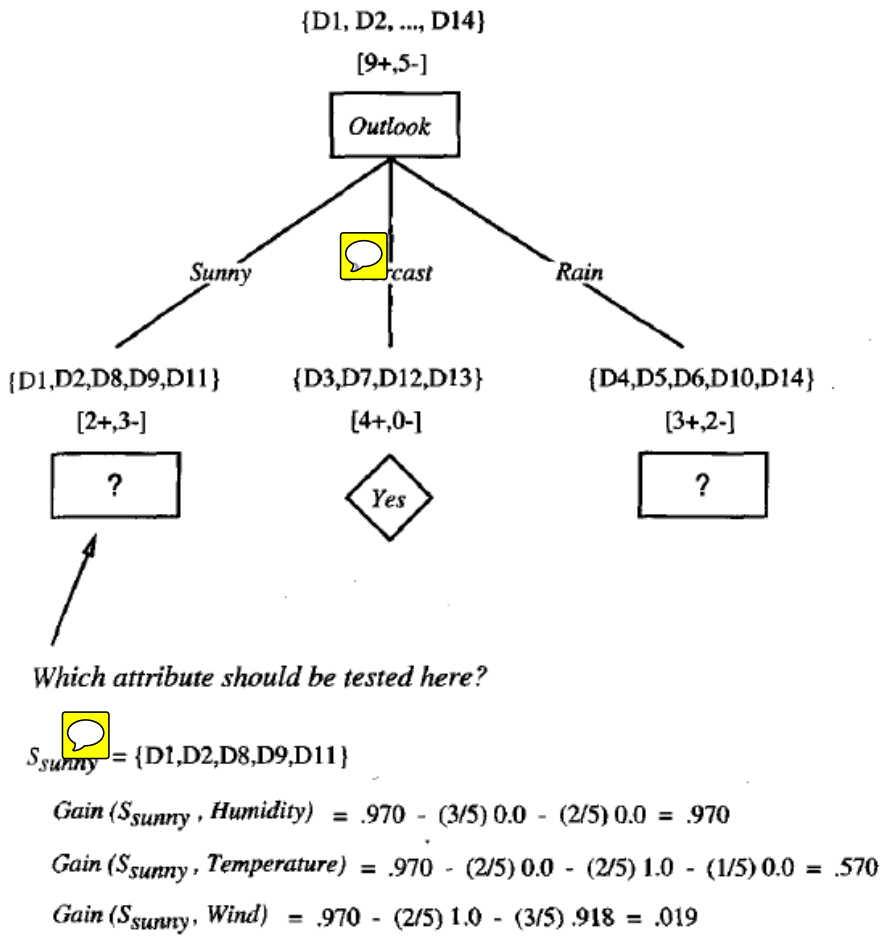
روز	Outlook	Temperature	Humidity	Wind	PlayTennis
روز ۱	Sunny	Warm	High	Weak	No
روز ۲	Sunny	Warm	High	High	No
روز ۳	Overcast	Warm	High	Weak	Yes
روز ۴	Rain	Mild	High	Weak	Yes
روز ۵	Rain	Cool	Normal	Weak	Yes
روز ۶	Rain	Cool	Normal	Strong	No
روز ۷	Overcast	Cool	Normal	Strong	Yes
روز ۸	Sunny	Mild	High	Weak	No
روز ۹	Sunny	Cool	Normal	Weak	Yes
روز ۱۰	Rain	Mild	Normal	Weak	Yes
روز ۱۱	Sunny	Mild	Normal	Strong	Yes
روز ۱۲	Overcast	Mild	High	Strong	Yes
روز ۱۳	Overcast	Hot	Normal	Weak	Yes
روز ۱۴	Rain	Mild	High	Strong	No

جدول ۳.۲ نمونه‌های آموزشی برای مفهوم هدف PlayTennis.

بنا بر بهره های اطلاعات محاسبه شده، ویژگی Outlook بیشترین تأثیر را بـر PlayTennis بـر روی نمونـه‌هـای آموزشـی دارد. بنـابراین Outlook بهترین ویژگی برای بررسی در گره ریشه‌ی درخت اسـت، و شـاخه‌هـا نیـز مقـادیر مختلـف ایـن ویژگـی (Sunny، Cloudy و Rainy) خواهند بود. درخت حاصل در شکل ۳.۴ به همراه نمونه‌های مربوط به هر شاخه نشان داده شده است. توجه داریـد کـه بـرای تمـامی نمونه‌هایی که Outlook=Cloudy است، مقدار PlayTennis نیز بله است. بنابراین این گره از درخت با PlayTennis=Yes علامـت گذاری می‌شود. در مقابل، برای دو وضع هوای Rainy و Sunny آنتروپی صفر نیست و درخت تصمیم گیری در زیر این شاخه‌ها رشد بیشتری خواهد کرد.

فرایند انتخاب یک ویژگی جدید و تقسیم نمونه‌ها دوباره برای گره های غیر پایانی[1] انجام می‌شود با این تفاوت که در این مرحله فقط نمونه‌هایی که با گره تطابق دارند مورد استفاده قرار می‌گیرند و ویژگی‌هایی که قبلاً استفاده شده‌اند از مجموعه‌ی مربوطه حذف می‌گردند تا هر ویژگـی در هر مسیر از ریشه تا برگ حداکثر یک بار ظاهر شود. این فرایند برای تمامی برگ‌های بدست آمده ادامه پیدا می‌کند تا یکـی از دو شـرط روبـرو درست شود: (۱) همه‌ی ویژگی‌ها استفاده شوند، (۲) نمونه‌های تمامی برگ‌ها از مقدار یکسانی از تابع هدف را داشته باشند (آنتروپـی‌شـان صـفر شود). شکل ۳.۴ محاسبه‌ی بهره‌ی اطلاعات برای مراحل بعدی رشد درخت را نشان می‌دهد. درخت کامل شده توسـط ID3 بـرای تمـامی ۱۴ نمونه‌ی جدول ۳.۲ در شکل ۳.۱ آمده است.

[1] nonterminal



شکل ۳.۴ درخت نیمه کاره ای که بعد از یک مرحله اجرای ID3 بدست می‌آید.
نمونه های آموزشی هر گره دسته بندی و جدا شده‌اند. مقدار Cloudy چون تنها نمونه‌های مثبت دارد پس با Yes علامت گذاری شده است. دو برگ دیگر با انتخاب ویژگی‌هایی که بهره‌ی اطلاعات بیشتر (برای نمونه های همان شاخه) دارند باز هم توسعه خواهند یافت.

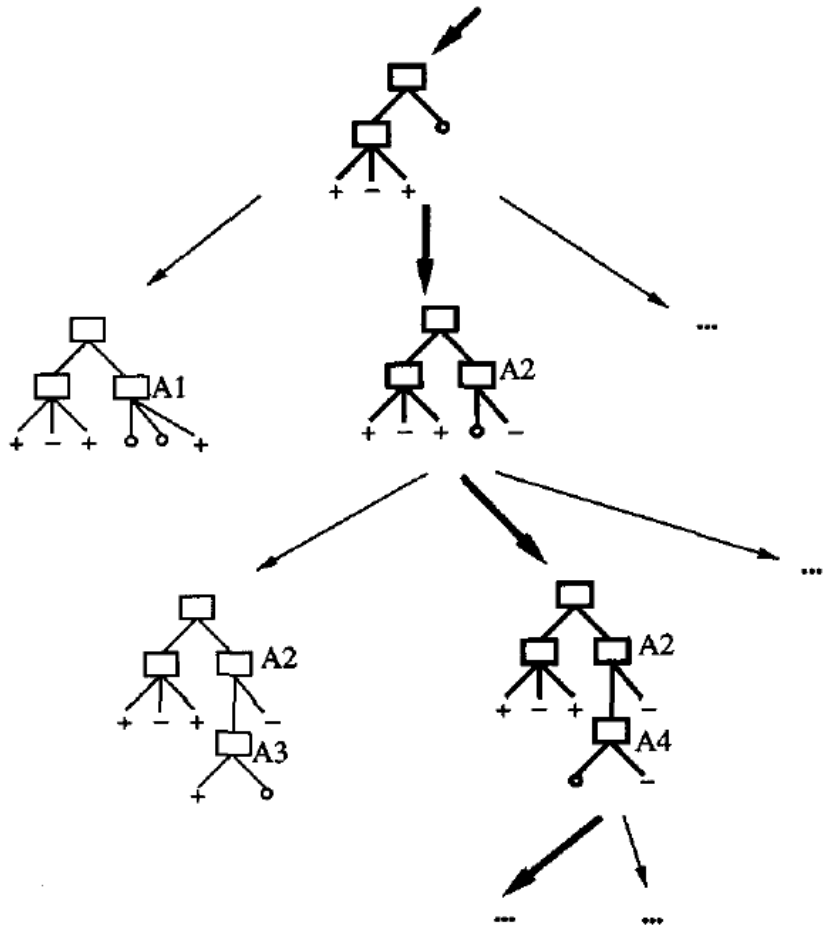
۳.۵ جستجو در فضای فرضیه‌ها در یادگیری درختی

مشابه دیگر متد های یادگیری استقرایی، ID3 را نیز می‌توان جستجویی در میان فضایی از فرضیه‌ها برای پیدا کردن متناسب‌ترین فرضیه با نمونه های آموزشی در نظر گرفت. فضای فرضیه ای که توسط ID3 جستجو می‌گردد مجموعه‌ی تمامی درخت‌های تصمیم گیری است. ID3 جستجوی ساده به پیچیده[1] و hill-climbing را در این فضای فرضیه ای انجام می‌دهد. ابتدای این جستجو درخت بسیار ساده‌ی تهی است، سپس با ادامه‌ی فرایند کم‌کم درخت جزئی‌تر می‌گردد تا به درختی برسد که بتواند تمامی نمونه های آموزشی را درست دسته بندی کند. تابع بهره‌ی اطلاعات این جستجوی hill-climbing را کنترل می‌کند. این جستجو در شکل ۳.۵ نشان داده شده است.

با نگاه به جنبه‌ی جستجویی و با توجه به فضای جستجو و استراتژی جستجوی ID3، تعدادی از قابلیت‌ها و محدودیت‌های آن مشاهده می‌شود:

[1] simple-to-complex

- فضای فرضیه ای ID3 که همان تمامی درخت‌های تصمیم گیری است تمامی توابع گسسته مقدار متناهی را با توجه به ویژگی‌های موجود در بر می‌گیرد. زیرا که هر تابع متناهی گسسته مقدار را می‌توان با درخت تصمیم گیری‌ای نشان داد، پس ID3 مشکلی (که ممکن است تابع هدف در فضای فرضیه ای نباشد) که خیلی از متدها (مثل متدهایی که فقط ترکیب‌های عطفی را در نظر می‌گیرند) در مورد محدودیت فضای فرضیه ای دارند را ندارد.



شکل ۳.۵ جستجوی ID3 در فضای فرضیه‌ای.

ID3 جستجویی ساده به پیچیده را در میان تمامی درخت‌های تصمیم گیری مختلف انجام می‌دهد. این جستجو توسط تابع بهره‌ی اطلاعات کنترل می‌شود.

- ID3 تنها یک فرضیه را در جستجو به دنبال فرضیه‌ی مطلوب در فضای درخت‌های تصمیم گیری دنبال می‌کند. این نوع جستجو با الگوریتم Candidate-Elimination که تمامی فرضیه های ممکن سازگار با نمونه های آموزشی را پیدا می‌کرد در تضاد است. با محدود شدن به یک فرضیه‌ی خاص، ID3 قابلیت معلوم کردن تمامی فرضیه های متناسب با نمونه های آموزشی را از دست می‌دهد. برای مثال، ID3 نمی‌تواند مشخص کند که چند درخت تصمیم گیری با نمونه های آموزشی مطابقند، یا اینکه نمی‌تواند بین این چندین درخت تصمیم گیری مختلف در زمان عدم هماهنگی رای گیری کند.

- نسخه‌ی اصلی ID3 هیچ بازنگری‌ای[1] به عملیات‌های قبلی در جستجویش نمی‌کند. زمانی که یک ویژگی را برای مرحله‌ی خاص در درخت انتخاب کرد هیچ گاه برای تغییر آن به این مرحله باز نمی‌گردد. بنابراین این احتمال وجود دارد که به ریسک‌های روش‌های hill climbing بدون بازنگری دچار شویم: احتمال دارد که به جای یک مینیمم مطلق به یک مینیمم موضعی میل کنیم. در مورد ID3، این مشکل با انتخاب درختی که به صورت موضعی و در مسیر مورد بررسی بهینه‌ترین است ایجاد می‌شود. با این وجود، این جواب موضعی بهینه شاید نسبت به درخت‌هایی که در دیگر شاخه های دیگر جستجو وجود دارند صلاحیت کمتری داشته باشد. در ادامه به تغییری در الگوریتم اصلی خواهیم پرداخت و نوعی بازنگری را به الگوریتم اضافه می‌کند. (هرس درخت)
- ID3 در مرحله‌ی جستجو از تمامی نمونه های آموزشی برای انتخاب آماری‌اش در چگونگی تغییر درخت فعلی استفاده می‌کند. این کار با متد های دیگر که تک‌تک به سراغ نمونه های آموزشی می‌روند در تضاد است (الگوریتم‌هایی چون Find-S یا Candidate-Elimination). یکی از مزایای استفاده از خواص آماری تمامی نمونه های آموزشی (خواصی چون بهره‌ی اطلاعات) این است که جستجو نسبت به خطاها حساسیت کمتری خواهد داشت. ID3 به راحتی می‌تواند با داده های خطا دار آموزشی نیز کار کند، فقط کافی است که شرط خروج را از دسته بندی درست تمامی نمونه های آموزشی به دسته بندی درست اکثریت نمونه های آموزشی تغییر دهیم.

۳.٦ بایاس استقرایی در یادگیری درختی

خط مشی که ID3 برای تعمیم بر روی داده های آموزشی استفاده می‌کند چیست؟ به عبارت دیگر، بایاس استقرایی ID3 چیست؟ با توجه به آنچه که در فصل ۲ گفته شد، بایاس استقرایی دسته فرض‌هایی است که علاوه بر داده های آموزشی فرض می‌شود تا بتوان تعمیم دسته بندی اعمالی یادگیر را توجیه کرد.

با معلوم بودن مجموعه‌ی نمونه های آموزشی، تعداد بسیار زیادی درخت تصمیم گیری سازگار با این نمونه‌ها را می‌توان مشخص کرد. توصیف بایاس استقرایی ID3 جواب این سؤال است که چرا ID3 با وجود درخت‌های سازگار بسیار با نمونه های آموزشی، درختی به خصوص را انتخاب می‌کند؟ این درخت چه ویژگی‌هایی دارد؟ الگوریتم ID3 اولین درخت قابل قبول را که در جستجوی ساده به پیچیده و hill-climbing آن با آن مواجه می‌شود را از میان تمامی درخت‌های ممکن بر می‌گزیند. استراتژی جستجوی ID3 را می‌توان به صورت روبرو توصیف کرد: (a) درخت‌های کوتاه‌تر نسبت به درخت‌های بلند تر ارجحیت دارند، (b) درختی برگزیده می‌شود که بهره‌ی اطلاعاتش در نزدیکی ریشه بیشتر باشد. به دلیل تأثیرات پیچیده‌ی انتخاب ویژگی‌ها در ID3 مشخص کردن بایاس استقرایی به طور دقیق کمی دشوار است. با این وجود، می‌توان به صورت کلی گفت که این الگوریتم درخت‌های کوتاه تر را بر درخت‌های بلند تر ترجیح می‌دهد.

بایاس استقرایی تخمینی ID3: درخت‌های کوتاه تر نسبت به درخت‌های بلند تر ارجحیت دارند.

در واقع، می‌توان گفت که الگوریتمی مشابه ID3 وجود دارد که دقیقاً بایاس استقرایی فوق را دارد. الگوریتمی را در نظر بگیرید که جستجوی میان فرضیه‌ها را با درختی تهی آغاز می‌کند و با جستجوی کم عمقی[2] (BFS) کم‌کم به طرف درخت‌های پیچیده تر می‌رود. یعنی ابتدا تمامی درخت‌هایی که عمق یک دارند را جستجو می‌کند سپس به سراغ عمق دو می‌رود و ... . این الگوریتم در مواجهه با درختی که با تمامی

[1] backtrack
[2] breath first search

نمونه های آموزشی سازگار است، کوچک‌ترین درخت ممکن را در عمق فعلی خروجی می‌دهد (مثلاً درختی که کمترین تعداد گره را داشته باشد). این الگوریتم به اختصار BFS-ID3 خوانده می‌شود. BFS-ID3 کوتاه‌ترین درخت تصمیم گیری‌ای سازگار با داده های آموزشی را پیدا می‌کند، یعنی دقیقاً بایاس استقرایی "درخت‌های کوتاه تر نسبت به درخت‌های بلند تر ارجحیت دارند" را داراست. ID3 را می‌توان تخمینی از BFS-ID3 دانست، با این تفاوت که جستجو در ID3، جستجویی حریصانه برای یافتن کوتاه‌ترین درخت ممکن است و تمامی فضای فرضیه‌ها جستجو نمی‌شود.

چون ID3 از بهره‌ی اطلاعات و استراتژی hill-climbing استفاده می‌کند بایاس پیچیده تری نسبت به BFS-ID3 دارد. در کل، همیشه کوتاه‌ترین درخت ممکن پیدا نمی‌شود و الگوریتم تمایل دارد که درخت‌هایی را انتخاب کند که بهره‌ی اطلاعاتشان در نزدیکی ریشه بیشتر باشد.

تخمینی مناسب‌تر از بایاس استقرایی ID3: درخت‌های کوتاه‌تری نسبت به درخت‌های بلند تر ارجحیت دارند. درخت‌هایی که بهره‌ی اطلاعات بیشتری در نزدیکی ریشه دارند نیز ارجحیت دارند.

۳.٦.۱ بایاس‌های محدود کننده و بایاس‌های مطلوب

تفاوت‌های جالبی میان دو نوع مختلف بایاس که توسط دو الگوریتم ID3 و Candidate-Elimination به کار برده می‌شود وجود دارد. به تفاوت‌های این دو روش جستجوی فضای فرضیه‌ها توجه کنید:

- ID3 فضای فرضیه ای کاملی را جستجو می‌کند (فضایی که تمامی توابع گسسته مقدار متناهی را می‌تواند توصیف کند). از طرفی این جستجو تمامی فضای فرضیه ای را شامل نمی‌شود، جستجو از فرضیه های ساده تر شروع شده و به محض رسیدن به شرط خروج پایان می‌یابد (مثلاً زمانی که فرضیه ای تمامی نمونه های آموزشی را درست دسته بندی می‌کند). بایاس استقرایی این الگوریتم فقط ناشی از نحوه‌ی ترتیب جستجوی این فرضیه‌هاست و بایاس دیگری وجود ندارد.
- الگوریتم Candidate-Elimination فضای فرضیه ای غیر کاملی را جستجو می‌کند (تنها زیر مجموعه ای از تمامی مفهوم‌هایی را که می‌توان از نمونه‌ها یاد گرفت). اما این فضای فرضیه ای را کامل جستجو می‌کند و تمامی فرضیه‌هایی که با نمونه های آموزشی سازگار هستند را پیدا می‌کند. بایاس استقرایی این الگوریتم فقط ناشی از میزان شمول فضای فرضیه ای آن است و استراتژی جستجویش هیچ نقشی در بایاس ندارد.

به طور خلاصه، بایاس استقرایی ID3 از استراتژی جستجویش ناشی می‌شود در حالی که بایاس استقرایی Candidate-Elimination ناشی از فضایی جستجو آن است.

پس، بایاس استقرایی ID3 یک ترجیح فرضیه‌ها بر دیگر فرضیه‌هاست، بدون اینکه فضای فرضیه ای هیچ محدودیتی ایجاد کند. این نوع بایاس را معمولاً بایاس ترجیحی[1] (یا بایاس جستجویی[2]) می‌نامند. در مقابل، بایاس Candidate-Elimination کاملاً به فضای فرضیه ای در نظر گرفته شده وابسته است. این نوع بایاس را نیز بایاس محدودیتی[3] (یا بایاس زبانی[4]) می‌نامند.

---

[1] preference bias
[2] search bias
[3] restriction bias
[4] language bias

با دانستن اینکه بایاس‌های استقرایی برای تعمیم نمونه‌ها هستند، (با توجه به آنچه که در فصل ۲ گفته شد)، کدام نوع از بایاس پسندیده تر است؟ بایاس ترجیحی یا بایاس محدودیتی؟

معمولاً، بایاس‌های ترجیحی به بایاس‌های محدودیتی ترجیح داده می‌شوند، زیرا که به یادگیر اجازه می‌دهند تا در فضای فرضیه ای کاملی که مطمئناً تابع هدف مجهول را در بر می‌گیرد کار کند. در مقابل، بایاس‌های محدودیتی که توابع قابل یادگیری را به دسته‌ی خاصی محدود می‌کنند از ارجحیت کمتری برخوردارند، زیرا که پیش‌فرضی را در مورد تابع هدف مجهول می‌گذارند.

با وجود اینکه دو الگوریتم بحث شده ID3 بایاسی کاملاً ترجیحی و Candidate-Elimination بایاسی کاملاً محدودیتی دارد، اما الگوریتم‌هایی وجود دارند که بایاسشان ترکیبی از این بایاس‌هاست. برای مثال، برنامه ای که در فصل ۱ برای تخمین عددی در یادگیری بازی‌ها توضیح داده شد را در نظر بگیرید. در این برنامه، توابع تخمین یاد گرفته شده ترکیب‌های خطی دسته ای از ویژگی‌های صفحه بودند، و الگوریتم یادگیری پارامتر های این ترکیب خطی را تعیین می‌کرد با سازگاری بیشتری با نمونه های آموزشی داشته باشد. در این مثال، اینکه از توابع خطی برای نمایش تابع تخمین استفاده کنیم نوعی بایاس محدودیتی است (توابع غیر خطی را نمی‌توان با این نمایش نشان داد). از طرف دیگر، اینکه با روش خاصی (مثل الگوریتم LMS) پارامترها را تعیین کنیم بایاسی ترجیحی است، بایاسی که باعث می‌شود تمامی پارامتر های ممکن را بررسی نکنیم.

۳.٦.۲ چرا فرضیه های کوتاه تر ارجحیت دارند؟

آیا ترجیح ID3 برای درخت‌های کوتاه‌تر برای تعمیم نمونه های آموزشی مفید است؟ فیلسوفان سال‌ها درباره‌ی چنین سؤالی بحث کرده و می‌کنند. ویلیام او اوکام (William of Occam) یکی از اولین افرادی بود که درباره‌ی بحث‌هایی را مطرح کرد (سال 1320)، به همین دلیل، این نوع بایاس را بایاس تیغ Occam می‌نامند.

بایاس تیغ Occam: در میان فرضیه های سازگار، فرضیه‌هایی که ساده‌ترند ارجحیت دارند.

البته با نام‌گذاری یک بایاس نمی‌توان آن‌را توجیه کرد. حال چرا باید فرضیه های ساده تر ارجح باشند؟ توجه دارید که دانشمندان بعضی مواقع چنین بایاسی را مورد استفاده قرار می‌دهند. برای مثال، فیزیکدانان نظریه های ساده تر درباره‌ی حرکت سیارات را ترجیح می‌دهند. چرا؟ یکی از استدلال‌های ممکن این است که تعداد فرضیه های ساده تر نسبت به فرضیه های پیچیده تر بسیار کمتر است، پس به نظر می‌رسد احتمال اینکه فرضیه ای پیدا شود که به طور اتفاقی با نمونه های آموزشی سازگار باشد کم است. در مقابل، تعداد بسیار زیادی از فرضیه های پیچیده موجود است که با نمونه های آموزشی سازگارند اما در تعمیم نمونه های آموزشی عاجزند. برای مثال، فرضیه های درخت‌های تصمیم گیری را در نظر بگیرید. تعداد درخت‌هایی که ۵۰۰ گره دارند بسیار بیشتر از درخت‌هایی است که ۵ گره دارند. اگر ۲۰ نمونه‌ی آموزشی داشته باشیم، تعداد بسیار زیادی از درخت‌های ۵۰۰ گره ای با آن‌ها سازگارند، در حالی که جای تعجب ندارد که فقط یک درخت ۵ گره ای متناسب با آن ۲۰ نمونه‌ی آموزشی باشد. بنابراین احتمال اینکه سازگاری درختی با ۵ گره اتفاقی بوده باشد بسیار کمتر از احتمال اتفاقی بودن سازگاری درختی با ۵۰۰ گره است.

با بررسی‌های بیشتر، معلوم می‌گردد که اشکال کلی‌ای در استدلال بالا وجود دارد. با همین استدلال می‌توان گفت که باید درخت‌هایی که ۱۷ گره برگ و ۱۱ گره غیر برگ دارند که تمامی ویژگی‌های یازده‌گانه‌ی نمونه‌ها را به ترتیب بررسی می‌کنند احتمال اتفاقی بودن بسیار کمتری دارند، زیرا تعداد چنین درخت‌هایی بسیار کم است پس شانس اتفاقی بودن (بنا به استدلال بالا) بسیار کمتر خواهد بود. این اشکال اینجاست که

زیرمجموعه های کوچک بسیاری از فضای فرضیه‌ها وجود دارد که چنین تعداد کمی را دارند و پیدا کردن همه‌ی آن‌ها ساده نیست. پس چرا باید باور داشته باشیم زیرمجموعه‌ی درخت‌هایی که طول کوتاه‌تری دارند باید نسبت به آن دیگر زیر مجموعه های کوچک برتری داشته باشد؟

اشکال دومی که درباره‌ی این استدلال برای تیغ Occam[1] پیش می‌آید این است که اندازه‌ی یک فرضیه با روش خاصی مشخص می‌شود که در یادگیر تعبیه شده. اگر دو یادگیر با روش‌های مختلف اندازه گیری اندازه‌ی فرضیه بر روی یک مسئله به کار گرفته شوند در آخر فرضیه های خروجی متفاوتی خواهند داشت، در حالی که هر دو عملیات خود را توسط تیغ Occam توجیه شده می‌دانند. برای مثال، تابعی که در شکل ۳.۱ نشان داده شده است را می‌توان با درختی با یک گره نیز نشان داد، درختی که یادگیر برای دسته بندی نمونه‌ها از ویژگی XYZ استفاده می‌کند، ویژگی منطقی XYZ زمانی درست است که نمونه، نمونه‌ی مثبتی باشد و در غیر این صورت غلت است. بنابراین دو یادگیر که هر دو از تیغ Occam استفاده می‌کنند اگر یکی ویژگی XYZ و دیگری ویژگی‌های Outlook، Temperature، Humidity و Wind را استفاده کنند درخت‌های خروجی متفاوتی خواهند داشت.

این بحث آخر نشان می‌دهد که تیغ Occam در دو یادگیر که از یک مجموعه نمونه های آموزشی یکسان استفاده می‌کنند و فقط نمایش داخلی نمونه‌هایشان متفاوت است دو فرضیه‌ی کاملاً متفاوت بدهد. با دانستن این حقیقت ممکن است به طور کلی تیغ Occam را رد کنیم. با این وجود، سؤال اینکه کدام نمایش درونی ممکن است با تکامل[2] یا انتخاب طبیعی[3] ایجاد شود را در نظر بگیرید. جمعیتی از یادگیر های مصنوعی‌ای را که از طریق فرایند های تکاملی زاد و ولد، جهش و انتخاب به وجود آمده‌اند را در نظر بگیرید. و بیایید فرض کنیم که این فرایند تکاملی می‌تواند سیستم‌های ادراکی این یادگیرها را از نسلی به نسلی تغییر دهد، مشابه تغییر ویژگی‌های داخلی‌ای که عوامل جهان اطراف را با آن‌ها درک می‌کنند. و برای بحث، فرض کنیم که این عوامل یادگیری از الگوریتم یادگیری یکسانی (مثلاً ID3) استفاده می‌کنند که با تکامل تغییر نخواهد یافت. منطقی است که فرض کنیم در طول زمان، تکامل نمایش‌های داخلی را ایجاد کند که موفقیت فرد در ارتباط با محیط را افزایش دهد. فرض کنیم که موفقیت یک عامل به شدت وابسته به قدرت تعمیمش دارد، بنابراین می‌توان انتظار داشت که نمایش‌های داخلی‌ای که خوب با الگوریتم یادگیری و بایاس استقرایی‌اش کار می‌کنند ایجاد شوند. اگر گونه‌هایی از یادگیرها از الگوریتم یادگیری از که از بایاس استقرایی تیغ Occam استفاده می‌کنند، وجود داشته باشد، انتظار خواهیم داشت که تکامل نمایش‌های داخلی‌ای را ایجاد کند که تیغ Occam برایشان استراتژی موفقی است. نکته اصلی بحث در اینجا این است که تکامل نمایش‌های داخلی‌ای را ایجاد خواهد کرد که بایاس الگوریتم یادگیری نوعی خود توجیهی[4] داشته باشد، زیرا که می‌تواند بسیار راحت‌تر از تغییر الگوریتم، نمایش را تغییر دهد.

فعلاً بحث مربوطه‌ی تیغ Occam را رها می‌کنیم. اما در فصل ۶، در قسمتی که قانون کمترین طول توضیح را بررسی خواهیم کرد، نسخه ای از تیغ Occam را که با چارچوب بیزی توجیه می‌شود را بررسی خواهیم کرد.

۳.۷ مشکلات یادگیری درختی

مشکلات کاربردی یادگیری درختی شامل، مشخص کردن حداکثر عمق درخت، چگونگی بررسی ویژگی‌های پیوسته، انتخاب معیار انتخاب ویژگی‌ها، یادگیری با داده‌هایی با بعضی ویژگی‌های مجهول، یادگیری با ویژگی‌های غیر هم هزینه و بهینه کردن محاسبات می‌شود. در ادامه

[1] Occam's razor
[2] evolution
[3] Natural selection
[4] self-fulfilling

هر یک از این موارد را بررسی خواهیم کرد و تغییراتی ID3 برای حل این مشکلات را نیز معرفی خواهیم کرد. C4.5 برای حل بعضی از این مشکلات ارائه شده است (Quinlar 1993).

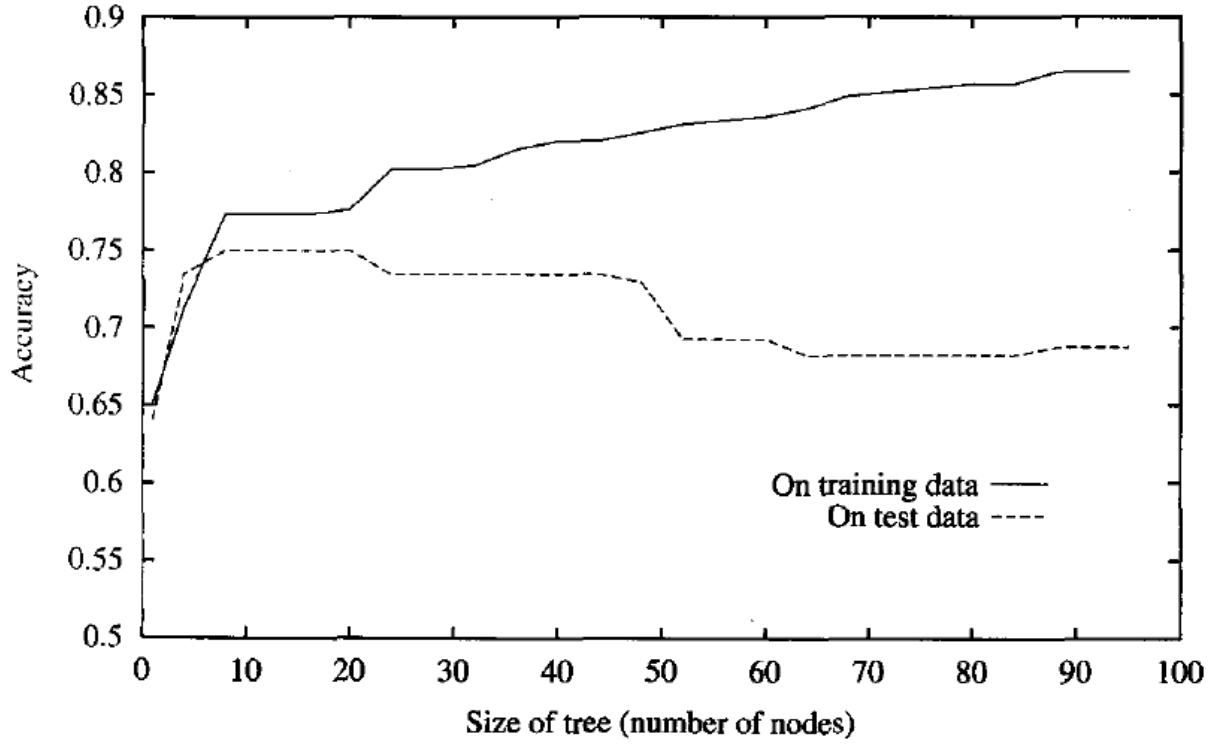
۳.۷.۱ حل مشکل overfit

الگوریتمی که در جدول ۳.۱ آمده است درخت را آنقدر رشد می‌دهد تا تمامی نمونه های آموزشی را درست دسته بندی کند. با وجود اینکه این استراتژی، استراتژی معقولی است اما همین استراتژی ممکن است مواقعی که داده‌ها خطا دارند یا تعدادشان به اندازه‌ی کافی نیست که تابع هدف را کامل تعریف کنند مشکل ساز باشد. به هر حال در چنین مواقعی، این الگوریتم درخت‌هایی را خروجی می‌دهد که مشکل overfit در نمونه های آموزشی دارند.

زمانی می‌گوییم که یک فرضیه مشکل overfit دارد که فرضیه ای دیگر موجود باشد که بر روی نمونه های آموزشی سازگاری کمتری داشته باشد اما در کل سازگاری بیشتری با کل نمونه‌ها (اعم از آموزشی و غیر آموزشی (جدید)) داشته باشد.

تعریف: اگر فضای فرضیه‌ها H باشد زمانی می‌گوییم که فرضیه مثل $h \in H$ مشکل overfit بر روی نمونه های آموزشی دارد که فرضیه ای مثل $h' \in H$ وجود داشته باشد به صورتی که خطای h بر روی نمونه های آموزشی نسبت به h′ کمتر باشد اما خطای h′ بر روی کل نمونه‌ها از خطای h کمتر باشد.

شکل ۳.۶ اثر پدیده‌ی overfit را در یک کاربرد Normal یادگیری درختی نشان می‌دهد. در این مثال، الگوریتم ID3 برای تشخیص بیماران دیابتی به کار رفته. محور افقی تعداد کل گره های درخت تصمیم گیری را در طول رشد درخت نشان می‌دهد. محور عمودی دقت تشخیص‌های درخت را نشان می‌دهد. منحنی توپر میزان دقت درخت را در نمونه های آموزشی و منحنی خط چین میزان دقت را بر روی دسته‌ی دیگری از نمونه‌ها نشان می‌دهد (دسته ای به جز نمونه های آموزشی). همان طور که پیش‌بینی می‌شد دقت درخت بر روی نمونه های آموزشی با افزایش اندازه‌ی درخت افزایش می‌یابد. با این وجود، دقت دسته‌ی دیگر کاهش می‌یابد. همان طور که دیده می‌شود، زمانی که اندازه‌ی درخت از حدود ۲۵ می‌گذرد، پیچیدگی بیشتر درخت باعث کاهش دقت در دسته‌ی دیگر می‌شود در حالی که دقت همچنان در نمونه های آموزشی بالا می‌رود.



شکل ۳.۶ overfit در یادگیری درختی.

همین طور که ID3 گره های بیشتری برای رشد درخت به آن اضافه می‌کند، به طور مشابه دقت بر روی نمونه های آموزشی افزایش پیدا می‌کند. با این وجود، زمانی که دقت بر روی دسته ای از نمونه های غیر آموزشی بررسی می‌شود، ابتدا افزایش و سپس کاهش مشاهده می‌شود. برنامه و داده های استفاده شده برای این آزمایش در http://www.cs.cmu.edu/~tom/mllbook.html موجود می‌باشد.

چگونه ممکن است درخت h که نسبت h′ عملکرد بهتری بر روی نمونه های آموزشی دارد، در کل نمونه‌ها عملکرد ضعیف‌تری داشته باشد؟ یکی از مواردی که چنین مشکلی ایجاد می‌شود مواقعی است که نمونه های آموزشی خطای تصادفی[1] یا همان نویز داشته باشند. برای تصور، نمونه‌ی آموزشی مثبت زیر را در نظر بگیرید که اشتباهاً نمونه‌ی منفی در نظر گرفته شده:

<Outlook = Sunny, Temperature=Hot,Humidity=Normal,Wind=Strong,PlayTennis=No>

با دادن داده های آموزشی بدون خطا به ID3 درخت نشان داده شده در شکل ۳.۱ بدست می‌آید. با این وجود اگر این نمونه‌ی اشتباه را به نمونه های آموزشی اضافه کنیم، ID3 درخت پیچیده تری خروجی می‌دهد. در کل، نمونه‌ی جدید در برگ دوم شاخه‌ی سمت چپ شکل ۳.۱ قرار می‌گیرد، همراه دو نمونه مثبت قبلی روز ۹ و روز ۱۱. حال چون که این نمونه منفی است، ID3 درخت را در زیر این شاخه بیشتر رشد می‌دهد. البته، تا زمانی که نمونه‌ی اشتباه این گونه باشد (فقط مقدار تابع هدف اشتباه تعیین شده باشد)، ID3 ویژگی‌ای خواهد یافت تا نمونه غلت را از نمونه‌ی درست جدا کند. نتیجه این خواهد بود که ID3 درخت تصمیم گیری h ای را خروجی می‌دهد که پیچیده تر از درخت تصمیم گیری درست h′ است (شکل ۳.۱). البته h بر روی نمونه های آموزشی دقت زیادی دارد در حالی که h′ ساده تر آن میزان دقت را ندارد. با این وجود، با داشتن گره ای که جدیداً اضافه شده و مستقیماً تأثیر نمونه‌ی خطا دار بوده، انتظار داریم که h از h′ دقت کلی بهتری داشته باشد.

[1] Random error

مثال بالا نشان داد که چگونه داده های خطا دار آموزشی می‌توانند باعث overfit شوند. در واقع، overfit حتی زمانی که نمونه های آموزشی خطا ندارند نیز ممکن است اتفاق بیفتد، مخصوصاً زمانی که تعداد نمونه‌ها با تعداد برگ‌های درخت برابر می‌شود. در چنین شرایطی، دور از انتظار نیست که نظم‌های اتفاقی در درخت پدیدار شوند، در این نظم‌ها به نظر می‌رسد که ویژگی‌های خاصی در دسته بندی نمونه‌ها تأثیر بسیار زیادی دارند در حالی که آن ویژگی‌ها هیچ ربطی به تابع هدف ندارند. هر گاه چنین نظم‌های اتفاقی‌ای ایجاد می‌شود، احتمال overfit نیز بالا می‌رود.

مشکل overfit مشکل قابل توجهی در یادگیری درختی و بسیاری از متد های یادگیری دیگر است . برای مثال، در یک مطالعه‌ی آزمایشی ID3 که بر روی ۵ کار یادگیری و با داده های خطا دار و غیر قطعی[1] انجام شد (Mingers 1989b)، در اکثر مسایل overfit دقت را بین ۱۰ تا ۲۵ درصد کاهش داد.

روش‌های بسیاری برای حل مسئله‌ی overfit در یادگیری درختی موجود است. این روش‌ها به دو دسته‌ی کلی تقسیم می‌شوند:

- روش‌هایی که جلوی رشد درخت را قبل از رسیدن به نقطه ای که تمامی نمونه‌ها را درست دسته بندی کند می‌گیرند،
- روش‌هایی که اجازه می‌دهند تا درخت به اندازه‌ی دلخواه رشد کند سپس درخت را هرس[2] می‌کنند.

با وجود اینکه به نظر می‌رسد روش‌های دسته‌ی اول مستقیم‌ترند، اما روش‌های دسته‌ی دوم در کاربرد موفقیت بیشتری را از خود نشان داده‌اند. از آنجا که در روش اول معلوم نیست که چه زمان باید جلوی رشد درخت گرفته شود.

جدا از اینکه درخت با کدام روش درخت به اندازه‌ی اصلی می‌رسد، سؤال کلیدی این است که معیار درست اندازه نهایی ی درخت چیست؟ روش‌های زیر برای جواب به این سؤال پیشنهاد می‌شوند:

- استفاده از دسته ای از نمونه های اضافی (که با نمونه های آموزشی تداخل ندارند) برای تخمین کارایی گره‌ها و هرس آن‌ها.
- استفاده از تمام نمونه های موجود برای آموزش، استفاده از آزمونی آماری برای تخمین اینکه آیا رشد (یا هرس) یک گره از درخت تعمیمی را ایجاد می‌کند یا تنها باعث overfit می‌شود. برای مثال، (Quinlar 1986) از آزمون کای اسکوار (کی دو)[3] برای جواب سؤال "آیا رشد یک گره به کارایی کلی درخت کمک می‌کند یا فقط باعث سازگاری با نمونه‌ی آموزشی می‌شود؟" استفاده می‌کند.
- استفاده از معیاری برای اندازه گیری پیچیدگی. در نظر گرفتن نوعی کد سازی برای درخت و متوقف کردن رشد درخت زمانی که این اندازه‌ی کد کمینه می‌شود. این روش مبتنی بر توجیه است که "کمترین طول توضیح"[4] نامیده می‌شود و مفصلاً در فصل ۶ بررسی شده. برای اطلاعات بیشتر به (Quinlar and Rivest 1989) و (Mehta 1995) مراجعه کنید.

روش اول متداول‌ترین روش است و گاهی روش آموزش و مجموعه‌ی تایید[5] نیز نامیده می‌شود. در اینجا به دو نسخه‌ی اصلی این روش می‌پردازیم. در این روش، داده های موجود به دو دسته تقسیم می‌شوند: دسته‌ی آموزشی، که از آن‌ها برای آموزش درخت استفاده می‌شود، و دسته‌ی تایید[6]، که از آن برای ارزیابی دقت فرضیه‌ها استفاده می‌شود، در کل، برای ارزیابی تأثیر هرس از این داده‌ها استفاده می‌شود. انگیزه‌ی

---

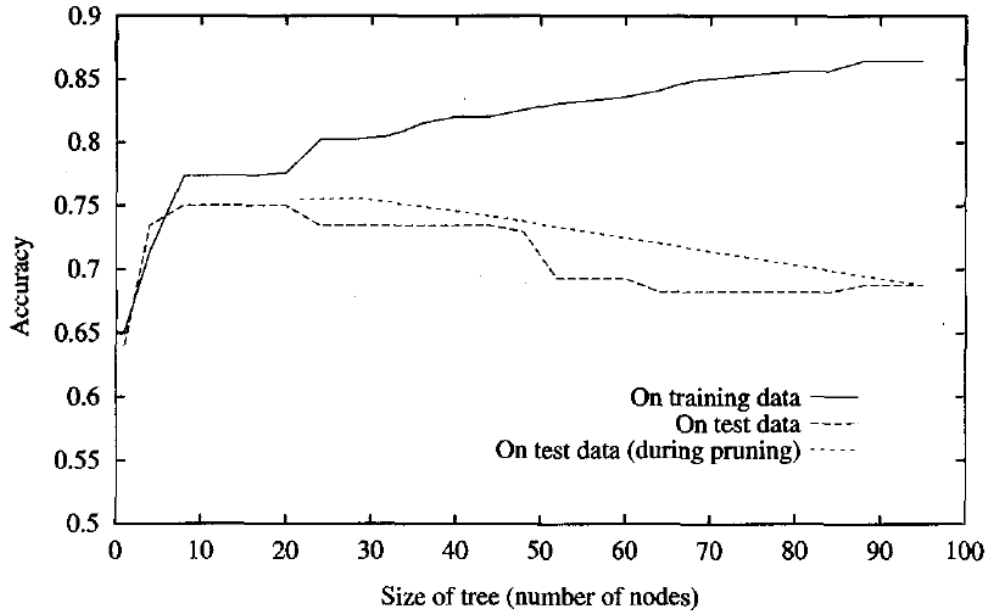
[1] nondeterministic
[2] post-prune
[3] chi-square
[4] Minimum Description Length principle
[5] training and validation set
[6] validation set

اولیه‌ی این روش این است که اگر چه ممکن است یادگیر با داده های خطا دار گمراه شود، و به نظم‌های تصادفی در میان نمونه های آموزشی میل کند، اما احتمال اینکه دسته‌ی تایید نیز همان نظم‌های اتفاقی که نمونه های آموزشی دارند را داشته باشد بسیار کم است. بنابراین، می‌توان انتظار داشت که دسته‌ی تایید معیار خوب و مطمئنی در مقابل معیار غلت داده های آموزشی به ما بدهد. البته، مهم است که اندازه‌ی دسته‌ی تایید به اندازه‌ی کافی بزرگ باشد تا بتواند به تنهایی نمونه‌ی کاملی از نمونه‌ها را داشته باشد. یکی از ایده های معمول این است که یک سوم کل داده‌ها را برای دسته‌ی تایید نگه می‌دارند و از دو سوم دیگر برای آموزش درخت استفاده می‌کنند.

۳.۷.۱.۱ کاهش خطا با هرس کردن

دقیقاً چگونه می‌توان با استفاده از یک دسته‌ی تایید از overfit جلوگیری کرد؟ یکی از روش‌های ممکن reduced-error pruning نام دارد (Quinlar 1987)، بر اساس این روش تمامی گره های درخت مستعد هرس شدن هستند. البته هرس کردن یک گره تصمیم باعث حذف شدن زیر درخت متصل به گره مذکور و تبدیل آن به برگ تبدیل خواهد شد با علامت متداول‌ترین دسته بندی از داده های آموزشی می‌شود. گره‌هایی هرس خواهند شد که هرس شدنشان باعث تأثیر منفی در دسته بندی دسته‌ی تایید نشود. این عمل باعث حذف برگ‌هایی که بر اثر نظم‌های تصادفی داده های آموزشی ایجاد شده‌اند می‌شوند، زیرا که احتمال تکرار همان نظم‌های تصادفی در دسته‌ی تایید ناچیز است. در چندین مرحله گره‌ها هرس می‌شوند، در هر مرحله گره ای که هرسش باعث حداکثر افزایش دقت درخت در دسته بندی دسته‌ی تایید می‌شود هرس می‌شود. هرس کردن آنقدر ادامه خواهد یافت تا زمانی که هرس کردن درخت اثر منفی داشته باشد (دقت دسته بندی دسته‌ی تایید را کم کند).

اثر reduce-error pruning بر روی دقت درخت تصمیم گیری در شکل ۳.۷ نشان داده شده است. مشابه شکل ۳.۶ دقت درخت برای نمونه های آموزشی و غیر آموزشی در هنگام هرس نشان داده شده است. منحنی اضافه شده دقت را بر روی دسته‌ی تست در درخت هرس شده نشان می‌دهد. زمانی که هرس کردن آغاز می‌شود درخت در حداکثر اندازه‌ی خودش است. با ادامه‌ی هرس کردن تعداد گره های درخت کاهش و دقت بر روی دسته‌ی تست افزایش می‌یابد. در اینجا داده های موجود به سه دسته تقسیم شده‌اند: نمونه های آموزشی، نمونه های دسته‌ی تایید، و دسته‌ی تست. از دسته‌ی آخر برای بررسی دقت درخت بر روی نمونه های جدید (قدرت تعمیم درخت) استفاده می‌شود. نمودار نشان داده شده دقت را بر روی نمونه های آموزشی و دسته‌ی تست نشان می‌دهد. دقت دسته‌ی ارزیابی که برای هرس کردن از آن استفاده می‌شود در شکل نشان داده نشده است.



شکل ۳.۷ اثر reduced-error pruning در درخت تصمیم گیری.

شکل همان منحنی‌های دقت نمونه های آموزشی و دسته‌ی تست را نشان می‌دهد (شکل ۳.۶). علاوه بر این، اثر reduced-error pruning بر درخت خروجی ID3 در شکل نشان داده شده است. توجه داشته باشید که دقت دسته‌ی تست با هرس شدن گره‌ها افزایش می‌یابد. در اینجا، دسته‌ی تایید که برای هرس استفاده شده از هر دو دسته‌ی آموزشی و تست مجزا بوده است.

زمانی که تعداد زیادی از داده‌ها در دسترس است، استفاده از دسته ای از آن‌ها برای کنترل هرس راه حل موثری است. مانع اصلی این روش این است محدودیت تعداد داده‌هاست. گاهی اوقات کم کردن قسمتی از داده‌ها برای استفاده در دسته‌ی تایید باعث کافی نبودن تعداد داده های موجود برای آموزش درخت می‌شود. در قسمت بعدی روش دیگری را برای هرس توضیح خواهیم داد که در کاربرد های عملی زمانی که تعداد داده‌ها کم است موفقیت آمیز بوده است. تکنیک‌های دیگری نیز از جمله بخش بندی داده‌ها در دسته های متعدد با ترکیب‌های مختلف در دفعات متعدد و میانگین گیری در میان درخت‌ها، ارائه شده است. بررسی‌های تجربی دیگر متد های هرس در (Migers 1989b) و (Malerba 1995) آمده است.

۳.۷.۱.۲ قانون پس هرس

در واقع، یکی از متد های موفق پیدا کردن فرضیه ای با دقت بالا، تکنیکی به نام پس هرس[1] است. نسخه ای از این متد هرس کردن در C4.5 (Quinlar 1993) استفاده شده است. قانون پس هرس مراحل زیر را شامل می‌شود:

۱. درخت متناسب با داده های آموزشی را پیدا کن، به درخت اجازه بده تا اندازه‌ی دلخواه رشد کند و overfit ایجاد شود.
۲. درخت را به دسته قوانین هم ارز تبدیل کن (برای هر مسیر از ریشه به برگ یک قانون).
۳. هر قانون را با حذف کردن شروطی که باعث افزایش دقت تخمینی‌اش می‌شود هرس کن.
۴. قوانین هرس شده را به ترتیب دقتشان مرتب کن، و در دسته بندی نمونه های جدید این سری را در نظر بگیر.

[1] post pruning

برای تصور، دوباره درخت تصمیم گیری شکل ۳.۱ را در نظر بگیرید. در قانون پس هرس، برای هر برگ در درخت یک قانون ایجاد می‌شود. تمامی گره‌هایی که بین ریشه و برگ قرار دارند جزو شروط قانون قرار می‌گیرند و دسته بندی برگ نیز، حکم قانون خواهد بود. برای مثال، برای چپ‌ترین مسیر درخت شکل ۳.۱ قانون زیر بدست می‌آید:

IF (Outlook = Sunny) ∧ (Humidity=High) THEN PlayTennis=No

مرحله‌ی بعدی حذف شروطی که حذفشان دقت تخمینی را کم‌تر نمی‌کند است. برای مثال، برای قانون بالا، قانون پس هرس حذف شروط (Outlook=Sunny)و (Humidity=High) را در نظر خواهد گرفت، و هر کدام از حذف‌ها که پیشرفت بهتری در دقت تخمینی قانون ایجاد کند را انجام می‌دهد و هرس شرط بعدی را به مرحله‌ی بعد موکول خواهد کرد. شرط اصلی هرس کردن این است که بعد از هرس دقت تخمینی کاهش نیابد.

همان طور که در بالا نیز گفته شد، یکی از راه‌های اندازه گیری دقت قوانین استفاده از دسته‌ی تایید است. متد دیگری که در C4.5 نیز آمده تخمین دقت قوانین بر اساس خود دسته‌ی آموزشی است، این تخمین با در نظر گرفتن تمایل نمونه های آموزشی به سمت قوانین موجود با بدبینی به نمونه های آموزشی انجام می‌شود. دقیق‌تر اینکه، C4.5 تخمین بدبین خود را با محاسبه‌ی دقت قوانین بر روی نمونه های آموزشی انجام می‌دهد سپس انحراف معیار[1] این دقت تخمینی را با فرض توزیع دو جمله ای محاسبه می‌کند. برای اطمینان، حد پایین تخمین را به عنوان دقت قانون در نظر می‌گیرد (برای مثال برای فاصله‌ی اطمینان 95% ی دقت قانون با نگاه بدبینانه همان دقت بر روی نمونه های آموزشی منهای ۱.۹۶ برابر انحراف از معیار خواهد بود). در کل، برای مجموعه های این تخمین بدبینانه بسیار نزدیک به دقت مشاهده شده خواهد بود (یعنی مقدار انحراف معیار بسیار کوچک است)، در حالی که با کاهش اندازه دسته داده این مقدار از دقت مشاهده شده کمتر می‌گردد. با وجود اینکه این روش توجیهی آماری ندارد، اما در عمل کاربرد خود را اثبات کرده است. برای بازه های اطمینان و تخمین میانگین به فصل ۵ مراجعه کنید.

چرا درخت تصمیم گیری را قبل از هرس به قوانین تبدیل کنیم؟ این کار سه مزیت دارد:

- تبدیل به قوانین باعث می‌شود که تأثیرات مختلف یک گره در درخت مشخص و جدا گردد، زیرا که هر مسیر از ریشه تا برگ یک قانون را تشکیل می‌دهد و هرس گره های تصمیم اثرات مختلفی بر مسیر های مختلف می‌گذارد. بعلاوه، اگر خود درخت را هرس کنیم دو انتخاب بیشتر نداریم، یکی اینکه گره را حذف کنیم و دیگری اینکه گره را دست نخورده باقی بگذاریم.
- تبدیل به قوانین تمایز بین ویژگی‌های که در نزدیک ریشه بررسی می‌شوند و ویژگی‌هایی که نزدیک برگ‌ها بررسی می‌شوند را از بین می‌برد. بنابراین با این کار مشکلات ساختاری مواجه نخواهیم شد، مشکلاتی نظیر چگونگی بازسازی دوباره درخت در صورت هرس شدن ریشه.
- تبدیل به قوانین درخت را برای خواندن راحت‌تر می‌کند. قوانین معمولاً راحت‌تر درک می‌شوند.

۳.۷.۲ کار با ویژگی‌های پیوسته

تعریف اولیه‌ی ما از ID3 منحصر به ویژگی‌های گسسته مقدار بود. هم خود ویژگی هدف و هم ویژگی‌هایی که در گره‌ها بررسی می‌شدند گسسته بودند. شرط گسسته بودن ویژگی‌هایی گره‌ها را می‌توان به راحتی با تغییرات کوچکی بر طرف کرد. ویژگی‌های پیوسته را می‌توان با

[1] Standard deviation

تعریف پویای هم ارز گسسته‌ی متغیر های پیوسته با بازه بندی به گسسته تبدیل کرد. در کل، ویژگی پیوسته‌ی A را می‌توان با ویژگی منطقی A_c جایگزین کرد. این ویژگی زمانی که A<c ، درست و در غیر این صورت غلت است. حال این سؤال پیش می‌آید که بهترین روش تعیین مقدار آستانه‌ی c چیست؟

برای مثال، فرض کنید که قصد داریم ویژگی پیوسته‌ی Tempereture را در نمونه های آموزشی کار PlayTennis در جدول ۳.۲ اضافه کنیم. فرض کنید که برای گره خاصی از درخت نمونه‌هایی با Tempereture و ویژگی هدف PlayTennis زیر را داریم.

Tempereture	۴۰	۴۸	۶۰	۷۲	۸۰	۹۰
PlayTennis	No	No	Yes	Yes	Yes	No

چه مقدار آستانه ای را باید برای Tempereture در نظر گرفت؟ مسلماً ما تمایل داریم مقدار آستانه ای را انتخاب کنیم که بیشترین بهره‌ی اطلاعات را داشته باشد. با ترتیب کردن نمونه‌ها بر اساس ویژگی پیوسته‌ی A و پیدا کردن مقادیر نزدیک به تغییر دسته بندی تابع هدف، می‌توان مقدار آستانه های پیشنهادی بدست آورد. می‌توان نشان داد که مقدار c در نقطه ای است که بهره‌ی اطلاعات این مرز ماکزیمم است (Fayyad 1991). با بررسی بهره‌ی اطلاعات برای این مقادیر پیشنهادی مقدار آستانه، می‌توان به مقدار c لازم پی برد. در مثال حاضر دو مقدار پیشنهادی برای مقدار آستانه‌ی Temperature وجود دارد (در دو نقطه ای که ویژگی هدف و حوالی‌شان تغییر می‌کند): (48+60)/2 و (80+90)/2. می‌توان برای هر کدام از ویژگی‌های نظیر این مقادیر پیشنهادی ($Temperature_{>54}$ و $Temperature_{>85}$) بهره‌ی اطلاعات را محاسبه کرد و ویژگی‌ای که بهره‌ی بیشتر را دارد بر گزید ($Temperature_{>54}$). این ویژگی منطقی پویای[1] ایجاد شده را می‌توان در کنار دیگر ویژگی‌ها در یادگیری درختی به کار برد. (Fayyad and Irani 1993) اضافاتی به این روش اضافه کردند، آن‌ها به جای مقدار آستانه از بازه‌هایی استفاده کردند. (Ufgoff and Brodley 1991) و (Murthy 1994) نیز در مورد روش‌های تعریف ویژگی‌ها با ترکیبات خطی چندین ویژگی پیوسته و آستانه بندی آن‌ها بحث کرده‌اند.

۳.۷.۳ معیار های دیگر برای انتخاب ویژگی‌ها

در تابع بهره‌ی اطلاعات بایاسی ذاتی وجود دارد که ویژگی‌هایی که تعداد بیشتری مقدار می‌پذیرند را به دیگر ویژگی‌ها ترجیح می‌دهد. برای مثال، ویژگی تاریخ را در نظر بگیرید که تعداد بسیار زیادی مقدار می‌تواند داشته باشد (مثلاً ۴ مارس ۱۹۷۹). اگر این ویژگی را به ویژگی‌های جدول ۳.۲ اضافه کنیم بهره‌ی اطلاعات این ویژگی از همه‌ی ویژگی‌ها بیشتر خواهد بود، زیرا که تاریخ هر روز به تنهایی می‌تواند ویژگی هدف را با استفاده از نمونه های آموزشی مشخص کند. بنابراین، ویژگی تاریخ به عنوان ویژگی ریشه انتخاب خواهد شد و الگوریتم به درختی با عمق یک خواهد رسید که تمامی نمونه های آموزشی را درست دسته بندی می‌کند. البته این درخت تصمیم گیری بر روی نمونه های جدید خیلی ضعیف عمل خواهد کرد، زیرا که با وجود اینکه تمامی نمونه های آموزشی را درست دسته بندی می‌کند اما قدرت پیش بینی بسیار ضعیفی دارد.

مشکل ویژگی تاریخ چیست؟ بیایید ساده نگاه کنیم، مقادیر این ویژگی بسیار زیاد است و تمامی نمونه‌ها را به دسته های بسیار کوچکی تقسیم می‌کند. به همین خاطر، بهره‌ی اطلاعات نسبی بسیار زیادی بر روی نمونه های آموزشی خواهد داشت در حالی که پیش بینی‌های بسیار ضعیفی دارد.

[1] dynamic

یکی از روش‌های حل این مشکل، انتخاب ویژگی‌ها با معیاری دیگر (به غیر بهره‌ی اطلاعات) است. یکی از جایگزین‌های موفق نسبت بهره[1] است (Quinlar 1986). معیار نسبت بهره ویژگی‌هایی چون تاریخ را با جمله ای به نام تقسیم اطلاعات[2] جریمه می‌کند. این جمله به حجم و یکنواختی پخش نمونه‌ها حساس است:

$$SplitInformation(S,A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \qquad (3.5)$$

در این رابطه S_1 تا S_c، c زیر مجموعه‌ی نمونه‌هایی هستند که از تقسیم S با استفاده از ویژگی c مقداری A ایجاد می‌شوند. توجه داشته باشید که SplitInformation در حقیقت همان آنتروپی S با توجه به مقادیر ویژگی A است. این تعریف با تعریف قبلی ما که فقط از آنتروپی برای تعیین ویژگی‌ای که بررسی می‌شود استفاده می‌کردیم کمی تفاوت دارد.

GainRatio یا همان نسبت بهره بر اساس بهره‌ی اطلاعات و تقسیم اطلاعات به شکل زیر تعریف می‌شود:

$$GainRatio(S,A) \equiv \frac{Gain(S,A)}{SplitInformation(S,A)} \qquad (3.6)$$

توجه داشته باشید که جمله‌ی SplitInformation احتمال انتخاب ویژگی‌هایی که تعداد زیادی مقدار دارند را کاهش می‌دهد. برای مثال، مجموعه ای از n نمونه که کاملاً با ویژگی A تقسیم می‌شوند را در نظر بگیرید (مثل تاریخ در مثال قبلی). در این حالت مقدار SplitInformation، $\log_2 n$ خواهد بود. در مقابل ویژگی منطقی B که همان n نمونه را به دو دسته‌ی مساوی تقسیم می‌کند مقدار SplitInformation ۱ خواهد داشت. اگر دو ویژگی A و B بهره‌ی اطلاعات مساوی داشته باشند مطمئناً B نسبت بهره‌ی بیشتری خواهد داشت.

یکی از مشکلات استفاده‌ی GainRatio به جای Gain این است که زمانی که برای S_i ی داشته باشیم $|S_i| \approx |S|$ مخرج بسیار کوچک یا حتی صفر خواهد شد. در هر صورت GainRatio یا بسیار بزرگ می‌شود یا تعریف نشده می‌گردد در حالی که این ویژگی تقریباً برای همه‌ی نمونه های S یکی است. برای پرهیز از این مشکل می‌توان ابتدا معیار Gain را محاسبه کرد و سپس برای ویژگی‌هایی که این معیار از میانگین بزرگ‌تر است GainRatio را محاسبه کرد (Quinlar 1986).

می‌توان برای حل مشکل مذکور به جای GainRatio از معیاری دیگری که بر اساس فاصله است و توسط (Lopez de Mantaras 1991) ارائه شده استفاده کرد. این معیار بر اساس تعریف فاصله‌ی متریک قسمت‌های داده عمل می‌کند. هر ویژگی بر اساس قسمت بندی دادهایی که انجام می‌دهد و قسمت بندی بهینه[3] (قسمت بندی‌ای که تمامی نمونه‌ها را درست دسته بندی می‌کند) سنجیده می‌شود و ویژگی‌ای که تشابه بسیاری به قسمت بندی بهینه دارد انتخاب خواهد شد. (Lopez de Mantaras 1991) این معیار فاصله را تعریف و اثبات می‌کند که این معیار به سمت ویژگی‌هایی که مقادیر بسیاری دارند بایاس ندارد. وی تحقیقاتی را که نشان می‌دهد درخت‌های تولیدی بر اساس

[1] gain ratio
[2] split information
[3] perfect partition

این معیار با درخت‌هایی که بر اساس Gain و GainRatio ساخته می‌شوند تفاوتی ندارند ارائه می‌کند. با این وجود این معیار مشکلات کاربردی معیار GainRatio را ندارد و در تحقیقات وی این معیار درخت‌های بسیار کوچک‌تری برای ویژگی‌هایی با مقادیر بسیار ایجاد می‌کند.

معیار های متنوع دیگری نیز برای این مسئله ارائه شده‌اند (برای مثال، Breiman et al. 1984; Mingers 1989a; Kearns and Mansour 1996; Dietterich 1996). Mingers (1989a) تحلیلی تحقیقی از تأثیر نسبی چندین معیار مختلف بر روی مسائل متنوع انجام داده‌هاست. وی اختلاف قابل توجهی را در اندازه‌ی درخت‌های هرس نشده ناشی از معیار های مختلف را گزارش می‌کند. با این وجود در زمینه های تحقیقات وی به نظر می‌رسد معیار انتخاب ویژگی‌ها تأثیر کمتری بر دقت نهایی تعمیم نسبت به متد پس هرس دارد.

۳.۷.٤ کار با نمونه های آموزشی‌ای که ویژگی‌های مجهول دارند

در بعضی موارد، در داده های موجود تمامی ویژگی‌ها معلوم نیست. برای مثال، در تشخیص بیماری‌ای که بیماری بر اساس دسته ای از آزمایشات آزمایشگاهی تشخیص داده می‌شود ممکن است "جواب آزمایش خون" (blood-Test-Result) برای دسته‌ی معدودی از بیماران در دسترس باشد. در چنین شرایطی، متداول است که این ویژگی‌های مجهول با دیگر ویژگی‌های نمونه و بر اساس دیگر نمونه‌ها تخمین زده می‌شوند.

وضعیتی را در نظر بگیرید که Gain(S,A) برای یک گره n در درخت تصمیم گیری محاسبه می‌شود تا تخمین زده شود که آیا برای این گره A بهترین ویژگی است یا خیر. فرض کنید که <x,c(x)> یکی از نمونه های آموزشی مجموعه‌ی S باشد که در آن ویژگی A(x) مجهول باشد.

یکی از روش‌های برخورد با این ویژگی مجهول این است که متداول‌ترین مقدار ویژگی نمونه‌هایی که به گره n می‌رسند را به آن اختصاص دهیم. یا ممکن است متداول‌ترین مقدار ویژگی را بین نمونه‌هایی که به گره n می‌رسند و مقدار تابع هدف c(x) را دارند به آن اختصاص دهیم. بعد از اختصاص مقدار به این ویژگی می‌توان از نمونه‌ها برای درخت تصمیم گیری موجود استفاده کرد. این استراتژی مفصلاً در (Migers 1989a) توضیح داده شده است.

راه حل دومی نیز وجود دارد، می‌توان از فرایند پیچیده تری (نسبت به اختصاص متداول‌ترین مقدار) برای اختصاص احتمال به هر کدام از مقادیر ممکن استفاده کرد. این احتمال‌ها را می‌توان بر اساس تعداد دفعات تکرار مقادیر مختلف A در میان نمونه های گره n مشخص کرد. برای مثال، اگر ویژگی A ویژگی‌ای منطقی باشد و گره n نیز ۶ نمونه با مقدار A=1 و ۴ نمونه با مقدار A=0 داشته باشد، آنگاه احتمال اینکه A(x)=1 را ۰.۶ و احتمال اینکه A(x)=0 باشد را ۰.۴ در نظر می‌گیریم. با این تقسیم بندی ۰.۶ نمونه‌هایی که های ویژگی مجهول را دارند از شاخه‌ی A=1 و ۰.۴ آن‌ها از شاخه‌ی A=0 پایین خواهند رفت. این نسبت‌ها برای محاسبه‌ی بهره‌ی اطلاعات به نمونه‌ها اختصاص داده می‌شود و ممکن است این کار در زیر درخت‌های بعدی نیز (اگر ویژگی‌ای معلوم نباشد) دوباره انجام گردد. همچنین می‌توان چنین نسبت‌هایی را بعد از یادگیری برای دسته بندی نمونه های جدیدی که بعضی ویژگی‌ها را ندارند اعمال کرد. در چنین حالتی این دسته بندی نمونه‌ی جدید محتمل‌ترین دسته بندی خواهد بود، این محتمل‌ترین[1] دسته بندی با جمع وزن دار دسته بندی‌های مختلف هر گره برگ درخت انجام خواهد گرفت. از این متد برای کار با نمونه های آموزشی‌ای که ویژگی‌های مجهول دارند در C4.5 مورد استفاده قرار گرفته است (Quinlan 1993).

[1] Most probable

۳.۷.۵ کار با ویژگی‌های غیر هم ارزش

در بعضی از کارهای یادگیری ممکن است نمونه‌ها ویژگی‌های غیر هم هزینه ای داشته باشند. برای مثال، در مثال تشخیص بیماری ممکن است بیماران را با ویژگی‌های درجه‌ی حرارت بدن، آزمایش بافت، نبض، نتیجه‌ی آزمایش خون، و ... توصیف کنیم. این ویژگی‌ها مسلماً قیمت یکسانی ندارند، هم از نظر هزینه پولی آزمایش و هم از نظر هزینه راحتی بیمار. در چنین کارهایی درخت‌هایی را ترجیح می‌دهیم که تا جایی که ممکن باشد آزمایش‌های کم هزینه تر را انجام دهد و آزمایش‌های پر هزینه را به تشخیص‌های آخر موکول کند.

ID3 می‌تواند با اضافه کردن جمله‌ی هزینه به معیار انتخاب ویژگی‌اش این هزینه‌ها را در نظر بگیرد. برای مثال، می‌توان رابطه‌ی Gain را بر هزینه تقسیم کرد تا ویژگی‌هایی که هزینه‌ی کمتر دارند ارجحیت بیشتری داشته باشند. با وجود اینکه این روش‌ها تضمین نمی‌کنند که روش بهینه ای در هزینه پیدا کنند اما بایاسی در جستجو به سمت ویژگی‌های کم هزینه تر ایجاد می‌کنند.

(Tan and Schlimmer 1990) و (Tan 1993) روشی برای این کار ابداع کردند و در کار ادراک یک ربات به کاربردند، در این کار ربات یاد می‌گرفت که چگونه اشیاء مختلف را با حس کردن آن‌ها با بازوهایش دسته بندی کند. در این کار، ویژگی‌ها خروجی‌های حسگر متحرک ربات بودند. هزینه‌ی هر ویژگی با تعداد ثانیه‌هایی که طول می‌کشید تا حسگر در آن موقعیت قرار گیرد و خروجی بدهد اندازه گیری می‌شد. آن‌ها ثابت کردند که با استفاده از معیار زیر می‌توان بدون کاهش دقت دسته بندی می‌توان مؤثرترین استراتژی تشخیص اشیا را یاد گرفت:

$$\frac{Gain^2(S,A)}{Cost(A)}$$

(Nunez 1988) راه حل مشابهی را ارائه می‌دهد و آن را برای یادگیری تشخیص‌های پزشکی به کار می‌برد. در این کاربرد ویژگی‌ها نشانه های بیماری و آزمایش‌های آزمایشگاهی با هزینه های مختلف هستند. در سیستمی که وی ارائه داد معیار دیگری برای انتخاب ویژگی‌ها ارائه شده بود:

$$\frac{2^{Gain(S,A)}-1}{(Cost(A)+1)^w}$$

در این رابطه $w\in[0,1]$ ثابتی است که اهمیت نسبی هزینه در مقابل بهره‌ی اطلاعات را معلوم می‌کند. (Nunez 1991) به روش تجربی این دو روش را در مسایل مختلف مقایسه کرد.

۳.۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل:

- یادگیری درختی متدی کاربردی در یادگیری مفهوم و توابع گسسته مقدار است. خانواده‌ی الگوریتم‌های مشابه ID3 شامل الگوریتم‌هایی می‌شود که درخت را از ریشه به سمت پایین حریصانه رشد می‌دهند، در هر مرحله از رشد درخت برای هر شاخه تصمیم جدید بهترین ویژگی انتخاب می‌شود.

- ID3 فضای فرضیه ای کاملی (فضای فرضیه ای تمامی درخت‌های تصمیم ممکن برای توابع گسسته مقدار) را جستجو می‌کند. به همین دلیل این الگوریتم مشکل اساسی که هنگام جستجوی دسته فضای فرضیه های محدود به وجود می‌آید، این احتمال که ممکن است تابع هدف در فضای فرضیه ای مفروض نباشد، را ندارد.
- بایاس استقرایی ID3 درخت‌های کوچک‌تر را ارجح می‌داند؛ به این معنا که درخت را فقط تا زمانی که نمونه های آموزشی را دسته بندی کند رشد می‌دهد.
- مسئله‌ی overfit بر روی نمونه های آموزشی مسئله ای مهم در یادگیری درختی است. زیرا که نمونه های آموزشی فقط نمونه ای از تمامی نمونه های ممکن هستند، ممکن است ما به درخت شاخه‌هایی را بیفزاییم که کارایی درخت را بر روی نمونه های آموزشی افزایش داده اما کارایی برای نمونه های خارج این مجموعه کاهش یابد. به همین دلیل متد های هرس درخت تصمیم برای پرهیز از overfit در یادگیری درختی (و دیگر الگوریتم‌های یادگیری‌ای که از بایاس ترجیحی استفاده می‌کنند) از اهمیت خاصی برخوردارند.
- انواع بسیاری از تغییرات برای ID3 توسط محققان ایجاد شده است. این تغییرات شامل متد های پس هرس درخت‌ها، کار با ویژگی‌های حقیقی مقدار، کار با نمونه های آموزشی‌ای که ویژگی‌های مجهول دارند، بازنگری در درخت با افزایش تعداد نمونه های آموزشی، استفاده از معیار های دیگری به جای معیار gain برای انتخاب ویژگی‌ها و در نظر گرفتن هزینه اندازه گیری ویژگی‌های نمونه‌هاست.

در میان اولین کارهایی که بر روی درخت تصمیم انجام گرفته، (Hunt's Concept Learning System) یا CLS (Hunt et al. 1966) و کار (Friedman and Breiman) روی سیستم CART (Friedman 1977; Breiman et al. 1984) جزو مهم‌ترین‌ها هستند. سیستم ID3 (Quinlan 1979, 1983) نیز پایه بحث این فصل را تشکیل می‌دهد. دیگر کارهای اولیه روی یادگیری درختی شامل ASSISTANT (Kononenko et al. 1984; Cestnik et al. 1987) می‌شود. پیاده سازی الگوریتم‌های استقرایی درختی هم اکنون به صورت تجاری روی بسیاری از سیستم عامل‌ها ارائه می‌شود.

برای مطالعه‌ی بیشتر روی استقرای یادگیری درختی، کتاب (Quinlan 1993) بسیاری از مسائل عملی را بررسی کرده و کد های قابل اجرایی برای C4.5 را در بر دارد.(Mingers 1989a) و (Buntine and Niblett 1992) دو بررسی بر روی اختلاف بین روش‌های مختلف انتخاب ویژگی را بررسی می‌کنند. بررسی‌هایی که یادگیری درختی و دیگر متدهای یادگیری را مقایسه می‌کنند را می‌توان در بسیاری از مقالات شامل (Dietterich et al. 1995; Fisher and McKusick 1989; Quinlan 1988a; Shavlik et al. 1991; Thrun et al. 1991; Weiss and Kapouleas 1989) پیدا کرد.

تمرینات

۳.۱ درخت‌های تصمیمی که توابع منطقی زیر را بیان می‌کند بیابید:

(a) $A \vee \neg B$

(b) $A \vee [B \wedge C]$

(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$

۳.۲ مجموعه‌ی نمونه های آموزشی زیر را در نظر بگیرید:

شماره‌ی نمونه‌ی آموزشی	دسته بندی	a_1	a_2
۱	+	T	T
۲	+	T	T
۳	−	T	F
۴	+	F	F
۵	−	F	T
۶	−	F	T

(a) آنتروپی این مجموعه از نمونه های آموزشی با توجه به دسته بندی تابع هدف چقدر است؟

(b) بهره‌ی اطلاعات ویژگی a_2 برای این نمونه های آموزشی چقدر است؟

۳.۳ عبارت زیر غلت یا درست است؟

اگر درخت تصمیم D2 یک خاص سازی از D1 باشد، آنگاه D1 کلی‌تر است از D2. فرض کنید که D1 و D2 درخت‌های تصمیم متغیر منطقی دلخواهی هستند و D2 یک خاص سازی از D1 است اگر ID3 بتواند D1 را به D2 تامیم دهد. اگر جمله بالا درست است آن را اثبات کرده در غیر این صورت مثال نقض بیاورید. (مفهوم کلی‌تر بودن در فصل ۲ تعریف شده است)

۳.۴ ID3 جستجویی برای یافتن تنها یک فرضیه‌ی سازگار انجام می‌دهد در حالی که Candidate-Elimination تمامی فرضیه های سازگار را پیدا می‌کند. رابطه ای بین این دو الگوریتم یادگیری در نظر بگیرید.

(a) درخت تصمیمی که ID3 با نمونه‌های آموزشی EnjoySport یاد می‌گیرد را پیدا کنید. مفهوم هدف در جدول ۲.۱ فصل ۲ آورده شده است.

(b) رابطه‌ی بین درخت تصمیم یادگیری شده و فضای ویژه‌ی نشان داده شده در شکل ۲.۳ در فصل ۲ که از همین نمونه‌های آموزش بدست آمده چیست؟

(c) نمونه‌ی آموزشی زیر را به نمونه‌های آموزشی اضافه کرده و درخت تصمیم جدیدی را یاد بگیرید. این بار بهره‌ی اطلاعات بدست آمده برای هر ویژگی در هر مرحله از رشد درخت را تعیین کنید.

Sky	Air-Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Weak	Warm	Same	No

(d) فرض کنید که می‌خواهیم یادگیری مشابه ID3 طراحی کنیم که فضایی از فرضیه های درخت‌های تصمیم را جستجو کرده و مشابه (Candidate-Elimination) تمامی فرضیه های سازگار با این داده‌ها را پیدا کند. به طور خلاصه این که، می‌خواهیم Candidate-Elimination را برای جستجوی فضای فرضیه ای درخت‌های تصمیم بکار بریم. مجموعه های S و G را که از نمونه های آموزشی جدول ۲.۱ بدست می‌آیند را تعیین کنید. توجه داشته باشید که S باید خاص‌ترین درخت‌های ساخته شده با استفاده از داده‌ها و G باید کلی‌ترین درخت‌های ساخته شده را در بر بگیرد. نشان دهید که این دو مجموعه با اعمال نمونه‌ی آموزشی دوم چگونه تغییر می‌کنند (می‌توانید درخت‌هایی را که یک مفهوم را ارائه می‌کنند و فقط ساختار غیر یکسان دارند حذف کنید). چه مشکلاتی را در اعمال الگوریتم Candidate-Elimination به فضای فرضیه ای درخت‌های تصمیم می‌بینید؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

آنتروپی	Entropy
بایاس	Bias
بایاس استقرایی	inductive bias
بهره‌ی اطلاعات	information gain
تقسیم اطلاعات	split information
تئوری اطلاعات	information theory
جستجوی کم عمق	breath first search
حریصانه	Greedy
خاص	Specific
خاص‌ترین کلی سازی‌ها	maximal generalization
درخت تصمیم گیری	decision tree
دسته بندی	Classify
دسته‌ی تایید	validation set
زیر درخت	Subtree
سازگار	Consistent
شامل	Expressive
علامت گذاری	Label
غیر قطعی	Nondeterministic
فرضیه	Hypothesis
فضای ویژه	space version
کلی	General
کلی‌ترین خاص سازی‌ها	minimal specialization
نمونه آموزشی	training example
نمونه غیر آموزشی	unobserved instance
مسائل دسته بندی	classification problems
مفهوم هدف	target concept
نسبت بهره	gain ratio
نقیض	Negation

هرس	post-prune
یادگیری درختی	decision tree learning

Telegram: @Farabar_BI
www.farabar.net

فصل چهارم: شبکه های عصبی مصنوعی

شبکه های عصبی مصنوعی[1] یا همان ANN ها متدی کلی و کاربردی برای یادگیری توابع حقیقی مقدار، گسسته مقدار و برداری از روی نمونه‌هاست. الگوریتم‌هایی یادگیری شبکه ای چون Backpropagation از روش‌هایی چون شیب نزول[2] استفاده کرده تا پارامترهای شبکه را طوری تنظیم کنند تا با دسته نمونه های آموزشی مطابقت داشته باشند. یادگیری شبکه های عصبی در مقابل خطاها در داده های آموزشی مقاوم[3] است و در تفسیر صحنه های تصویری، تشخیص صحبت و یادگیری استراتژی‌های کنترل ربات به کاربرد دارد.

٤.١ معرفی

متد یادگیری شبکه های عصبی روش‌هایی مقاوم به نویز برای تخمین توابع هدف حقیقی مقدار، گسسته مقدار و برداری ارائه می‌کند. در انواع خاصی از مسائل مثل یادگیری تفسیر ورودی‌های پیچیده‌ی حسگرها، شبکه های عصبی بهترین روش شناخته شده هستند. برای مثال، الگوریتم Backpropagation، که در این فصل کاملاً آن‌را توضیح خواهیم داد، موفقیت‌های چشم گیری در حل مسائل کاربردی‌ای نظیر تشخیص کاراکتر های دست نویس (LeCun et al. 1989) ، تشخیص صحبت (Lang et al. 1990) و تشخیص چهره (Cottrell 1990) از خود نشان داده است. بررسی‌ای از کاربرد های واقعی شبکه های عصبی توسط (Rumelhart et al. 1994) گرد آوری شده است.

٤.١.١ انگیزه‌ی زیستی

مطالعه‌ی شبکه های عصبی مصنوعی از سیستم‌های یادگیر زیستی که از شبکه های خیلی پیچیده‌ی اعصاب ساخته شده‌اند الهام گرفته شده است. در نگاه سطحی، این سیستم‌ها از انبوهی از دسته واحد های متصل به هم ساده ساخته شده‌اند که هر واحد ورودی‌های حقیقی مقداری

[1] Artificial neural networks
[2] Gradient decend
[3] robust

www.farabar.net

دریافت کرده (بیشتر این ورودی‌ها خروجی‌های واحد های دیگر هستند) و مقدار حقیقی‌ای را محاسبه می‌کند (که ممکن است ورودی واحد های دیگری باشد).

برای درک بهتر، چند حقیقت از عصب شناسی را با هم ملاحظه می‌کنیم. برای مثال، تخمین زده می‌شود که مغز انسان 10^{11} عصب دارد که هر کدام به طور متوسط به 10^{4} عصب دیگر متصلند. فعالیت عصب به طور عادی در یکی از دو حالت برانگیخته[1] و غیر برانگیخته[2] است. سریع‌ترین اعصاب در مرتبه‌ی 10^{-3} ثانیه بین این دو حالت سوییچ می‌کنند (این مقدار در مقابل کامپیوترها 10^{-10} مرتبه‌ی کندتر است). با این حال انسان می‌تواند به سرعت تصمیمات بسیار پیچیده ای بگیرد. برای مثال، شما تصویر مادرتان را در مدت حدوداً 10^{-1} ثانیه تشخیص می‌دهید. توجه دارید که در مدت این 10^{-1} ثانیه، با توجه به سرعت عملکرد اعصاب، اعصاب حداکثر چند صد بار برانگیخته شده‌اند. مشاهدات نشان داده که قدرت پردازش اطلاعات در سیستم‌های عصبی زیستی ناشی از عملیات‌های موازی بسیاری است که بر روی تعداد زیادی از اعصاب اجرا می‌شوند. یکی از انگیزه های به کار گیری شبکه عصبی رسیدن به چنین محاسبات موازی‌ای که توسط تعدادی زیادی واحد انجام می‌شود است. با وجود اینکه الگوریتم‌های سریع‌تری بر روی ماشین‌های محاسبه‌ی موازی استفاده شده و همچنین سخت افزار های خاصی برای برنامه های شبکه عصبی طراحی شده، اما اکثر برنامه های شبکه عصبی بر روی ماشین‌هایی ترتیبی[3] اجرا می‌شوند که عمل محاسبه‌ی غیر متمرکز را شبیه سازی می‌کنند.

با وجود اینکه شبکه های عصبی برداشتی از شبکه های عصبی زیستی است، اما بسیاری از پیچیدگی‌های شبکه های عصبی زیستی در شبکه های عصبی مدل سازی نمی‌شود، همان طور که بسیاری از خواص شبکه های عصبی (که درباره‌ی آن‌ها بحث خواهیم کرد) با سیستم‌های زیستی مطابقت ندارد. برای مثال، ما فرض می‌کنیم که واحد های شبکه عصبی یک سیگنال خروجی دارند، درحالی‌که در اعصاب زیستی خروجی سری‌ای ترکیبی از ضربه‌ها در طول زمان است.

بر اساس تاریخچه، دو دسته از محققان بر روی شبکه های عصبی مصنوعی کار می‌کردند. گروه اول که سعی داشتند با تقلید شبکه های عصبی فرایند های یادگیری زیستی را مطالعه و مدل سازی کنند و گروه دوم کسانی که سعی داشتند به الگوریتم‌های یادگیری ماشین موثری دست یابند، جدا از اینکه این الگوریتم‌ها از شبکه های عصبی بدست آمده است. در طول این کتاب، ما نیز جزو گروه دوم محسوب می‌شویم و بنابراین بر مدل سازی زیستی تاکید نمی‌کنیم. برای اطلاعات بیشتر در مورد مدل سازی سیستم‌های زیستی توسط شبکه های عصبی می‌توانید به کتب زیر مراجعه کنید:

Churchland and Sejnowski (1992);

Zornetzer et al. (1994);

Gabriel and Moore (1990).

---

[1] excited
[2] inhibited
[3] sequential

٤.٢ معرفی شبکه های عصبی

یک نمونه‌ی تمام عیار از یادگیری شبکه عصبی توسط سیستم Pomerleau (1993)، به نام ALVINN شبیه سازی شده است. این سیستم برای کنترل فرمان اتومبیل با سرعت متوسط در بزرگراه‌ها طراحی شده است. ورودی این شبکه‌ی عصبی یک تصویر 30x32 نقطه ای است که از دوربین رو به جلویی که در داخل اتومبیل کار گذاشته شده گرفته می‌شود. خروجی شبکه‌ی عصبی جهتی است که نشان به آن سمت باید بچرخد. شبکه برای تقلید فرمان دهی انسان در طول حدود ۵ دقیقاً آموزش داده می‌شود. ALVINN موفق شده تا با شبکه‌ی آموزش دیده‌ی خود، خودرو را تا سرعت ۷۰ مایل در ساعت و برای مسافت ۹۰ مایل در بزرگراه کنترل کند (رانندگی‌ای که در خط سرعت بزرگراه بوده و بزرگراه نیز خط کشی شده بوده و دیگر وسایل نقلیه هم در بزرگراه حضور داشته‌اند).

شکل ۴.۱ شبکه‌ی عصبی استفاده شده در یکی از نسخه های ALVINN و نحوه‌ی نمایش بسیاری از شبکه های عصبی را نشان می‌دهد. شبکه با نمونه ای از تصویر های دوربین در چپ تصویر نشان داده شده است. هر واحد[1] (در شکل دایره) در شبکه نشان دهنده‌ی خروجی یک واحد است و خطوط متصل به زیر هر واحد نیز ورودی‌های آن هستند. همان طور که در شکل نیز نشان داده شده ۴ واحد مستقیماً به تمامی نقاط تصویر متصلند. این چهار واحد پنهان[2] نامیده می‌شوند زیرا که بر خروجی به طور غیر مستقیم اثر می‌کنند و هیچ گاه به صورت مستقیم تأثیری ندارند. هر یک از این چهار واحد خروجی‌ای بر اساس ۹۶۰ ورودی وزن دار خود ایجاد می‌کنند. خروجی این ۴ واحد پنهان به عنوان ورودی به ۳۰ واحد خروجی[3] داده می‌شود. هر واحد خروجی متناسب با یکی از فرمان‌های جهتی اتومبیل است (مثل کمی به راست، کمی به چپ، کاملاً به چپ ، کاملاً به راست یا مستقیم) و این خروجی‌ها نشان می‌دهد که کدام جهت برای فرمان ارجحیت دارد.

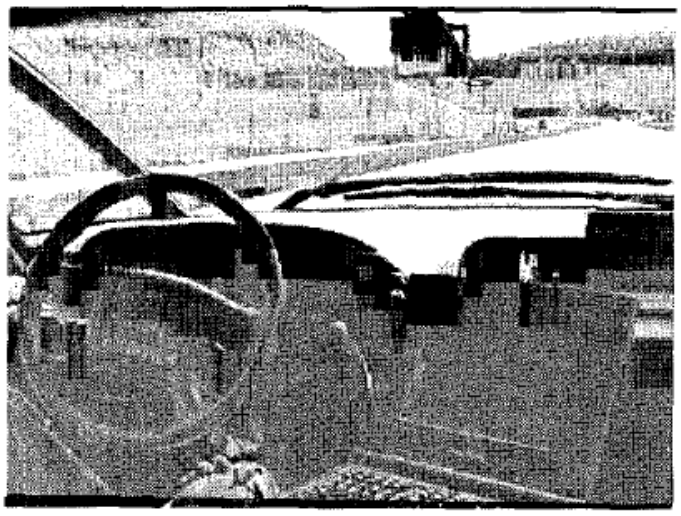
سمت راست شکل وزن‌های[4] یاد گرفته شده‌ی متناسب با یکی از این چهار واحد را نمایش می‌دهد. ماتریکس سیاه و سفیدی که در سمت راست و پایین شکل نشان داده شده وزن‌های متناسب با نقطه‌ها در یکی از ۴ واحد پنهان است. در این شکل مربع‌های سیاه نشان دهنده‌ی وزن منفی و مربع‌های سفید نشان دهنده‌ی وزن مثبتند و اندازه‌ی مربع نیز بزرگی وزن را نشان می‌دهد. مستطیل بالای مربع وزن‌های ورودی از هر ۴ واحد پنهان به ۳۰ خروجی را نشان می‌دهد.

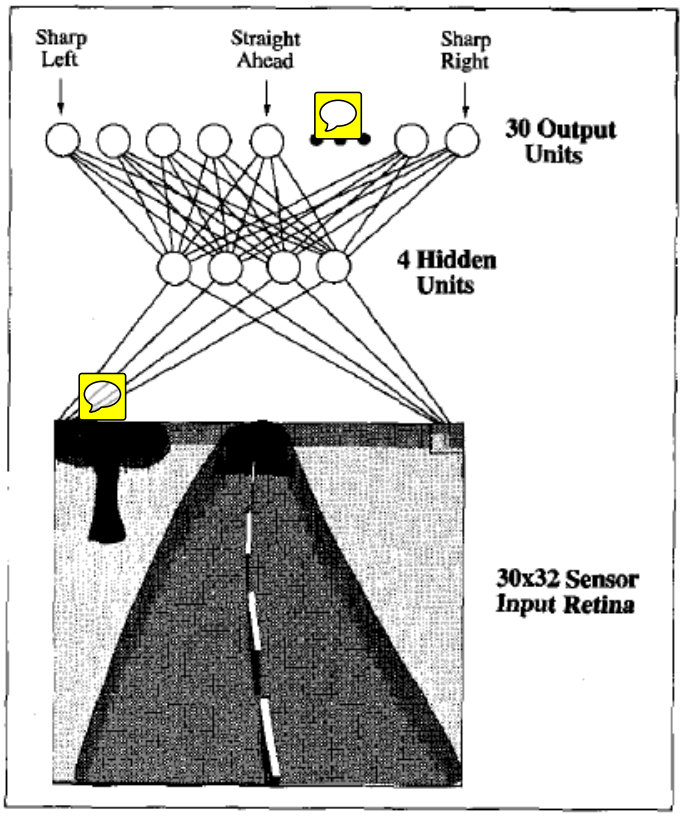
ساختار شبکه‌ی ALVINN در بسیاری از شبکه های عصبی به کار برده می‌شود. در این چنین شبکه‌هایی فقط ارتباط‌های بین لایه ای وجود دارد و گراف جهت دار نظیر دور ندارد[5]. در کل، ساختار شبکه‌ها ممکن است هر نوع گرافی باشند (اعم از دور دار[6] و بدون دور[7]، جهت دار[8] و یا بدون جهت[9]) . در این فصل به بررسی پرکاربردترین و عمومی‌ترین ویژگی‌های شبکه های عصبی که بر پایه‌ی الگوریتم Backpropagation است می‌پردازیم. الگوریتم Backpropagation فرض می‌کند که شبکه ساختاری ثابت و متناسب با یک گراف

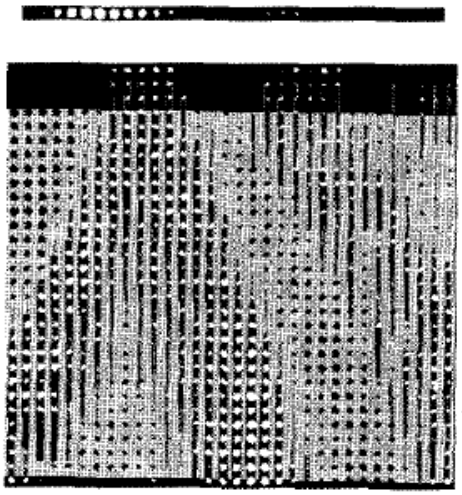
---

[1] node
[2] hidden
[3] output
[4] Weight value
[5] Directed acyclic graph
[6] cyclic
[7] acyclic
[8] directed
[9] adirected

جهت دار دارد که ممکن است دور نیز داشته باشد. و سعی می‌کند تا مقادیر متناسب با هر یال[1] در این گراف را یاد بگیرد. با وجود اینکه حلقه در شبکه مجاز است اما اکثر شبکه های کاربردی بدون حلقه و به فرم تک سویه[2] هستند، درست مثل ساختار شبکه‌ی ALVINN.







شکل ۴.۱ شبکه‌ی عصبی‌ای که برای کنترل فرمان خودرو طراحی شده.

[1] edge
[2] feed-forward

سیستم ALVINN از Backpropagation با هدف یادگیری کنترل خودرو استفاده می‌کند (شکل بالا یی). شکل سمت چپ نشان می‌دهد که چگونه ۹۶۰ نقطه‌ی تصویر دوربین به ۴ واحد پنهان متصل به ۳۰ واحد خروجی متصل شده‌اند. خروجی شبکه فرمان‌های کنترل فرمان خواهد بود. شکل سمت راست وزن‌های نظیر یکی از واحد های پنهان را نشان می‌دهد. وزن‌ها در یک ماتریس 30x32 نشان داده شده‌اند. در این ماتریس وزن‌های مثبت سفید و وزن‌های منفی سیاه رنگند، اندازه‌ی هر وزن متناسب با اندازه‌ی هر مربع است. مستطیل کوچک بالای این ماتریس وزن‌های ۳۰ واحد خروجی متصل به این واحد را مشخص می‌کند. همان طور که در مستطیل نیز معلوم است تهییج این واحد پنهان باعث گردش به چپ می‌گردد.

٤.٣ مسائل متناسب با یادگیری شبکه های عصبی

یادگیری شبکه های عصبی مناسب مسائلی با داده های ورودی نویز دار یا ترکیبی از چندین حسگر[1] مثل دوربین و میکروفن است. همچنین در مسائلی که کاملاً به صورت نمادی[2] بیان می‌شود، مثل مسائلی که در فصل ۳ بررسی شد، کاربرد دارند. در چنین مسائلی شبکه های عصبی و درخت تصمیم گیری دقت قابل مقایسه ای دارند. در Shvlik et al. (1991) و Weiss and Kapouleas (1989) مقایسه های تجربی این دو راهبرد برای مسائل مختلف بررسی شده است. الگوریتم Backpropagation پرکاربردترین متد یادگیری شبکه های عصبی محسوب می‌شود. این الگوریتم برای مسائلی با ویژگی‌های زیر متناسب است:

- نمونه‌ها به صورت n تایی‌های مرتبند. تابع هدف بر روی نمونه‌هایی تعریف شده که توسط بردارهایی از ویژگی‌ها بیان می‌شوند، مثل مقدار نقطه های تصویر در مثال ALVINN. این مقادیر ممکن است کاملاً وابسته[3] و یا کاملاً مجزا[4] باشند. مقادیر ورودی می‌توانند هر مقدار حقیقی‌ای باشند.
- تابع هدف ممکن است گسسته مقدار، حقیقی مقدار، یا برداری ترکیبی از حقیقی مقدار و گسسته مقدار باشد. برای مثال در ALVINN خروجی برداری از ۳۰ ویژگی است. خروجی هر یک از مقادیر حقیقی بین ۰ تا ۱ را می‌تواند داشته باشد، در این مثال این مقدار اطمینان شبکه به پیچیدن به آن جهت را مشخص می‌کند. همچنین می‌توان شبکه ای آموزش داد که علاوه بر کنترل فرمان کنترل سرعت اتومبیل را نیز در اختیار داشته باشد. کافی است مقداری برای کنترل شتاب به خروجی‌ها اضافه کنیم.
- نمونه‌ها ممکن است خطا داشته باشند. متد های یادگیری شبکه عصبی در مقابل داده های آموزشی نویز دار مقاوم است.
- زمان آموزش زیاد قابل قبول است. الگوریتم‌های آموزش شبکه زمانی بیشتر از آموزش‌های دیگر الگوریتم‌ها (مثلاً درخت تصمیم گیری) لازم دارند. زمان آموزش‌ها بسته به تعداد وزن‌های در شبکه ، تعداد نمونه های آموزشی، و تنظیمات پارامتر های الگوریتم یادگیری ممکن است از چند ثانیه تا چندین ساعت تغییر کند.
- ارزیابی سریع تابع هدف لازم باشد. با وجود اینکه شبکه های عصبی به نسبت کند آموزش داده می‌شوند، اما ارزیابی نمونه های جدید توسط شبکه‌ی آموزش دیده بسیار سریع انجام می‌گردد. برای مثال در ALVINN هر ثانیه چندین بار شبکه‌ی عصبی خود را ارزیابی می‌کند و دستورات فرمان دهی را تغییر می‌دهد.
- قدرت انسان برای درک تابع هدف یاد گرفته شده مهم نیست! گاهی تفسیر مقادیر یاد گرفته شده برای وزن‌ها ممکن نیست و قوانین شبکه های عصبی آموزش دیده برای انسان به سادگی قابل درک نیستند.

---

[1] sensor
[2] Symbolic representation
[3] correlated
[4] independent

در ادامه‌ی این فصل: ابتدا انواع طراحی واحد را در شبکه های عصبی بررسی خواهیم کرد (واحد های پرسپترون[1] ، واحد های خطی و واحد های سیگموید[2])، سپس به الگوریتم‌های آموزش تک واحدها می‌پردازیم. پس از ان، الگوریتم Backpropagation را برای آموزش شبکه های چند لایه ساخته شده از چنین واحدهایی بیان خواهیم کرد و به مطالب کلی‌تری نظیر قابلیت‌های شبکه‌ها، مسئله‌ی overfit و جایگزین‌های Backpropagation می‌پردازیم. و در آخر نیز یک مثال توضیحی از استفاده الگوریتم Backpropagation برای آموزش شبکه با هدف تشخیص چهره آورده‌ایم تا خواننده بتواند از این الگوریتم برای آموزش شبکه استفاده‌ی کاربردی کند.

٤.٤ پرسپترون‌ها

نوعی از سیستم‌های شبکه عصبی بر پایه‌ی نوعی واحد به نام پرسپترون ساخته می‌شود (شکل ۴.۲). پرسپترون برداری حقیقی مقدار دریافت کرده و ترکیبی خطی از آن‌را محاسبه می‌کند، اگر این مقدار از مقدار خاصی (مقدار آستانه)[3] بیشتر بود خروجی را ۱ و در غیر این صورت خروجی را 1- می‌دهد. به صورت دقیق‌تر، اگر $x_1, \dots, x_n$ ورودی‌ها باشند و $o(x_1, \dots, x_n)$ خروجی واحد باشد داریم:

$$o(x_1, \dots, x_n) = \begin{cases} 1\,, & w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0 \\ -1\,, & \text{در غیر این صورت} \end{cases}$$

در این تابع هر w_i مقدار حقیقی ثابتی یا همان وزن است که میزان تأثیر x_i را در خروجی پرسپترون تعیین می‌کند. توجه دارید که مقدار w_0 نیز یک مقدار آستانه است و ترکیب $w_1x_1 + w_2x_2 + \cdots + w_nx_n$ باید حداقل مقدار $(-w_0)$ را داشته باشد تا خروجی ۱ شود.

برای ساده تر شدن نمایش فرض می‌کنیم که $x_0 = 1$، و به جای عبارت شرط می‌نویسیم $\sum_{i=0}^{n} w_ix_i > 0$ و یا به صورت برداری داریم که $\vec{w}.\vec{x} > 0$. و برای اختصار می نویسیم

$$o(\vec{x}) = sgn(\vec{w}.\vec{x})$$

در این نمایش تابع sgn همان تابع علامت است:

$$sgn(y) = \begin{cases} 1\,, y > 0 \\ -1\,, & \text{در غیر این صورت} \end{cases}$$

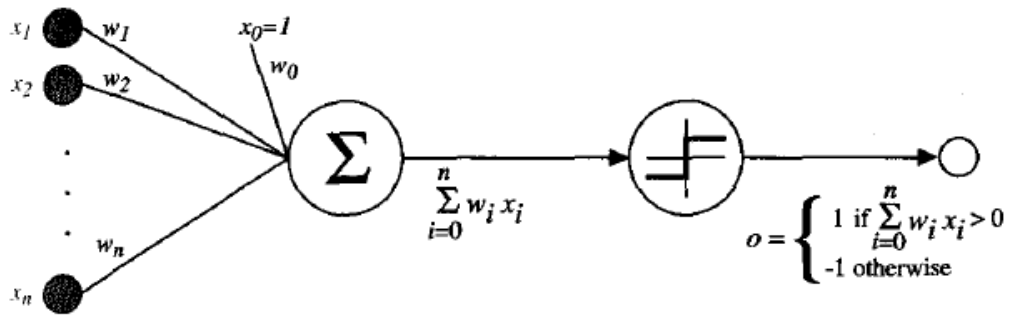
یادگیری برای پرسپترون به معنای پیدا کردن مقدار مناسب برای $w_0, \dots, w_n$ است. پس فضای فرضیه ای متناسب با این یادگیری تمام بردار های حقیقی مقدار خواهند بود:

$$H = \{\vec{w} | \vec{w} \in \Re^{(n+1)}\}$$

[1] perceptron
[2] sigmoid
[3] threshold

۴.۴.۱ قدرت پرسپترون‌ها

ما می‌توانیم پرسپترون را ابر صفحه ای[1] سطح تصمیم در فضای n بعدی نمونه‌ها بدانیم. پرسپترون برای نمونه‌هایی که در یک طرف این ابر صفحه هستند ۱ و برای نمونه‌هایی که در طرف دیگر این ابر صفحه هستند مقدار 1- را برمی گرداند (شکل ۴.۳). معادله‌ی این ابر صفحه‌ی تصمیم گیری به فرم $\vec{w}.\vec{x} = 0$ نوشته می‌شود. البته تمامی دسته نمونه های آموزشی را نمی‌توان بدین شکل دسته بندی کرد. دسته مثال‌هایی را که این گونه دسته بندی می‌شوند دسته بندی پذیر خطی[2] می‌نامند.

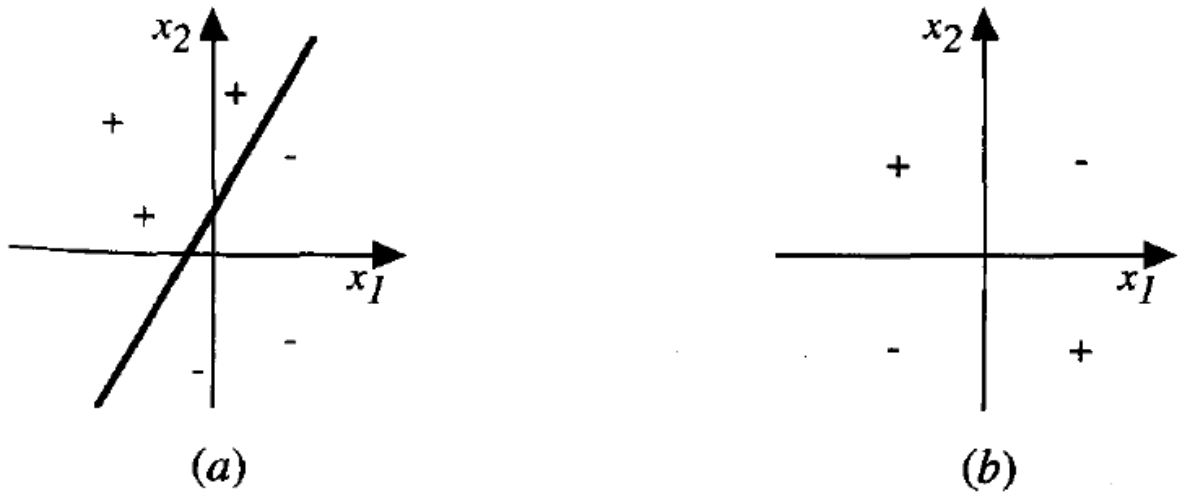


شکل ۴.۲ یک پرسپترون.

پرسپترون‌ها به تنهایی می‌توانند بسیاری از توابع منطقی مقدار را یاد بگیرند. برای مثال اگر مقدار ۱ را درست (True) و مقدار 1- را غلت (False) در نظر بگیریم برای شبیه سازی تابع AND می‌توان وزن‌ها را به صورت $w_1 = w_2 = .5$ و $w_0 = -.8$ را در نظر گرفت. همین پرسپترون را می‌توان با عوض کردن مقدار w_0 به 3.- به تابع OR تبدیل کرد. در واقع توابع AND و OR را می‌توان به صورت توابع خاص m از n دانست: توابعی که زمانی مقدار درست را بر می‌گردانند که حداقل m تا از n ورودیشان درست باشد. در تابع OR، m=1 و در تابع AND، m=n است. چنین توابعی را به سادگی می‌توان با یکی کردن وزن‌ها و تعیین مقدار متناسب w_0 توسط پرسپترون‌ها تقلید کرد.

پرسپترون‌ها تمامی توابع ساده‌ی منطقی اعم از AND، OR، NAND (¬AND) و NOR (¬OR) را تقلید می‌کنند. اما متأسفانه پرسپترون‌ها نمی‌توانند توابعی همچون XOR را تقلید کنند. تابع XOR زمانی درست است که $x_1 \neq x_2$. در شکل ۴.۳ (b) تمام نمونه های آموزشی مربوط به XOR نشان داده شده است.

[1] hyperplane
[2] linearly separable



شکل ۴.۳ فضای مثال‌ها برای پرسپترون‌هایی که دو ورودی دارند.
(a) نمونه‌هایی آموزشی و فضای آن‌ها که پرسپترون آن‌ها را درست دسته بندی می‌کند. (b) دسته مثال‌هایی آموزشی که دسته بندی پذیر خطی نیستند (با هیچ خطی نمی‌توان نمونه های مثبت را از نمونه های منفی جدا کرد). نمونه های مثبت با "+" و نمونه های منفی با "-" در شکل نشان داده شده است.

قدرت پرسپترون‌ها برای یادگیری توابع AND، OR، NAND و NOR از این رو اهمیت دارد که تمامی توابع منطقی توسط این ترکیب توابع شبیه سازی هستند. در واقع تمامی توابع منطقی توسط دو سری از پرسپترون‌های متصل به هم (خروجی سری اول به ورودی سری دوم وصل باشد) قابل شبیه سازی‌اند. یک راه معمول بیان توابع به صورت فصلی از توابع پایه است (برای مثال، فصلی (OR) از عطف‌های (AND) بین ورودی‌ها و نقیضشان). توجه داشته باشید که می‌توان به سادگی تمام ورودی‌ها را با تغییر علامتشان وزنشان نقیض کرد.

چون که شبکه‌ی واحد های آستانه ای می‌توانند دسته‌ی وسیعی از توابع را یاد بگیرند (در مقابل تک واحد های آستانه ای که فقط تعداد کمی از توابع را یاد می‌گیرند)، علاقه‌ی ما بیشتر به شبکه های چند لایه‌ی این نوع واحدهاست.

۴.۴.۲ قانون آموزش پرسپترون‌ها

با وجود اینکه علاقه‌ی ما بیشتر به شبکه‌هایی با تعداد زیاد از واحدهاست، اما بیایید از نحوه‌ی یادگیری وزن‌های یک تک پرسپترون شروع کنیم. اینجا مسئله این است که برداری از وزن‌ها را بیابیم که پرسپترون با آن بتواند تمامی برای نمونه های آموزشی خروجی درست را تعیین کند.

برای حل چنین مسائلی الگوریتم‌های بسیاری وجود دارد. در اینجا ما به بررسی دو تا از این الگوریتم‌ها می‌پردازیم: قانون پرسپترون[1] و قانون دلتا[2] (نسخه ای از قانون LMS که در فصل ۱ برای یادگیری تابع ارزیابی استفاده شد). این دو الگوریتم تضمین می‌کنند که در شرایط خاص مختلف به فرضیه های مختلف قابل قبولی میل کنند. اهمیت چنین الگوریتم‌هایی از آن جهت است که پایه‌ی یادگیری برای شبکه‌ی با تعداد بالای واحد هستند.

[1] perceptron rule
[2] delta rule

یکی از راه‌های یادگیری بردار وزن‌ها ایجاد برداری تصادفی و امتحان کردن آن با تک‌تک نمونه های آموزشی است، اگر با خروجی یکی از نمونه سازگار نبود، وزن‌ها را عوض می‌کنیم، این فرایند آنقدر ادامه می‌یابد تا برداری پیدا شود که با تمامی نمونه های آموزشی سازگار باشد. بر اساس قانون آموزشی پرسپترون در هر مرحله وزن‌ها به صورت زیر تغییر می‌کنند:

$$w_i \leftarrow w_i + \Delta w_i$$

که در آن

$$\Delta w_i = \eta(t - o)x_i$$

در این رابطه o خروجی تابع هدف برای نمونه فعلی، o خروجی پرسپترون و η ثابتی به نام ضریب یادگیری[1] است. نقش ضریب یادگیری کنترل میزان تغییر وزن‌ها در هر مرحله است. ضریب یادگیری معمولاً عددی کوچک (مثلاً 0.1) است که با زیاد شدن تعداد تکرارها کم‌کم کمرنگ می‌شود.

چرا باید چنین فرایندی به سمت مقادیر درست برای وزن‌ها میل کند؟ برای درک بهتر، حالتی خاص را بررسی می‌کنیم. فرض کنید که نمونه های آموزشی همگی توسط پرسپترون درست دسته بندی می‌شوند در چنین شرایطی همیشه مقدار عبارت (t-o) صفر و متعاقباً Δw_i صفر خواهند بود پس وزن‌ها تغییر نخواهند کرد. حال فرض کنید که پرسپترون برای یک مثال که خروجی 1+ است اشتباهاً خروجی 1- می‌دهد. برای اینکه این اشتباه تصحیح شود وزن‌ها باید طوری تغییر کنند که مقدار $\vec{w}.\vec{x}$ بیشتر شود. مثلاً اگر $x_i > 0$، با افزایش w_i می‌توان مقدار پرسپترون را درست کرد. توجه داشته باشید که چون (t-o)، η، و x_i در این مثال همگی مثبتند w_i افزایش می‌یابد. برای مثال اگر

$$x_i = .8 \text{ و } \eta = .1 \text{ و } t = 1 \text{ و } o = -1$$

خواهیم داشت که

$$\Delta w_i = \eta(t - o)x_i = 0.1\big(1 - (-1)\big)0.8 = 0.16$$

از طرف دیگر اگر t=-1 و o=1 مقدار تغییر وزن به صورت عکس در می‌آمد و x_i کاهش می‌یافت.

در واقع، ثابت می‌شود که فرایند بالا به در طی تعداد محدودی تکرار به برداری از وزن‌ها خواهد رسید که تمامی نمونه های آموزشی را درست دسته بندی می‌کند (به شرط آنکه نمونه های آموزشی دسته بندی پذیر خطی و η نیز به اندازه‌ی کافی کوچک باشد (Papert 1969)). اگر داده‌ها دسته بندی پذیر خطی نباشند اطمینانی نیست که داده‌ها به مقدار خاصی میل کنند.

۴.۴.۳ شیب نزول[2] و قانون دلتا

با وجود اینکه قانون پرسپترون زمانی که داده‌ها دسته بندی پذیر خطی باشند به درستی برداری برای وزن‌ها پیدا می‌کند، اما در زمانی که داده‌ها دسته بندی پذیر خطی نیستند در این کار شکست می‌خورد. قانون آموزش دومی، به نام قانون دلتا، طراحی شده که حتی با وجود چنین مشکلی

[1] learning rate
[2] Gradient Descent

به مقدار خاصی میل کند. اگر داده‌ها دسته بندی پذیر خطی نباشند، قانون دلتا به سمتی میل می‌کند تا بهترین تقریب را از تابع هدف داشته باشد.

نکته‌ی کلیدی‌ای که در قانون دلتا به کار رفته این است که این قانون از شیب نزول برای جستجوی فضای فرضیه ای بردار های وزن را برای پیدا کردن متناسب‌ترین بردار استفاده می‌کند. اهمیت قانون دلتا از این رو است که پایه ای برای الگوریتم Backpropagation است. الگوریتم Backpropagation برای آموزش شبکه‌هایی با تعداد زیادی واحد به کار می‌رود. از سوی دیگر، شیب نزول پایه ای برای الگوریتم‌هایی که جستجو در فضای پیوسته‌ی فرضیه ای انجام می‌دهند است.

قانون دلتا، در پرسپترون‌های بدون مقدار آستانه قابل درک تر است. در چنین پرسپترون‌هایی داریم :

$$o(\vec{x}) = \vec{w}.\vec{x}$$

بنابراین، واحدی خطی (بدون مقدار آستانه) متناسب با هر پرسپترون مشخص می‌شود. برای اشتقاق یک وزن برای واحد خطی، از تعریف میزان خطای فرضیه (بردار های وزن) شروع می‌کنیم. با وجود اینکه توابع بسیاری برای بدست آوردن خطا وجود دارد اما تعریف می‌کنیم که:

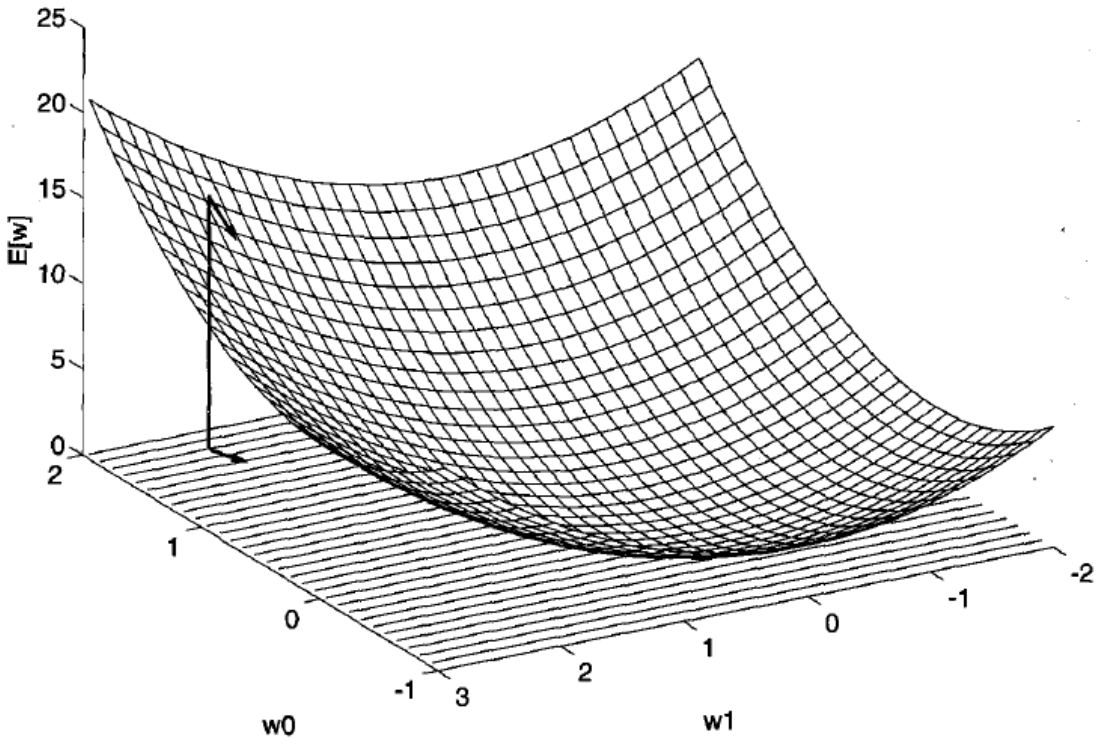
$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

در این رابطه D دسته نمونه‌ها، t_d مقدار تابع هدف برای نمونه d، و o_d مقدار خروجی پرسپترون برای نمونه‌ی d است. طبق این تعریف، $E(\vec{w})$ نصف مجموع مجذور اختلاف‌های بین تابع هدف t_d و خروجی پرسپترون خطی o_d در تمامی نمونه های آموزشی است. در اینجا ما E را به عنوان تابعی از $\vec{w}$ تعریف کرده‌ایم زیرا که خروجی o به $\vec{w}$ وابسته است. البته E علاوه بر $\vec{w}$ به نمونه های آموزشی نیز وابسته است اما این نمونه ثابت فرض شده‌اند. در فصل ۶ توجیهی بیزی[1] برای نحوه‌ی تعریف E می‌آوریم. در کل، نشان خواهیم داد که در تحت شرایطی فرضیه ای که E را مینیمم کند متناسب‌ترین فرضیه‌ی درون H با داده های آموزشی است.

۴.۴.۳.۱ تصور فضای فرضیه‌ها

برای درک الگوریتم شیب نزول، بد نیست فضای فرضیه ای و رابطه‌ی آن را با مقادیر E تصور کنیم (شکل ۴.۴). در شکل دو محور w_0 و w_1 دو مقدار ممکن برای بردار وزن واحد خطی هستند. محور سوم E میزان خطای مربوط به دسته ای از نمونه های آموزشی خاص را نشان می‌دهد. سطح خطای نشان داده شده در شکل ارجحیت هر بردار وزن را در فضای فرضیه‌ها نشان می‌دهد (بردارهایی ارجحیت دارند که خطای کمتری داشته باشند). با توجه به نحوه‌ی تعریف E، برای واحدهای خطی، سطح خطا همیشه سهمی‌وار است و یک نقطه‌ی مینیمم مطلق خواهد داشت. این نقطه‌ی مینیمم مطلق، همان طور که واضح است، به دسته نمونه های آموزشی وابسته است.

[1] Bayesian



شکل ۴.۴ خطای فرضیه های مختلف.

برای واحدی خطی با دو وزن، فضای فرضیه ای H صفحه‌ی w_1 و w_0 خواهد بود. محور عمودی میزان خطای فرضیه‌ها را برای دسته نمونه ثابتی نشان می‌دهد. فلش‌های شکل شیب منفی را در نقطه ای خاص نشان می‌دهند، این فلش‌ها به سمتی اشاره می‌کنند که میزان خطا در آنجا به حداقل می‌رسد.

جستجوی شیب نزول بردار وزنی را مشخص می‌کند که در آن E کمینه است. در این الگوریتم ابتدا از برداری دلخواه شروع کرده و مرحله به مرحله آن با تغییر های کوچک به بردار وزن مطلوب میل می‌کند. در هر مرحله، بردار وزن به طرف بیشترین کاهش خطا حرکت داده می‌شود. این فرایند آنقدر ادامه پیدا خواهد کرد تا به مینیمم مطلق تابع خطا برسیم.

٤.٤.٣.٢ اشتقاق قانون شیب نزول

چگونه می‌توان بیشترین کاهش خطا را پیدا کرد؟ این جهت با مشتق گرفتن ضمنی از میزان خطای E بر حسب تمامی مؤلفه های بردار $\vec{w}$ بدست می‌آید. این بردار گرادیان[1] E نامیده می‌شود و به صورت $\nabla E(\vec{w})$ نشان داده می‌شود.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right] \qquad (4.3)$$

توجه داشته باشید که خود $\nabla E(\vec{w})$ نیز یک بردار است که مؤلفه‌هایش مشتقات E بر حسب w_i هاست. زمانی که به گرادیان به صورت برداری در فضای وزن‌ها نگاه کنیم، سمت بیشترین افزایش E را مشخص خواهد کرد. در نقطه‌ی مقابل خلاف این سمت بیشترین کاهش E را

[1] gradient

به دنبال خواهد داشت. برای مثال، در شکل ۴.۴ عکس گرادیان $(-\nabla E(\vec{w}))$ برای نقطه ای دلخواه در صفحه‌ی w_1 و w_0 نشان داده شده است.

از آنجایی که گرادیان سمت بیشترین کاهش E را مشخص می‌کند، قانون یادگیری برای شیب نزول به شکل زیر خواهد بود:

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

که در آن

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \tag{4.4}$$

در اینجا نیز η مقداری مثبت است که ضریب یادگیری نامیده می‌شود. این مقدار اندازه‌ی قدم‌های را در الگوریتم شیب نزول مشخص می‌کند. علامت منفی به خاطر این است که می‌خواهیم بردار وزن‌ها را به سمت کاهش میزان E حرکت دهیم. می‌توان به صورت ساده تر این قانون را بر روی مؤلفه های بردار وزن‌ها نیز نوشت:

$$w_i \leftarrow w_i + \Delta w_i$$

که در آن

$$\Delta w_i = -\eta\left(\frac{\partial E}{\partial \mathrm{w}_i}\right) \tag{4.5}$$

این نشان می‌دهد که برای رسیدن به بیشترین کاهش باید هر مؤلفه را متناسب با مقدار $\frac{\partial E}{\partial \mathrm{w}_i}$ تغییر داد.

برای تبدیل این فرایند به الگوریتم و تکرار مراحل توسط رابطه‌ی (۴.۵) لازم است که راهی موثر برای محاسبه‌ی گرادیان در هر مرحله داشته باشیم. خوشبختانه این کار چندان هم مشکل نیست. مشتقات سازنده‌ی بردار گرادیان $\frac{\partial E}{\partial \mathrm{w}_i}$ به سادگی با استفاده از رابطه‌ی (۴.۲) محاسبه می‌شود:

$$\frac{\partial E}{\partial \mathrm{w}_i} = \frac{\partial}{\partial \mathrm{w}_i}\frac{1}{2}\sum_{d\in D}(\mathrm{t_d} - \mathrm{o_d})^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \mathrm{o_d})^2$$

$$= \frac{1}{2}\sum_{d\in D}2(\mathrm{t_d} - \mathrm{o_d})\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \mathrm{o_d})$$

$$= \frac{1}{2}\sum_{d\in D}2(\mathrm{t_d} - \mathrm{o_d})\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \vec{\mathrm{w}}.\vec{\mathrm{x}}_\mathrm{d})$$

$$\frac{\partial E}{\partial \mathrm{w}_i} = \sum_{d\in D}(\mathrm{t_d} - \mathrm{o_d})(-\mathrm{x_{id}}) \tag{4.6}$$

در این رابطه x_{id} نشان دهنده‌ی مؤلفه‌ی i ام در نمونه‌ی d است. حالا ما معادلی برای $\frac{\partial E}{\partial w_i}$ داریم که به مقادیر x_{id}، o_d و t_d (مقدار تابع هدف نمونه آموزشی) وابسته است. با جایگزینی مقادیر رابطه‌ی (4.6) در رابطه‌ی (4.5) رابطه‌ی تغییر مقادیر وزن‌ها برای شیب نزول بدست می‌آید:

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)(x_{id}) \qquad (4.7)$$

به طور خلاصه، الگوریتم شیب نزول برای آموزش واحد های خطی به صورت زیر است: ابتدا برداری دلخواه برای وزن‌ها انتخاب کن. سپس برای هر نمونه آموزشی مقدار واحد خطی را محاسبه کن، و Δw_i ها را برای هر وزن حساب کن (رابطه‌ی ۴.۷). هر وزن را با اضافه کردن Δw_i تغییر بده و این فرایند را تا اتمام نمونه های آموزشی تکرار کن. در جدول ۴.۱ این الگوریتم آورده شده است. چون سطح خطا فقط یک مینیمم مطلق دارد، این الگوریتم به برداری با کمترین خطا میل می‌کند، بدون توجه به اینکه داده‌ها دسته پذیر خطی هستند یا نه. فقط کافی است که η به اندازه‌ی کافی کوچک باشد. اگر η خیلی بزرگ باشد، احتمال دارد الگوریتم شیب نزول به کمترین مقدار خطا میل نکند. یکی از روش‌های حل این مشکل کم کردن تدریجی η در طول مراحل الگوریتم است.

٤.٤.٣.٣ تقریب اتفاقی شیب نزول

شیب نزول نمونه‌ی کلی مهمی از یادگیری است. این الگوریتم استراتژی‌ای برای جستجوی فضاهای بزرگ و نامتناهی فرضیه ای است. از این الگوریتم به شرطی می‌توان استفاده کرد که (1) فضای فرضیه ای پیوسته (برای مثال، فضای وزن‌ها در واحد خطی)، و (2) خطاها بر حسب پارامترهای این فرضیه صریح باشد. مشکلات استفاده از شیب نزول این است که (1) همگرایی به یک مقدار مینیمم موضعی بعضی مواقع زیادی طول می‌کشد (مثلاً، صدها گام لازم است تا به مقدار خاصی همگرا شویم) و (2) اگر چند مینیمم موضعی وجود داشته باشد تضمینی نیست که الگوریتم به مینیمم مطلق میل کند.

الگوریتم Gradient-Descent(training_examples,η)

هر نمونه آموزشی به صورت $< \vec{x}, t >$ مشخص می‌شود، $\vec{x}$ نمونه و t مقدار نمونه است. η نیز نرخ یادگیری را تعیین می‌کند.

- w_i ها را با مقادیر دلخواه کوچکی مقدار دهی اولیه کن.
- تا زمانی که به شرط پایانی نرسیده ای حلقه‌ی زیر را اجرا کن
  - هر Δw_i را صفر مقدار دهی اولیه کن.
  - برای هر مثال $< \vec{x}, t >$ حلقه‌ی زیر را اجرا کن
    - $\vec{x}$ را به واحد خطی بده و خروجی o را دریافت کن
    - برای هر وزن w_i دستور زیر را انجام بده

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \qquad (T4.1)$$

  - برای هر وزن واحد خطی دستور زیر را انجام بده

$$w_i \leftarrow w_i + \Delta w_i \qquad (T4.2)$$

جدول ۴.۱ الگوریتم Gradient-Descent برای آموزش یک واحد خطی.

برای تبدیل به تقریب اتفاقی برای شیب نزول رابطه‌ی (T4.2) حذف می‌شود و رابطه‌ی (T4.1) نیز با رابطه‌ی $\boldsymbol{w_i \leftarrow w_i + \eta(t-o)x_i}$ *جایگزین می‌شود.*

یکی از راه‌های حل این مشکلات ، استفاده از متد شیب نزولی افزایشی[1] یا متد شیب نزولی تصادفی[2] است. قانون شیب نزول تغییر وزن‌ها را بعد از جمع بستن همه‌ی نمونه‌ها انجام می‌دهد (رابطه‌ی 4.7). اما متد شیب نزول تصادفی سعی می‌کند تا با افزایش ذره ذره‌ی وزن‌ها روش جستجوی شیب نزول را تخمین بزند و سپس خطا را برای هر نمونه محاسبه کند. این قانون آموزش نظیر قانون آموزش بیان شده در معادله‌ی 4.7 است با این فرق که بعد از هر تکرار طبق رابطه‌ی زیر وزن‌ها را تغییر می‌دهیم

$$\Delta w_i = \eta(t-o)x_i \tag{4.10}$$

در این رابطه t، o و x_i به ترتیب مقدار تابع هدف، خروجی واحد و i امین ویژگی نمونه آموزشی مورد بحث هستند. برای تبدیل الگوریتم شیب نزول (جدول ۴.۱) به شیب نزول تصادفی، رابطه‌ی (T4.2) حذف و به جای رابطه‌ی (T4.1) رابطه‌ی $w_i \leftarrow w_i + \eta(t-o)x_i$ جایگزین می‌شود. برای بیان الگوریتم شیب نزول تصادفی به بیانی دیگر کافی است تابع خطایی به نام $E_d(\vec{w})$ را برای هر نمونه آموزشی d به شکل زیر تعریف کنیم:

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2 \tag{4.11}$$

که در آن t_d و o_d به ترتیب مقدار تابع هدف و خروجی واحد برای نمونه d هستند. الگوریتم شیب نزول تصادفی برای تمامی نمونه های d در D تکرار خواهد شد و در هر تکرار وزن‌ها را بر اساس گرادیان و با توجه به $E_d(\vec{w})$ تغییر می‌دهد. سری‌ای از این تغییر وزن‌ها معیار خوبی برای تخمین کاهش گرادیان با توجه به تابع خطای اصلی $E(\vec{w})$ است. با کم کردن مقدار η (اندازه‌ی قدم‌ها در شیب نزول) به اندازه‌ی کافی، شیب نزول تصادفی می‌تواند به اندازه‌ی دلخواه به خود شیب نزول نزدیک شود. فرق‌های اساسی بین شیب نزول و شیب نزول تصادفی در زیر آورده شده است:

- در شیب نزول، خطا برای تمامی نمونه‌ها قبل از تغییر در وزن‌ها جمع زده می‌شد اما در شیب نزول تصادفی محاسبه‌ی خطاها و تغییر وزن‌ها همزمان انجام می‌شود.
- جمع خطا برای چندین نمونه در شیب نزول نیاز به محاسبات بیشتری در هر تکرار حلقه دارد. در مقابل چون از گرادیان اصلی برای تغییرات استفاده می‌شود، در هر قدم (به نسبت شیب نزول تصادفی) بیشتر به مینیمم E نزدیک می‌شود.
- در مواقعی که $E(\vec{w})$ چندین مینیمم‌های موضعی دارد گاهی شیب نزول تصادفی می‌تواند از افتادن در چنین مینیمم‌هایی پرهیز کند زیرا که شیب نزول تصادفی برای کنترل سمت جستجو بجای $\nabla E(\vec{w})$ از $\nabla E_d(\vec{w})$ استفاده می کند.

هر دو الگوریتم شیب نزول و شیب نزول تصادفی به یک اندازه در کاربرد استفاده می‌شوند.

به قانون آموزش رابطه‌ی (4.10) را قانون دلتا، LMS[3]، قانون Adaline، یا قانون Window-Hoff (همنام ارائه کننده) نیز می‌نامند. در فصل ۱ از LMS برای توصیف کاربردش برای یادگیری ارزیابی‌ای از بازی استفاده کردیم. توجه داشته باشید که قانون دلتا در رابطه‌ی 4.10 مشابه قانون آموزش پرسپترون‌ها در قسمت ۴.۴.۲ است. در واقع از نظر ظاهری این دو رابطه با هم یکی هستند، با این وجود قانون دلتا o

[1] incremental gradient descent
[2] stochastic gradient descent
[3] least-mean-square

مربوط به رابطه‌ی خروجی واحد خطی یعنی $o(\vec{x}) = \vec{w}.\vec{x}$ و قانون پرسپترون o مربوط به رابطه‌ی خروجی واحد آستانه یعنی $o(\vec{x}) = sgn(\vec{w}.\vec{x})$ است.

با وجود اینکه قانون دلتا برای واحد های خطی بدون مقدار آستانه بررسی شد اما می‌توان این قانون را برای آموزش پرسپترون‌ها نیز استفاده کرد. فرض کنید که $o = \vec{w}.\vec{x}$ خروجی بدون مقدار آستانه واحد خطی باشد و $o' = sgn(\vec{w}.\vec{x})$ خروجی واحد آستانه دار یا همان پرسپترون باشد. حال اگر می‌خواهیم که پرسپترون را با توجه به مقادیر تابع هدف که ± 1 هستند آموزش دهیم، می‌توانیم همان مقادیر را با استفاده از قانون برای آموزش o به کار ببریم. واضح است که اگر واحد خطی بتواند تمامی نمونه‌ها را یاد بگیرد پرسپترون نیز به حالت نظیر می‌تواند تمامی نمونه‌ها را یاد بگیرد (زیرا که sgn(1)=1 و sgn(-1)=-1). حتی زمانی که نمی‌توان با استفاده از واحد خطی همه‌ی نمونه‌ها را به دقت یاد گرفت مقدار آستانه این کمبود دقت را جبران می‌کند و مقادیر خروجی را به ± 1 می‌رساند (فقط کافیست واحد خطی علامت خروجی را درست تعیین کرده باشد). توجه داشته باشید که تلاش فرایند برای کم کردن خطای واحد خطی o است و ممکن است الزاماً دسته وزن‌هایی را مشخص نمی‌کند که کمترین خطای دسته بندی o' را داشته باشد.

٤.٤.٤ ملاحظات

در قسمت قبلی دو الگوریتم تکراری[1] برای پیدا کردن وزن‌های پرسپترون ارائه کردیم. تفاوت این دو الگوریتم در اینجا است که قانون پرسپترون وزن‌ها را برای پرسپترون‌هایی با مقدار آستانه پیدا می‌کند اما قانون دلتا وزن‌ها را برای پرسپترون‌هایی بدون مقدار آستانه پیدا می‌کند.

تفاوت این دو الگوریتم بر ویژگی‌های همگرایی آن‌ها نیز تأثیر گذاشته است. قانون پرسپترون، با فرض اینکه داده‌ها دسته بندی پذیر خطی باشند، پس از تعداد محدودی تکرار به فرضیه‌ی درست می‌رسد. در حالی که قانون دلتا به طور مجانبی به فرضیه‌ی درست میل می‌کند، و ممکن است برای همگرایی تا بی نهایت طول بکشد. در عوض قانون دلتا بدون توجه نیاز به دسته بندی پذیر خطی بودن داده‌ها همگرا می‌شود. برای اطلاعات بیشتر در مورد همگرایی این دو روش به Hertz et al. (1991) مراجعه کنید.

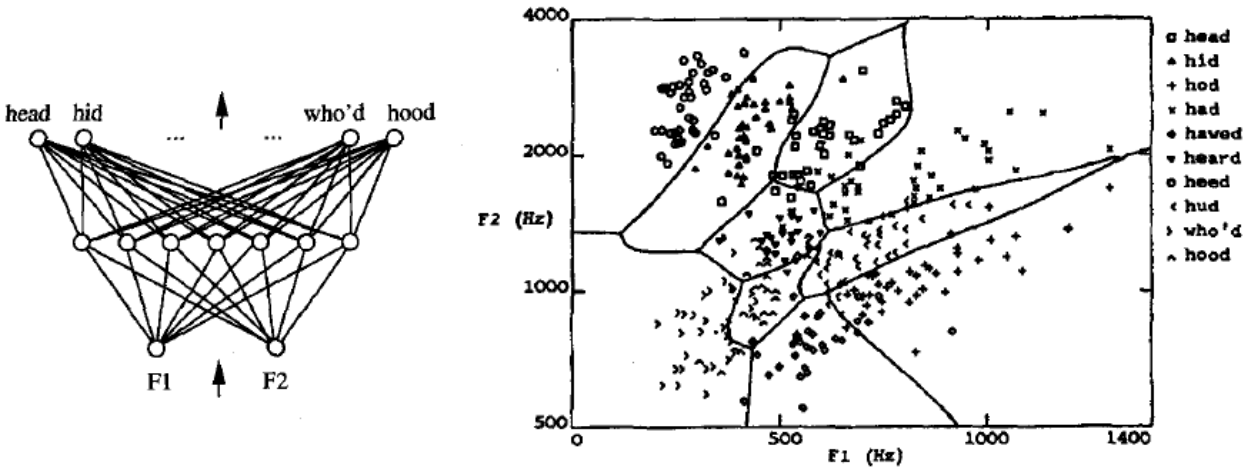
الگوریتم سوم برای یادگیری بردار وزن‌ها برنامه نویسی خطی[2] است. برنامه نویسی خطی متدی کارامد و کلی برای حل نامساوی‌های خطی است. توجه دارید که هر نمونه آموزشی متناسب با یک نامساوی به فرم $\vec{w}.\vec{x} > 0$ یا $\vec{w}.\vec{x} \leq 0$ است و جواب نا معادله نیز همان بردار وزن‌هاست. متأسفانه این روش نیز فقط زمانی به جواب می‌رسد که داده‌ها دسته بندی پذیر خطی باشند، با این وجود (Duda and Hart 1973, p. 168) فرمولی زیرکانه برای مواقعی که داده‌ها دسته بندی پذیر خطی نیز نیستند پیشنهاد داده است. به هر حال روش برنامه نویسی خطی برای شبکه های چند لایه تعمیم ندارد. در مقابل، روش شیب نزول که قانون دلتا نیز با کمک آن ساخته شده، به راحتی برای شبکه های چند لایه تعمیم می‌یابد. در قسمت آینده این تعمیم را بررسی خواهیم کرد.

٤.٥ شبکه های چند لایه و الگوریتم Backpropagation

همان طور که در قسمت ۴.۴.۱ نیز گفته شد، تک پرسپترون‌ها فقط سطوح خطی تصمیم گیری را می‌توانند یاد بگیرند. در مقابل، شبکه های چند لایه که توسط الگوریتم Backpropagation آموزش داده می‌شوند می‌توانند انواع مختلفی از سطوح تصمیم گیری غیر خطی را نیز یاد

[1] iterative
[2] linear programming

بگیرند. برای مثال، یک شبکه‌ی چند لایه و سطح تصمیم گیری آن در شکل ۴.۵ نشان داده شده است. در این مثال کار تشخیص گفتار برای تشخیص حرف صدا دار بین دو حرف بی صدای h و d (در ده حالت مختلف) آورده شده. ورودی سیگنال صحبت به صورت دو پارامتر عددی که از آنالیز صدا بدست آمده می‌باشد. این اعداد سطح ۲ بعدی تصمیم گیری را تشکیل می‌دهند. همان طور که در شکل نیز نشان داده شده شبکه های چند لایه می‌توانند سطوح تصمیم گیری خیلی پیچیده تری را نسبت به سطوح خطی (شکل ۴.۳) یاد بگیرند. در این بخش به نحوه‌ی آموزش شبکه های چند لایه توسط الگوریتم شیب نزول می‌پردازیم.



شکل ۴.۵ فضای تصمیم گیری یک شبکه‌ی چند لایه‌ی تک سویه.
شبکه‌ی نشان داده شده برای تشخیص یکی از ده صدای بین حروف h و d آموزش داده شده است. ورودی شبکه دو پارامتر F1 و F2 هستند که از آنالیز صدا بدست می‌آیند. ده خروجی شبکه متناسب با ده صدای مختلف هستند. پیش بینی شبکه صدایی است که بیشترین مقدار خروجی شبکه را داشته باشد. سمت راست سطح تصمیم گیری غیر خطی این شبکه را نشان می‌دهد. نقطه های نشان داده شده در شکل نمونه های آموزشی هستند. (گرفته شده از (Haung and Lippmann 1988).

٤.٥.١ واحد آستانه ای مشتق پذیر

چه نوع واحدهایی برای تشکیل پایه های شبکه های چند لایه به کار می‌روند؟ در ابتدا ممکن است فکر کنیم که واحد های خطی‌ای که پیش‌تر قانون یادگیریشان را پیدا کردیم مناسبند. با این وجود، درحالی‌که شبکه‌هایی که ترکیب واحد های خطی‌اند فقط توابع خطی را ایجاد می‌کنند، در حالی که هدف ما از شبکه های چند لایه پیدا کردن شبکه‌هایی است که توابع غیر خطی را بیان کنند. گزینه‌ی دیگر واحد پرسپترون است، اما ناپیوستگی مقدار آستانه‌ی این واحد آن‌را مشتق ناپذیر می‌کند. و واحدهایی که مشتق ناپذیرند، گرادیان ندارند و متعاقباً برای شیب نزول مناسب نیستند. در اینجا به واحدی با خروجی مشتق پذیر و غیر خطی نیاز داریم. واحد سیگموید یکی از راه حل‌های ممکن است. واحدی که خیلی مشابه پرسپترون و تابع مقدار آستانه‌اش پیوسته و مشتق پذیر است.

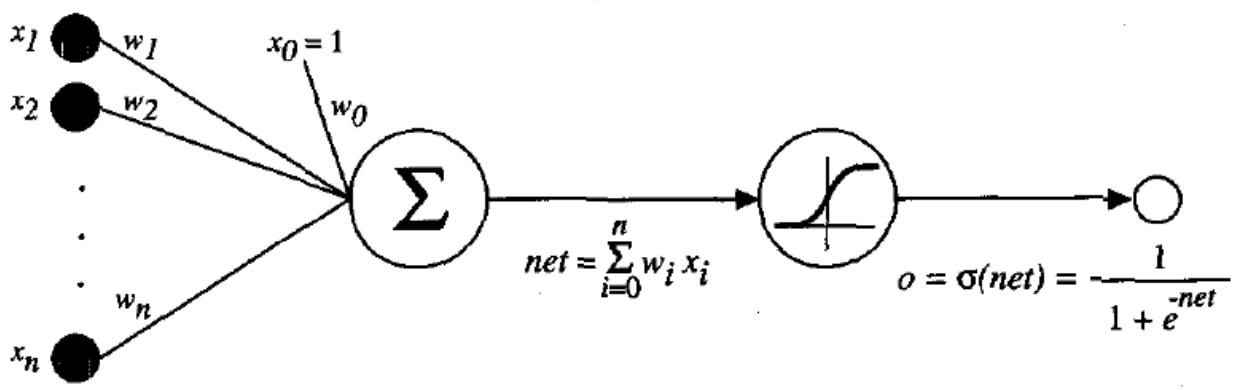
تابع سیگموید در شکل ۴.۶ نشان داده شده. مثل واحد پرسپترون، سیگموید نیز ابتدا ترکیبی خطی از ورودی‌ها را محاسبه کرده و سپس آن‌را بعد از تأثیر تابع آستانه‌اش خروجی می‌دهد. در واحد سیگموید خروجی تابعی پیوسته از ورودی‌هاست. به عبارت دقیق‌تر خروجی واحد سیگموید از فرمول زیر محاسبه می‌شود:

$$o = \sigma(\vec{w}.\vec{x})$$

که در آن

$$\sigma(y) = \frac{1}{1+e^{-y}} \qquad (4.12)$$

به تابع σ تابع سیگموید[1] یا تابع منطق[2] نیز می‌گویند. توجه دارید که خروجی این تابع عددی بین صفر تا یک است، که متناسب با ورودی‌هاست. (تابع سیگموید در شکل ۴.۶ نشان داده شده). چون تابع سیگموید پهنای بزرگی از خروجی‌ها را به پهنای کوچکی می‌برد گاهی به آن تابع فشرده ساز[3] نیز می‌گویند. یکی دیگر از خواص بسیار مفید تابع سیگموید بیان مشتق آن بر حسب خودش است. [به عبارت دیگر، $\frac{\partial\sigma(y)}{\partial y} = \sigma(y).(1-\sigma(y))$]. همان طور که بعداً نیز خواهیم دید، در استفاده از گرادیان این رابطه محاسبات را بسیار ساده تر می کند. توابع مشتق پذیر دیگر در بعضی موارد به جای σ به کار می روند. برای مثال گاهی به جای e^{-y} در تابع سیگموید $e^{-k.y}$ قرار می دهند که در آن k عددی ثابت و مثبت است که تندی مقدار آستانه را مشخص می کند. در بعضی موارد نیز از تابع tanh به جای سیگموید استفاده می شود (تمرین ۴.۸).



شکل ۴.۶ واحد آستانه ای سیگموید.

۴.۵.۲ الگوریتم Backpropagation

الگوریتم Backpropagation وزن‌های لازم برای یک شبکه‌ی چند لایه با ساختار شبکه‌ی ثابت را پیدا می‌کند. این الگوریتم از شیب نزول برای مینیمم کردن میزان خطا، مربع اختلاف بین خروجی شبکه و تابع هدف، استفاده می‌کند. در این بخش الگوریتم Backpropagation و در بخش بعدی مشتقات لازم برای قانون شیب نزول وزن‌ها در این الگوریتم را ارائه می‌کنیم.

چون در این شبکه‌ها خروجی یک عدد نیست، پس کار را با تعریف دوباره‌ی E آغاز می‌کنیم و E را جمع خطای تمامی خروجی‌ها تعریف می‌کنیم:

---

[1] sigmoid
[2] logistic function
[3] squashing function

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2 \qquad (4.13)$$

که در آن outputs مجموعه‌ی تمامی خروجی‌های شبکه، t_{kd} مقدار تابع هدف و o_{kd} خروجی شبکه برای k امین خروجی و d امین نمونه است.

مسئله فعلی پیدا کردن وزن‌های متناسب با نمونه های آموزشی در میان فضای فرضیه ای تمامی وزن‌های ممکن است. می‌توان وضعیت را دوباره مثل شکل ۴.۴ تصور کرد. در وضعیت محور عمودی تعریف جدید E، و بقیه‌ی محورها، وزن‌های تمامی واحد های شبکه هستند. درست مشابه زمانی که تنها یک واحد داشتیم در این وضعیت نیز می‌توان از شیب نزول برای یافتن فرضیه ای با کمترین میزان خطا کمک گرفت.

تنها فرق این است که برعکس حالت قبلی که فقط یک مینیمم داشت (شکل ۴.۴) در این حالت ممکن است چندین مینمم موضعی وجود داشته باشد. متأسفانه شیب نزول تنها تضمین می‌کند که به سوی مینیممی موضعی میل کند، و این مقدار همیشه با مینیمم مطلق یکی نیست. با وجود این مانع، در عمل Backpropagation ثابت کرده که می‌تواند جواب‌های بسیار خوبی در کاربردهای واقعی پیدا کند.

الگوریتم Backpropagation در جدول ۴.۲ آورده شده است. این الگوریتم برای شبکه های یک سویه ای است که از دو لایه واحد سیگموید تشکیل شده‌اند که هر واحد در هر لایه به تمامی واحدهای لایه‌ی قبلی متصل است. این الگوریتم یکی از دو نسخه‌ی شیب نزول تصادفی یا افزایشی الگوریتم Backpropagation است. نماد گزاری استفاده شده جز در موارد زیر مشابه قسمت‌های قبلی است:

- به هر گره[1] یک اندیس[2] نسبت داده شده است که در آن گره یک ورودی به شبکه یا خروجی واحدی است.
- x_{ij} نماینده‌ی ورودی گره‌ی i به واحد j است و w_{ij} وزن این متناظر است.
- δ_n نماد خطای مربوط به واحد n است. و نقش مقدار (t-o) را که قبلاً در قانون دلتا درباره‌ی آن بحث کردیم ایفا می‌کند. همان طور که بعداً نیز خواهیم دید: $\delta_n = \frac{\partial E}{\partial net_n}$.

Backpropagation ($training_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

هر نمونه آموزشی به صورت زوج مرتب $<\vec{x},\vec{t}>$ مشخص می‌شود که در آن $\vec{x}$ بردار مقدار های ورودی شبکه و $\vec{t}$ مقادیر تابع هدف است.

η ضریب یادگیری است، n_{in} تعداد ورودی‌های شبکه n_{out} تعداد خروجی‌های شبکه و n_{hidden} تعداد واحدها پنهان شبکه هستند.

ارتباط بین واحد i ام و واحد j ام به صورت x_{ij} نشان داده شده است و وزن متناسب با این ارتباط نیز با نماد w_{ij} نشان داده شده.

به هر گره[3] یک اندیس[4] نسبت داده شده است که در آن گره یک ورودی به شبکه یا خروجی واحدی است.

- شبکه ای یک طرفه با n_{in} واحد ورودی n_{hidden} واحد پنهان و n_{out} واحد خروجی بساز

---

[1] node
[2] index
[3] node
[4] index

- تمامی وزن‌های شبکه را با اعداد کوچک تصادفی مقدار دهی اولیه کن (مثلاً بین ۰.۵ و 0.5-)
- تا رسیدن به شرط پایانی حلقه‌ی زیر را اجرا کن
  - برای هر مثال $<\vec{x},\vec{t}>$ در training_examples حلقه‌ی زیر را اجرا کن

ورودی را در جهت شبکه میان شبکه پخش کن:

۱. $\vec{x}$ را به ورودی بده و خروجی o_u را برای هر خروجی u دریافت کن

خطاها را خلاف جهت شبکه در میان شبکه پخش کن:

۲. برای هر خروجی k مقدار δ_k را از رابطه‌ی زیر بدست آور

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k) \qquad (T4.3)$$

۳. برای هر واحد پنهان h مقدار زیر را حساب کن

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in output} w_{kh}\delta_k \qquad (T4.4)$$

۴. هر وزن w_{ji} را از رابطه‌ی زیر تغییر بده

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

که در آن

$$\Delta w_{ji} = \eta\delta_j x_{ji}$$

جدول ۴.۲ نسخه‌ی شیب نزول اتفاقی الگوریتم Backpropagation برای شبکه های تک سویه که دو لایه واحد سیگموید دارند.

توجه دارید که الگوریتم جدول ۴.۲ با ساخت یک شبکه‌ی جدید با همان تعداد واحد پنهان و همان تعداد واحد خروجی و مقدار دهی اولیه‌ی وزن‌های آن با اعداد تصادفی کوچک آغاز می‌گردد. با توجه به اینکه ساختار شبکه ثابت و معلوم است، حلقه‌ی اصلی الگوریتم فقط برای نمونه های آموزشی مختلف تکرار می‌شود و بقیه‌ی موارد تغییری نمی‌کنند. برای هر نمونه‌ی آموزشی، نمونه به شبکه داده شده و خروجی را دریافت می‌شود، سپس خطای خروجی را برای نمونه مذکور محاسبه می‌کند. در ادامه، گرادیان را با توجه به خطای محاسبه شده محاسبه و در آخر نیز مقدار وزن‌ها را تغییر می‌دهد. این مرحله‌ی شیب نزول تا زمانی که خطای شبکه به حد مطلوب برسد تکرار می‌شود (گاهی این تکرارها تا صدها بار ادامه می‌یابد و همان نمونه‌ها چندین دفعه تکرار می‌شوند).

قانون تغییر وزن‌های شیب نزول (رابطه‌ی [T4.5] در جدول ۴.۲) مشابه رابطه‌ی قانون دلتا (رابطه‌ی [4.10]) است. مثل قانون دلتا، این رابطه مقدار هر وزن را به نسبت ضریب یادگیری η و مقدار ورودی x_{ji} که وزن به آن اعمال شده و مقدار خطای خروجی تغییر می‌دهد. تنها تفاوت بین این دو رابطه این است که خطا در قانون دلتا (t-o) بوده و در رابطه‌ی جدید با مقداری پیچیده تر δ_j جایگزین شده است. صورت دقیق δ_j از اشتقاق رابطه‌ی تغییر وزن‌ها در قسمت ۴.۵.۳ ناشی شده است. برای درک بهتر، ابتدا به فرمول محاسبه‌ی δ_k برای خروجی k ام شبکه دقت کنید (رابطه‌ی [T4.4]). با این وجود، از آنجایی که نمونه های آموزشی مقدار t_k را برای خروجی‌های شبکه دارند، پس مقدار تابع هدف برای واحد های پنهان معلوم نیست و نمی‌توان خطا را به صورت مستقیم محاسبه کرد. پس به جای آن برای محاسبه‌ی خطای واحد پنهان h، از جمع

خطا های δ_k برای هر خروجی که h بر آن تأثیر دارد می‌شود. برای محاسبه‌ی درست‌تر لازم است که هر میزان خطا در میزان تأثیر h بر k w_{kh} ضرب شود. این کار باعث می‌شود که هر واحد پنهان به اندازه ای که در هر خطا "مسئول[1]" است در خطا سهم داشته باشد.

الگوریتم جدول ۴.۲ وزن‌ها را به صورت افزایشی و با برخورد به نمونه های مختلف تغییر می‌دهد. این تقریب تصادفی از شیب نزول است. برای رسیدن به خود گرادیان E باید مقادیر $\delta_j x_{ji}$ را قبل از تغییر وزن‌ها برای تمامی نمونه های آموزشی جمع زد.

حلقه‌ی تغییر وزن‌ها در Backpropagation در کاربردهای واقعی ممکن است صدها بار تکرار شود. با داشتن تنوع شروط خروج[2] می‌توان تکرار این فرایند را به متوقف کند. مثلاً ممکن است می‌توان تعیین کرد که شرط پس از تعداد خاصی تکرار متوقف شود، یا زمانی که مقدار خطا به کمتر از مقدار آستانه‌ی خاصی رسید، یا میزان خطا برای دسته نمونه های جداگانه به کمتر از مقدار خاصی برسد. انتخاب شرط پایانی از اهمیت خاص برخوردار است زیرا که تعداد کم تکرار ممکن است به مینیمم نشدن خطا بینجامد و تکرار زیاد نیز باعث می‌شود که شبکه فقط نمونه های آموزشی را تشخیص دهد (مشکل overfit). درباره‌ی این مسئله بعداً در قسمت ۴.۶.۵ مفصلاً بحث خواهد شد.

٤.٥.٢.١ اضافه کردن تکانه

چون Backpropagation الگوریتم پرکاربردی است، نسخه های بسیاری از این الگوریتم پدید آمده است. شاید معروف‌ترین این نسخه‌ها، نسخه ای است که به جای رابطه‌ی تغییر وزن‌ها (رابطه‌ی (T4.5)) از رابطه‌ی بازگشتی استفاده می‌کند:

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha w_{ji}(n-1)$$

در اینجا $\Delta w_{ji}(n)$ تغییر وزنی است که در حلقه‌ی n ام حلقه‌ی اصلی انجام می‌شود. و به α که 0≤α<1 تکانه[3] می‌گویند. توجه دارید که جمله‌ی اول این رابطه همان تغییر وزن در رابطه‌ی (T4.5) است. جمله اضافی، جمله‌ی دوم، جمله‌ی تکانه نامیده می‌شود. برای درک بهتر، فرض کنید که در الگوریتم شیب نزول مسیر طی شده توسط یک گوی طی می‌شد، در آنجا این گوی هیچ تکانه ای نداشت. اثر α اضافه کردن تکانه به گوی مورد بحث است، و باعث می‌شود در هر حلقه ما تمایل داشته باشیم به سمتی حرکت کنیم که در حلقه‌ی قبلی به آن سمت حرکت کرده‌ایم. این اثر باعث به دام نیفتادن در مینیمم‌های نسبی‌ای که خطا خیلی در آن کم نمی‌شود و حرکت به سمت مینیمم مطلق خواهد شد. همچنین در جایی که سطح افقی می‌شود گوی بدون تکانه از حرکت باز می‌ایستد در حالی که گویی که تکانه دارد چنین مشکلی ندارد. همچنین در جایی که شیب تغییر نمی‌کند، اندازه‌ی قدم‌ها را بیشتر می‌کند تا در حلقه های کمتری به مینیمم برسیم.

٤.٥.٢.٢ یادگیری در شبکه های بدون دور با ساختار دلخواه

تعریفی که در جدول ۴.۲ از Backpropagation آورده شد فقط برای شبکه های دو لایه بود، با این وجود به راحتی می‌توان این تعریف را برای تمامی شبکه های تک سویه تعمیم داد. در این تعمیم، تغییری در رابطه‌ی تغییر وزن‌ها (رابطه‌ی (T4.5)) به وجود نمی‌آید و فقط رابطه‌ی محاسبه‌ی δ عوض می‌شود. در کل برای محاسبه‌ی δ_r برای واحد r از لایه‌ی m از مقدار δ_s های لایه‌ی بعدی از فرمول زیر محاسبه می‌شود:

---

[1] responsible
[2] termination condition
[3] momentum

$$\delta_r = o_r(1 - o_r) \sum_{s \in layer\ m+1} w_{sr}\delta_s \qquad (4.19)$$

توجه دارید که این رابطه معادل پله‌ی سوم در الگوریتم جدول ۴.۲ است، و این پله در الگوریتم باید برای هر لایه‌ی پنهان در شبکه تکرار شود.

در واقع چنین استراتژی‌ای را می‌توان برای تمام شبکه‌هایی که ساختاری مشابه گراف‌های بدون دور دارند به کار گرفت، و نیازی به لایه ای بودن ساختار گراف نیست. برای شبکه‌هایی که لایه ای نیستند رابطه‌ی محاسبه‌ی δ برای تمامی واحد های میانی به فرم زیر خواهد بود:

$$\delta_r = o_r(1 - o_r) \sum_{s \in Downstream(r)} w_{sr}\delta_s \qquad (4.20)$$

در رابطه‌ی فوق Downstream(r) مجموعه‌ی تمامی واحدهایی است که به طور مستقیم از واحد r ورودی دریافت می‌کنند یا به عبارت دیگر تمامی واحدهایی که مستقیماً پایین r هستند. در قسمت بعدی برای محاسبات از این فرم استفاده می‌کنیم.

٤.٥.٣ اشتقاق قانون Backpropagation

در این بخش به مشتق رابطه‌ی تغییر وزن در قانون Backpropagation می‌پردازیم. می‌توانید در اولین خواندن این کتاب این قسمت را نخوانید!

در اینجا ما به اشتقاق رابطه‌ی شیب نزول تصادفی استفاده شده در جدول ۴.۲ می‌پردازیم. با توجه به رابطه‌ی ۴.۱۱ داریم که شیب نزول تصادفی به هر نمونه به طور جداگانه نگاه می‌کند و برای هر نمونه d خطای E_d را به طور مجزا کم می‌کند. به عبارت دیگر، برای هر نمونه آموزشی d هر وزن w_{ji} با اضافه کردن Δw_{ji} تغییر می کند:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \mathrm{w_{ji}}} \qquad (4.21)$$

که در آن E_d طبق تعریف خطای نمونه d است که برای تمام خروجی‌های شبکه محاسبه و جمع زده شده است:

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

در این رابطه outputs مجموعه‌ی واحد های خروجی در شبکه، t_k مقدار تابع هدف برای خروجی k ام و نمونه آموزشی d و o_k مقدار خروجی واحد k ام در شبکه برای نمونه d است.

اشتقاق قانون شیب نزول تصادفی از نظر مفهومی آسان است اما نیاز به توجه به اندیس‌ها و متغیرها دارد. در اینجا از همان نمایش شکل ۴.۶ استفاده می‌کنیم با این تفاوت که اندیس j را برای نمایش j امین واحد شبکه اضافه می‌کنیم:

- x_{ji} = i امین ورودی به واحد j

- w_{ji} = وزن مربوطه‌ی i امین ورودی به واحد j
- $net_j = \sum_i w_{ji} x_{ji}$ (مجموع وزن دار ورودی های واحد j)
- o_j = خروجی محاسبه شده برای واحد j
- σ = تابع سیگموید
- outputs = مجموعه‌ی واحد های خروجی در لایه‌ی آخر شبکه
- Downstream(j) = مجموعه‌ی تمامی واحدهایی که از خروجی واحد j (در ورودی) استفاده می‌کنند

حال مقدار عبارت $\frac{\partial E_d}{\partial w_{ji}}$ را برای قانون شیب نزول تصادفی که در رابطه‌ی ۴.۲۱ آمده محاسبه می‌کنیم. برای شروع، توجه داشته باشید کـه وزن w_{ji} فقط از طریق net_j بر شبکه اثر بگذارد. پس با توجه به قاعده‌ی زنجیره ای مشتق داریم:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j}\, x_{ji} \qquad (4.22)$$

با توجه به رابطه‌ی ۴.۲۲، فقط کافی است که $\frac{\partial E_d}{\partial net_j}$ را به طرز قابل قبولی بیان کنیم. دو حالت را در نظر می‌گیریم: حالتی که واحـد j واحـدی خروجی است و حالتی که واحد j واحدی داخلی است.

حالت اول: قانون آموزش برای واحد های خروجی. همان طور که گفته شد w_{ji} فقط از طریق net_j می‌تواند بر بقیه‌ی شـبکه تـأثیر بگذارد و net_j نیز فقط از طریق o_j می‌تواند بر شبکه تأثیر بگذارد. بنابراین با توجه به قاعده‌ی زنجیره ای در مشتق:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \qquad (4.23)$$

برای شروع، فقط جمله‌ی اول رابطه‌ی ۴.۲۳ را محاسبه می‌کنیم:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

مقدار مشتق $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ برای تمامی خروجی‌های k به جز j صفر است. پس خواهیم داشت:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}\, 2\,(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j) \qquad (4.24)$$

حالا جمله‌ی دوم رابطه‌ی ۴.۲۳ را محاسبه می‌کنیم. از آنجایی که $o_j = \sigma(net_j)$، مشتق $\frac{\partial o_j}{\partial net_j}$ فقط مشتق تابع سیگموید به ازای net_j خواهد بود ($\sigma(net_j)(1-\sigma(net_j))$). بنابراین:

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1-o_j) \tag{4.25}$$

با توجه به روابط ۴.۲۴ ، ۴.۲۵ و ۴.۲۳ داریم:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1-o_j) \tag{4.26}$$

با ترکیب این رابطه با روابط ۴.۲۱ و ۴.۲۲ قانون شیب نزول تصادفی برای واحد های خروجی بدست می‌آید.

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \mathrm{w_{ji}}} = \eta(t_j - o_j)o_j(1-o_j)x_{ji} \tag{4.27}$$

توجه داشته باشید که این قانون تغییر وزن‌ها معادل رابطه های (T4.3) و (T4.5) در جدول ۴.۲ است. علاوه بر این حال معلوم شد که مقدار δ_k در رابطه‌ی (T4.3) مساوی کمیت $-\frac{\partial E_d}{\partial net_k}$ است. در ادامه‌ی این قسمت از δ_i به جای $-\frac{\partial E_d}{\partial net_i}$ استفاده می کنیم.

حالت دوم: قانون آموزش برای واحد های پنهان. در این حالت واحد j واحدی پنهان یا داخلی است، در مشتق گیری از قانون آموزش برای w_{ji} باید توجه داشت که به طور غیر مستقیم w_{ji} بر خروجی شبکه و متعاقباً خطای E_d تاثیر خواهد داشت. به همین دلیل، بد نیست که به تمامی واحدهایی که مستقیماً از j ورودی دریافت می‌کنند اسمی اطلاق کنیم. این دسته از واحدها را با Downstream(j) نماد گذاری می‌کنیم. توجه داشته باشید که net_j فقط از طریق Downstream(j) می‌تواند بر روی خروجی‌ها و متعاقباً E_d تاثیر بگذارد. بنابراین داریم:

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j) \qquad (4.28)$$

با بازنویسی‌ای رابطه و استفاده از δ_j به جای $-\frac{\partial E_d}{\partial net_j}$ داریم:

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

و

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

که دقیقاً همان قانون کلی‌ای است که در رابطه‌ی (4.20) آمده. از این رابطه می‌توان برای آموزش تمامی واحدهای پنهان در شبکه های بدون دور دلخواه استفاده کرد. توجه داشته باشید که رابطه‌ی (T4.4) در جدول 4.2 حالت خاصی از همین قانون است که Downsteam(j)=outputs.

٤.٦ نکاتی در مورد الگوریتم Backpropagation

٤.٦.١ همگرایی و مینمم نسبی

همان طور که در بالا نیز گفته شد، الگوریتم Backpropagation از شیب نزول برای جستجوی فضای وزن‌های ممکن استفاده می‌کند و در هر بار اجرای حلقه مقدار خطای E (اختلاف بین تابع هدف و خروجی) را کمتر می‌کند. چون سطح خطا برای شبکه های چند لایه ممکن است چندین مینیمم نسبی داشته باشد، این امکان وجود دارد که شیب نزول در یکی از این مینمم های نسبی به دام بیفتد و هیچ تضمینی نیست که این مینیمم نسبی همان مینیمم مطلق برای E باشد.

بر خلاف این ضعف الگوریتم Backpropagation، این الگوریتم در عمل متد تقریبی بسیار مفیدی است. در بسیار از کاربردهای واقعی مشکل مینمم های نصبی به اندازه ای که گفته شد شدید نیست. برای درک مستقیم، شبکه‌هایی با تعداد زیادی از وزن‌ها در فضای خطایی با بعد زیاد را در نظر بگیرید (به ازای هر وزن یک بعد اضافه می‌شود). زمانی که شیب نزول داخل یکی از این مینمم های نسبی می‌افتد، مینیمم یکی از وزن‌هاست، برای دیگر وزن‌ها داخل مینیمم نسبی نخواهد بود. در واقع، زمانی که تعداد وزن‌ها در شبکه افزایش می‌یابد متناسباً تعداد وزن‌ها و بعدها نیز افزایش یافته و امکان وجود راه فرار از مینیمم نسبی نیز افزایش می‌یابد.

مینیمم‌های نسبی جنبه‌ی دیگری نیز دارند و این تأثیر هنگامی که تعداد تکرار های حلقه‌ی الگوریتم افزایش می‌یابد ظاهر می‌شود. توجه دارید که اگر برای مقدار دهی اولیه وزن‌ها مقدار صفر را انتخاب کنیم، شیب نزول در قدم‌های اول اجرایش به تابعی بسیار هموار همگرا می‌شود که تقریباً خطی است. دلیل این امر این است که تابع سیگموید نیز در زمانی که وزن‌ها نزدیک به صفرند تقریبی خطی است (شکل تابع سیگموید در شکل ۴.۶ آمده است). فقط هنگامی که وزن‌ها زمان کافی برای به اندازه ای کافی بزرگ شدن را داشته باشند می‌توانند به نقطه ای برسند که توابع شبکه غیر خطی را نیز تقلید کنند. می‌توانیم تصور کنیم که زمانی که تعداد مینیمم‌های نسبی زیادی در فضای وزن‌ها زیاد است شبکه

توابع پیچیده تری را می‌تواند تقلید کند. و می‌توان امید داشت که زمانی که وزن‌ها به چنین نقاطی می‌رسند، به اندازه‌ی کافی به مینیمم مطلق نزدیک شده‌ایم.

بر خلاف آنچه در بالا گفته شد، شیب نزول برای سطوح خطای پیچیده تر قابل درک نیست و هیچ متدی وجود ندارد که با اطمینان مواردی که مینیمم مطلق مشکل ساز می‌شود را مشخص کند. ایده‌هایی که برای حل مشکل مینیمم نسبی ارائه شده به شرح زیر است:

- اضافه کردن جمله‌ی تکانه به رابطه‌ی تغییر وزن‌ها (استفاده از معادله‌ی 4.18). در بعضی موارد استفاده از این روش می‌تواند شیب نزول را از یک مینیمم موضعی به مینیمم مطلق ببرد (و در بعضی موارد نیز برعکس می‌تواند ما را از مینیمم مطلق به مینیمم موضعی بکشاند)
- استفاده از شیب نزول تصادفی به جای خود شیب نزول. همان طور که در بخش ۴.۴.۳.۳ نیز گفته شده شیب نزول تصادفی تخمینی مفید از شیب نزول دارد که خطای دیگری را برای هر نمونه کم می‌کند، و با توجه به میانگین این خطاها و کل نمونه‌ها گرادیان را تخمین می‌زند. این سطوح مختلف خطا معمولاً مینیمم‌های نسبی مختلفی دارند و معمولاً الگوریتم در آن‌ها به دام نمی‌افتد.
- آموزش چندین شبکه با نمونه های آموزشی یکسان، مقادیر مختلف تصادفی وزن‌ها در ابتدای هر آموزش. اگر این چند آموزش مختلف به چند مینیمم موضعی مختلف در خطاها برسد، آنگاه می‌توان شبکه ای را انتخاب کرد که مینمم موضعی کمتری دارد. متناوباً، می‌توان تمامی شبکه‌ها را دوباره آموزش داد و به عنوان "کمیته[1]" یا مجموعه ای از شبکه‌ها که خروجی آن‌ها متوسط خروجی شبکه های نظیر است استفاده کرد.

٤.٦.٢ معرفی قدرت شبکه های تک سویه

چه توابعی را می‌توان به شبکه های تک سویه آموزش داد؟ البته جواب این سؤال به عمق[2] و پهنای[3] شبکه وابسته است. با وجود اینکه هنوز هیچ اطلاعاتی کامل در مورد اینکه چه دسته توابعی را می‌توان به چه شبکه‌هایی آموزش داد در دسترس نیست، اما سه دسته تابع بخصوص را می‌توان به این نوع شبکه‌ها آموزش داد:

- توابع منطقی. هر تابع منطقی را می‌توان با شبکه های دو لایه یاد گرفت، اما با در بدترین حالت[4] این وجود تعداد گره های پنهان با افزایش ورودی‌های به صورت نمایی بالا می‌رود. برای معلوم شدن این توانایی، تابع منطقی دلخواهی را در نظر بگیرید، به ازای هر بردار بخصوص از ورودی‌ها واحد پنهانی را در نظر بگیرید که وزن‌هایش به شکلی هستند که تنها با آن ورودی بخصوص تهییج می‌شود. با چنین ساختار و آموزشی، شبکه ای به وجود می‌آید که در لایه‌ی پنهان آن همیشه یک واحد فعال است. حال با استفاده از واحد های OR برای خروجی شبکه ای به سازید که به ازای مقادیر مختلف لایه‌ی پنهان خروجی متناسب را بدهد.
- توابع پیوسته. هر تابع کران دار پیوسته را می‌توان با مقداری خطای دلخواه (کمتر از حد دلخواه خاصی) با شبکه‌ی دو لایه یاد گرفت (Cybenko 1989;Hornic et al 1989). چنین شبکه‌هایی در لایه‌ی پنهان واحد سیگموید و در لایه‌ی خروجی واحد خطی (بدون مقدار آستانه) دارند. تعداد واحد های پنهان لازم به تابع بستگی دارد.

---

[1] comettee
[2] depth
[3] width
[4] worse case

- توابع دلخواه. هر تابع دلخواهی را می‌توان با دقت دلخواه توسط شبکه ای با ۳ لایه یاد گرفت (Cybenko 1988). باز هم واحدهای خروجی واحد خطی و واحد های لایه های پنهان واحد سیگموید هستند. باز هم در حالت کلی معلوم نیست که هر لایه چند واحد نیاز دارد. اثبات این قضیه با استفاده از این است که نشان می‌دهند هر تابعی را می‌توان با ترکیب توابع خطی‌ای که فقط در محدوده ای غیر صفرند نشان داد. در ادامه‌ی اثبات ثابت می‌کنند که دو لایه واحد سیگموید کافی است تا تقریب خطی را برای هر محدوده‌ی کوچکی نشان دهند.

این نتایج به دست آمده نشان می‌دهد که شبکه های تک سویه با عمق محدود فضای فرضیه ای شاملی برای الگوریتم Backpropagation ایجاد می‌کنند. با این وجود بد نیست همیشه در نظر داشته باشیم که بردار وزن‌هایی که از طریق الگوریتم شیب نزول از مقدار دهی اولیه بدست می‌آیند همیشه تمامی بردار وزن‌های ممکن را در بر ندارند. در کتاب (Hertz et al. 1991) درباره‌ی موارد بالا بیشتر بحث شده است.

٤.٦.٣ جستجو در فضای فرضیه‌ها و بایاس استقرایی

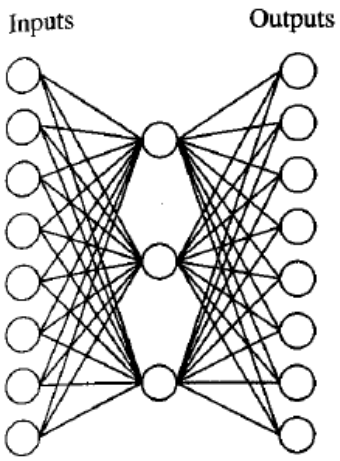
بد نیست که فضای فرضیه ای حاصل از جستجوی الگوریتم Backpropagation را با جستجوی دیگر الگوریتم‌ها مقایسه کنیم. در Backpropagation هر بردار وزن‌ها یک فرضیه را تشکیل می‌دهد که ممکن است توسط یادگیر یاد گرفته شود. به عبارت دیگر، فضای فرضیه ای فضایی n بعدی اقلیدسی است که بر پایه‌ی n بردار پایه ایجاد می‌شود. توجه دارید که این فضای فرضیه ای پیوسته است اما در مقابل فضای فرضیه ای درخت یادگیری و بقیه‌ی متدها گسسته هستند. با توجه به پیوستگی فضای فرضیه ای و اینکه E مشتق پذیر است (زیرا که متغیرهایش پیوسته‌اند)، پس خطای تعریف شده گرادیان دارد که همین گرادیان کمک بسیار بزرگی در سازماندهی جستجو می‌کند. این ساختار سازماندهی با ترتیب کلی‌تری (که در یادگیری مفهوم نمادین بود) و ترتیب ساده به پیچیده (که در درخت تصمیم گیری الگوریتم‌های ID3 و C4.5 بود) بسیار متفاوت است.

بایاس استقرایی‌ای که Backpropagation برای استقرا روی نمونه های آموزشی فرض می‌کند چیست؟ دقیقاً مشخص کردن بایاس استقرایی Backpropagation مشکل است زیرا که به اثر متقابل بین جستجوی شیب نزول و نحوه ای که فضای وزن‌ها فضای توابع قابل نمایش را پوشش می‌دهد بستگی دارد. با این وجود می‌توان این بایاس استقرایی را درون یابی بین نقاط داده‌ها دانست. مثلاً با داشتن دو نمونه مثبتی که هیچ نمونه منفی‌ای بین آن‌ها نیست، Backpropagation تمایل دارد که نقاط میانی این دو نقطه را نیز مثبت دسته بندی کند. چنین رفتاری را می‌توان در سطح تصمیم گیری‌ای که در شکل ۴.۵ آمده دید، در این شکل نمونه های آموزشی منطقه های تصمیم گیری را معلوم می‌کنند.

٤.٦.٤ معرفی لایه‌ی پنهان

یکی از ویژگی‌های خاص Backpropagation این است که در لایه‌ی پنهان در داخل شبکه مقادیر مفیدی را نمایش می‌دهد. زیرا که نمونه های آموزشی فقط مقادیر ورودی و خروجی را در خود دارند و فرایند تغییر وزن‌ها آزاد است که مقادیر لایه‌ی پنهان را به دلخواه تغییر دهد تا خطا را مینیمم کند. همین آزادی باعث می‌شود در لایه های پنهان مقادیری را پیدا کند که صریحاً در نمونه‌ها بیان نشده اما ویژگی‌هایی را بیان می‌کنند که بیشترین تأثیر را در یادگیری تابع هدف دارند.

برای مثال شبکه‌ی شکل ۴.۷ را در نظر بگیرید، در این شبکه ۸ ورودی به ۳ واحد لایه‌ی پنهان وصل شده‌اند و این ۳ واحد نیز به ۸ واحد خروجی متصل هستند. بخاطر این ساختار، سه واحد لایه‌ی پنهان لازم است به صورتی مقادیر ۸ ورودی را با ویژگی‌های مرتبطی بیان کنند تا در انتها بتوانند همان مقادیر را به عنوان خروجی بدهند.



Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

شکل ۴.۷ مقادیر لایه‌ی پنهان برای نمونه های آموزشی.
این شبکه‌ی 8x3x8 با ۸ نمونه که در شکل است برای یادگیری تابع همانی آموزش داده شده است. بعد از ۵۰۰۰ بار اجرای حلقه مقادیر ۳ واحد پنهان مقادیر ورودی را به درستی کد می‌کنند. توجه داشته باشید که مقادیر کد شده را به صفر و یک گرد کنیم نتیجه کد باینری برای هشت ورودی خواهد بود.

شبکه‌ی شکل ۴.۷ برای یادگیری تابع هدف بسیار ساده‌ی $f(\vec{x}) = \vec{x}$ که در آن $\vec{x}$ برداری با هشت صفر و یک یک است در نظر بگیرید. شبکه باید یاد بگیرد تا ۸ ورودی را دوباره ایجاد کند. با اینکه این تابع هدف بسیار ساده است اما ۳ واحد پنهان برای یادگیری این تابع بسیار کم است. در این مثال شبکه مجبور است مهم‌ترین اطلاعات لازم را از طریق این سه واحد به سمت خروجی‌های انتقال دهد.

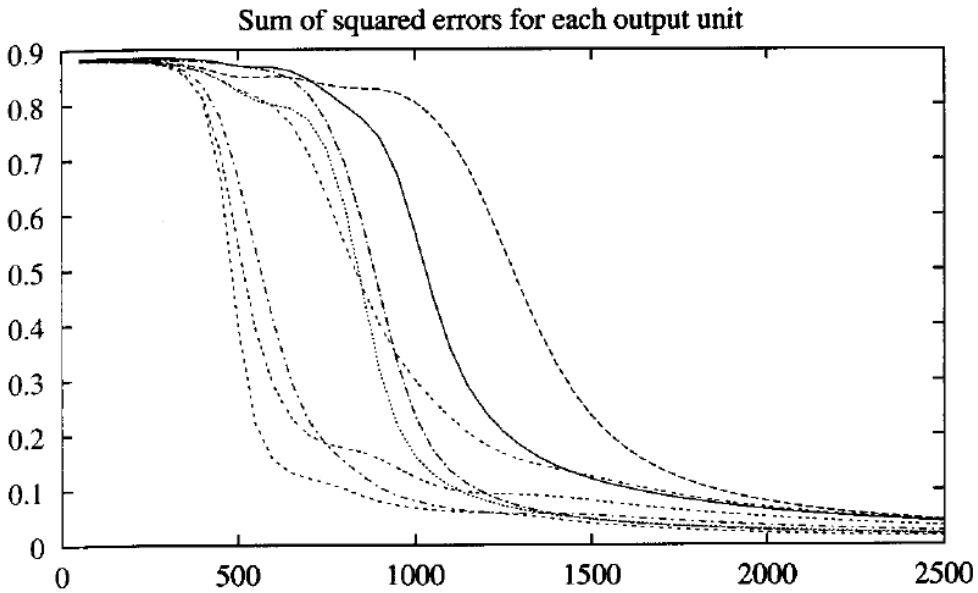
با استفاده از الگوریتم Backpropagation و نمونه های آموزشی مشخص شده در شکل این تابع هدف به شبکه یاد داده شده است. چه نمایشی از ورودی‌ها توسط شیب نزول در لایه‌ی پنهان نمایش داده می‌شود؟ با بررسی بیشتر مشخص می‌شود که این مقادیر که در لایه‌ی پنهان ظاهر می‌شود همان کد آشنای باینری برای هشت عدد است که با ۳ بیت نمایش داده می‌شود (۰۰۰، ۰۰۱، ۰۱۰، ...، ۱۱۱). مقدار دقیق این مقادیر در شکل ۴.۷ آورده شده است.

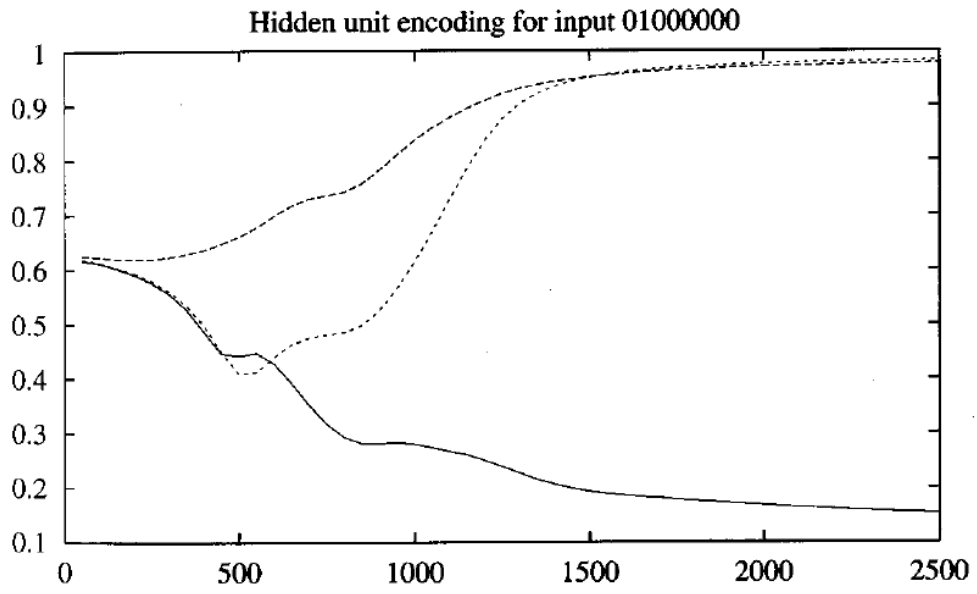
این قابلیت شبکه های عصبی در پیدا کردن نمایش‌های خاص در لایه های پنهان منحصر به فرد است. بر خلاف دیگر متد های یادگیری که فقط مواردی را که طراح انسانی در نظر گرفته را در نظر می‌گیرند، این خاصیت به شبکه های عصبی این قابلیت را می‌دهد کاملاً انعطاف پذیر باشند و ویژگی‌هایی را استخراج کنند که طراح انسانی در نظر نگرفته است. البته بدیهی است که تمامی این ویژگی‌های استخراج شده باید از ورودی‌ها توسط واحد های سیگموید قابل استخراج باشند. توجه دارید که زمانی که لایه های بیشتری در شبکه وجود دارند خواص پیچیده تری قابل استخراجند. مثال دیگری از خواص لایه‌ی پنهان در قسمت ۴.۷، کاربرد در تشخیص چهره[1]، آورده شده است.

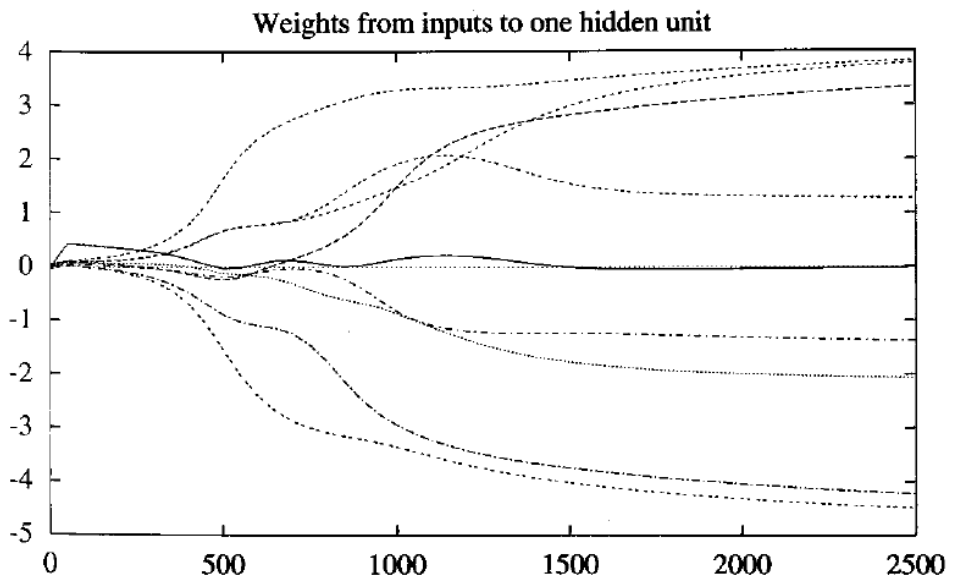
[1] face recognition

برای درک بهتر از عملیات Backpropagation بیایید در این مثال عملیات فرایند شیب نزول را دقیق‌تر بررسی کنیم (کد استفاده شده در این مثال در آدرس http://www.cs.cmu.edu/~tom/mlbook.html آمده است). شبکه‌ی شکل ۴.۷ با الگوریتم جدول ۴.۲ آموزش داده شده است، در مقدار دهی اولیه‌ی وزن‌ها از اعداد بازه‌ی (-0.1,0.1) استفاده شده است و ضریب آموزشی نیز η=0.3 بوده و تکانه هم استفاده نشده است (α=0). یادگیری با استفاده از ضرایب آموزشی دیگر و تکانه نیز به همین نتایج رسیده است. مقادیر لایه‌ی پنهان (که در شکل ۴.۷) آمده بعد از ۵۰۰۰ بار اجرای حلقه اصلی الگوریتم (یعنی ۵۰۰۰ بار تکرار هر نمونه آموزشی) بدست آمده است. البته اکثر وزن‌ها در ۲۵۰۰ اجرای اول مشخص شده بودند.

با کشیدن نمودار خطا بر حسب تعداد گامی که شیب نزول برداشته، می‌توان تلاش شیب نزول را برای کاهش خطا دید. این نمودار در شکل ۴.۸ کشیده شده است. هر خط در این نمودار نشان دهنده‌ی مجموع خطاها برای تمامی نمونه های آموزشی در یکی از خروجی‌هاست. محور افقی تعداد تکرار های حلقه‌ی اصلی الگوریتم را نشان می‌دهد. همان طور که نمودار نیز گویای مطلب است با ادامه‌ی کار شیب نزول مجموع خطای خروجی برای خروجی‌ها کاهش پیدا می‌کند، ممکن است این کاهش در بعضی خروجی‌ها شدید تر و در بعضی دیگر ملایم‌تر باشد.







شکل ۴.۸ یادگیری شبکه‌ی 8x3x8.

نمودار اول مجموع خطاها را برای هر یک از ۸ خروجی را بر حسب تعداد تکرار حلقه‌ی اصلی الگوریتم نشان می‌دهد. نمودار دوم مقادیر لایه‌ی پنهان را برای ورودی ی "01000000" نشان می‌دهد. و نمودار آخر وزن‌ها را برای یکی از سه واحد پنهان نشان می‌دهد.

سیر تکامل لایه‌ی پنهان در نمودار دوم شکل ۴.۸ دیده می‌شود. این نمودار مقدار سه واحد لایه‌ی پنهان را که در هر مرحله محاسبه می‌شود برای یکی از ورودی‌ها ("01000000") نشان می‌دهد. مثل نمودار اول محور افقی تعداد تکرار های حلقه‌ی اصلی الگوریتم را نشان می‌دهد. همان طور که شکل نیز گویاست قبل از اینکه شبکه به نحوه‌ی کد سازی آخری برسد تعدادی از کد سازی‌های ممکن را برای ورودی امتحان کرده است.

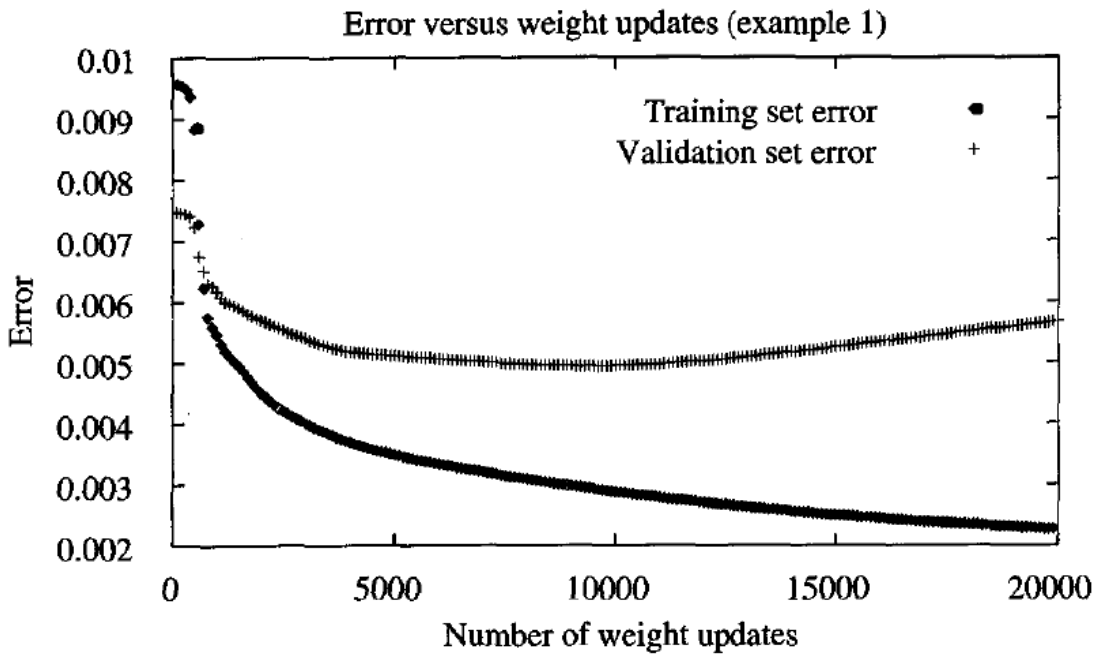
و بالاخره نمودار آخر شکل ۴.۸ سیر تکامل وزن‌های شبکه را نشان می‌دهد. این نمودار تکامل وزن‌های ارتباط دهنده‌ی بین هشت واحد ورودی (و مقدار ثابت ۱ برای مقدار آستانه) و یکی از واحد های لایه‌ی پنهان نشان می‌دهد. توجه داشته باشید که تغییرات قابل توجه در مقدار وزن‌ها

همزمان با تغییرات قابل توجه در میزان خطا و مقدار لایه‌ی پنهان است. وزنی که به مقداری نزدیک صفر میل می‌کند همان وزنی است که برای مقدار آستانه در نظر گرفته شده (w_0).

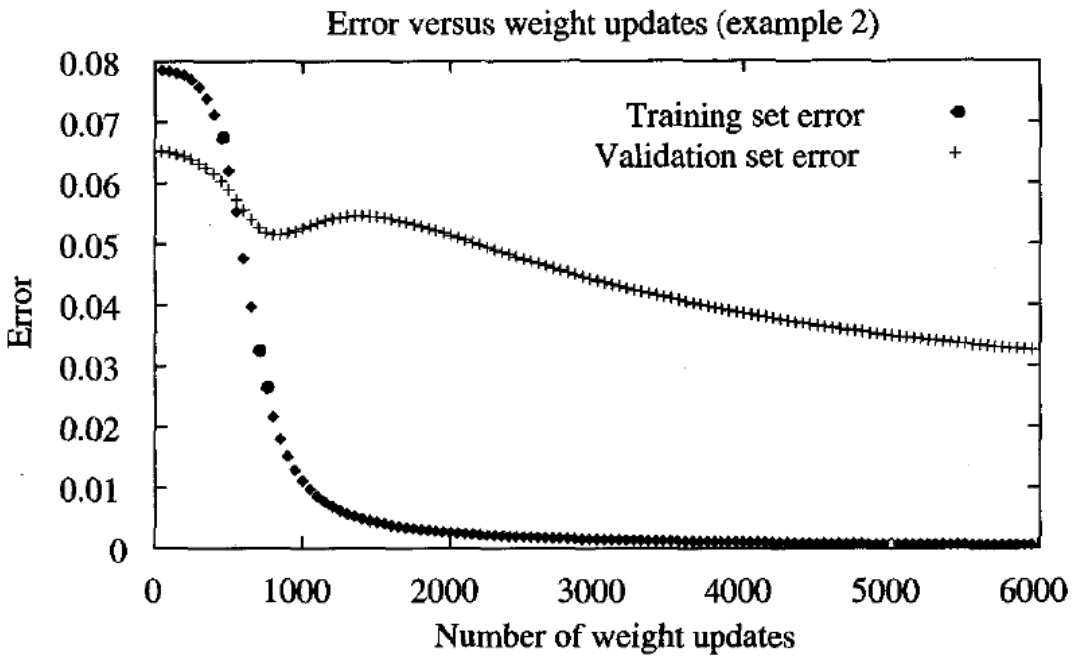
٤.٦.٥ معیارهای تعمیم، overfit و توقف

در جدول ۴.۲ از شرط پایانی[1] صحبت شد اما معلوم نشد که این شرط دقیقاً چیست. شرط مناسب برای پایان حلقه‌ی تغییر وزن‌ها چیست؟ یک شرط بسیار ساده ادامه دادن مراحل تا زمانی که خطا (E) برای نمونه های آموزشی کمتر از مقدار خاصی بشود است. در واقع این استراتژی برای پایان بسیار ضعیف است زیرا که Backpropagation مستعد است تا برای کم کردن خطا قدرت تعمیم شبکه را برای نمونه های جدید کم کند.

برای روشن شدن خطر کم کردن افراطی خطا برای نمونه های آموزشی، توجه کنید که با افزایش تعداد تکرار های حلقه‌ی اصلی الگوریتم خطای E چگونه کاهش می‌یابد. شکل ۴.۹ این تفاوت را برای استفاده از Backpropagation را برای دو مثال کاربردی نشان می‌دهد. نمودار اول را در نظر بگیرید. منحنی پایین‌تر در نمودار خطای E را برای نمونه های آموزشی نشان می‌دهد که با افزایش تعداد تکرار های الگوریتم کاهش می‌یابد. منحنی بالایی خطای E را برای مجموعه‌ی تایید نشان می‌دهد که از نمونه های آموزشی مجزاست. این منحنی قدرت تعمیم شبکه[2] را برای نمونه های جدید نشان می‌دهد.



[1] termination condition
[2] generalization accuracy



شکل ۴.۹ نمودار خطای E به عنوان تابعی از تعداد تکرار تغییر وزن‌ها برای دو درک متفاوت یک ربات.
در هر دو حالت خطای E در طی تکرار های بیشتر کمتر می‌شود. خطای مربوط به مجموعه‌ی تایید[1] معمولاً در ابتدا کاهش پیدا می‌کند و سپس بعد از افزایش تعداد های تکرار بر اثر overfit احتمال دارد افزایش یابد. شبکه ای که کمترین خطا برای مجموعه‌ی تایید را داشته باشد مناسب‌ترین شبکه برای تعمیم روی نمونه های جدید است. توجه داشته باشید که در نمودار دوم به محض افزایش جزئی مقدار خطای مجموعه‌ی تایید نباید اجرای حلقه متوقف شود.

توجه داشته باشید که قدرت تعمیم شبکه که از مجموعه‌ی تایید محاسبه می‌شود ابتدا کاهش می‌یابد و سپس افزایش می‌یابد، حتی اگر خطای نمونه های آموزشی همچنان کاهش بیابد. دلیل این اتفاق چیست؟ دلیل این اتفاق این است که وزن‌ها بعد از آن طوری تغییر می‌کنند که منحصراً با نمونه های آموزشی مطابق باشند و خاصیت تعمیمی خود را از دست می‌دهند. تعداد زیاد پارامترهای شبکه عصبی درجه آزادی‌های زیادی برای الگوریتم باقی می‌گذارد تا بتواند شبکه را منحصراً با نمونه های آموزشی مطابقت دهد (overfit).

چرا همیشه overfit در تکرار های آخر الگوریتم اتفاق می‌افتد و در تکرار های ابتدایی هیچ اثری از overfit نیست؟ فرض کنید که وزن‌ها را با مقادیر کوچک تصادفی مقدار دهی اولیه کرده‌ایم. چون وزن‌ها تقریباً یکی هستند، سطح تصمیم گیری بسیار هموار خواهد بود. در ادامه برای کم کردن میزان خطای نمونه های آموزشی بعضی از وزن‌ها افزایش می‌یابد و پیچیدگی بیشتری به سطح تصمیم گیری می‌دهند. در ادامه با افزایش تعداد تکرارها بر پیچیدگی فرضیه‌هایی که به آن می‌رسیم بالا می‌رود، (پیچیدگی در این مراحل مفید است). با ادامه‌ی این تکرارها به اندازه‌ی کافی می‌توان به سطوح تصمیم گیری پیچیده تری رسید که هم نویز نمونه های آموزشی را بر طرف می‌کند و هم ویژگی‌های غیر مربوطه به تابع هدف نمونه های آموزشی را بیان می‌کند. مشکل overfit در شبکه های عصبی درست مشابه همین مشکل در درخت تصمیم گیری است (فصل ۳).

تکنیک‌های بسیاری برای حل مشکل overfit در Backpropagation وجود دارد. یکی از این تکنیک‌ها weight decay نام دارد. در این متد در هر حلقه مقداری از هر وزن کم می‌شود. این درست مشابه این است که در تعریف E جمله ای را اضافه کنیم تا برخلاف آن عمل

[1] validation set

کند و مانع overfit شود. به این جمله، جمله‌ی جریمه[1] می‌گویند. هدف از این متد این است که مقدار وزن‌ها را کوچک نگه داریم تا بایاسی ایجاد کرده باشیم تا سطح تصمیم گیری بسیار پیچیده نشود.

یکی از موفق‌ترین متدها برای حل مشکل overfit، استفاده از دسته‌ی تایید به همراه نمونه های آموزشی برای کنترل جستجوی شیب نزول است. الگوریتم می‌توان از خطای این مجموعه‌ی تایید برای کنترل شیب نزول استفاده کند. از این نظر، این روش به الگوریتم اجازه می‌دهد که دو منحنی نشان داده شده در شکل ۴.۹ را در دسترس داشته باشد. اما حلقه‌ی تغییر وزن‌های الگوریتم چند بار باید اجرا شود؟ واضح است که حلقه باید به تعدادی اجرا شود که خطا روی دسته‌ی تایید مینیمم شود. در کاربرد های معمول این روش دو نسخه از وزن‌های شبکه ذخیره می‌شود: یک کپی برای آموزش و کپی‌ای از بهترین وزن‌ها بدست آمده تا این مرحله بر اساس خطا بر روی مجموعه‌ی تایید. زمانی که وزن‌های به دست آمده به مقدار خطای قابل توجهی بیشتر بر روی مجموعه‌ی تایید می‌رسد، آموزش پایان می‌یابد و وزن‌های ذخیره شده به عنوان فرضیه‌ی نهایی خروجی داده می‌شود. زمانی که از این فرایند برای حالت شکل ۴.۹ استفاده شد بعد از ۹۴۰۰ بار تکرار الگوریتم به وزن‌های خروجی رسید. شکل دوم در شکل ۴.۹ نشان می‌دهد که همیشه رسیدن به کمترین مقدار خطای دسته‌ی تایید را نمی‌توان به راحتی تعیین کرد. در این شکل ابتدا خطا بر روی دسته‌ی تایید کاهش، سپس افزایش و دوباره کاهش می‌یابد. باید دقت کافی را در نتیجه گیری رسیدن خطای تایید به مینیمم خود مبذول داشت، این شبکه در تکرار ۸۵۰ به مینیمم خطای تایید می‌رسد.

در کل، مشکل overfit و چگونگی حل آن نیاز به دقت زیادی دارد. روش cross-validation بالا زمانی کارایی دارد که مقدار زیادی داده در دسترس باشد تا بتوان دسته‌ی تایید تشکیل داد. با این وجود، مشکل overfit در مجموعه های آموزشی کوچک‌تر شدید تر است. در چنین شرایطی، گاهی از روش k-fold cross-validation استفاده می‌شود، در این روش cross-validation، k بار متفاوت و هر بار با قسمت متفاوتی از داده‌ها به عنوان مجموعه‌ی آموزشی و دسته‌ی تایید انجام می‌شود و نتیجه میانگین نتایج خواهد بود. در یکی از نسخه های این روش، m نمونه‌ی موجود به k زیر مجموعه‌ی مجزا با اندازه های m/k تقسیم می‌شوند. فرایند cross-validation، k بار و با استفاده از یکی از این k زیرمجموعه به عنوان دسته‌ی تایید و بقیه‌ی نمونه‌ها به عنوان مجموعه‌ی آموزشی اجرا می‌شود. بنابراین، هر نمونه در یک آزمایش در دسته‌ی تایید و در k-1 آزمایش در دسته‌ی آموزشی خواهد بود. در هر یک از آزمایش‌های روش cross-validation بالا برای تعیین تعداد تکرارها، i، مورد استفاده قرار می‌گیرد، i شماره‌ی تکراری است که در آن دسته‌ی تایید بهترین خطا را داشته است. میانگین i برای این مقادیر محاسبه می‌شود، در انتها نیز از backpropagation برای آموزش شبکه بر روی تمامی n نمونه بدون دسته‌ی تایید با تعداد تکرار میانگین i ها استفاده می‌شود. این فرایند بسیار مشابه فرایند مقایسه‌ی دو متد یادگیری با داده های محدود در فصل ۵ است.

۴.۷ یک مثال: تشخیص چهره[2]

برای درک انتخاب‌های طراحی در نظر گرفته شده در اعمال الگوریتم Backpropagation، در این بخش استفاده از آن را در عمل یادگیری تشخیص چهره بررسی می‌کنیم. تمامی داده‌ها و اطلاعات مربوطه‌ی این قسمت در http://www.cs.cmu.edu/~tom/mlbook.html، به همراه نکات مربوطه‌ی استفاده از این کدها موجود می‌باشد.

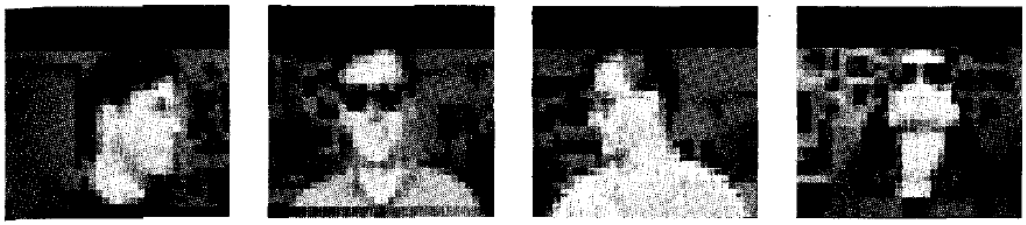
[1] penalty term
[2] face recognition

Telegram: @Farabar_BI
www.farabar.net

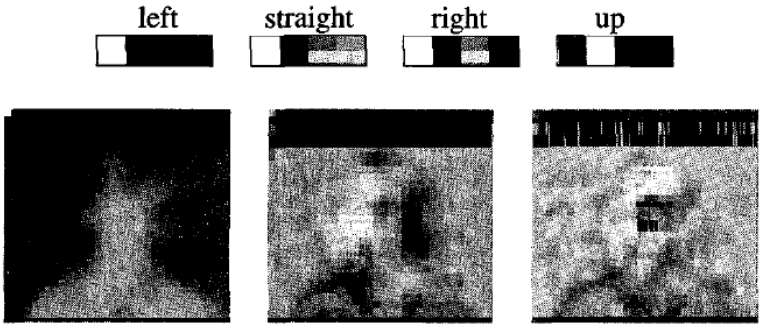
۴.۷.۱ تعریف مسئله

مسئله‌ی یادگیری در اینجا شامل دسته بندی تصاویر دوربین از چهره‌ی افراد مختلف در زاویه های مختلف است. تصاویر ۲۰ نفر، به طور تقریبی ۳۲ تصویر از هر نفر با حالات مختلف چهره (شاد، غمگین، خشمگین، معمولی) و با زاویه های مختلف (چپ، راست، تمام رخ، بالا)، جمع آوری شده است. در شکل ۴.۱۰ نمونه ای از این تصاویر را می‌بینید، همچنین در این تصاویر مکان چهره‌ی شخص، پس زمینه، لباس افراد ثابت نیستند. در مجموع ۶۲۴ تصویر سیاه و سفید با دقت ۱۲۰×۱۲۸ با دقت رنگ ۰ (سیاه) تا ۲۵۵ (سفید) جمع آوری شد.

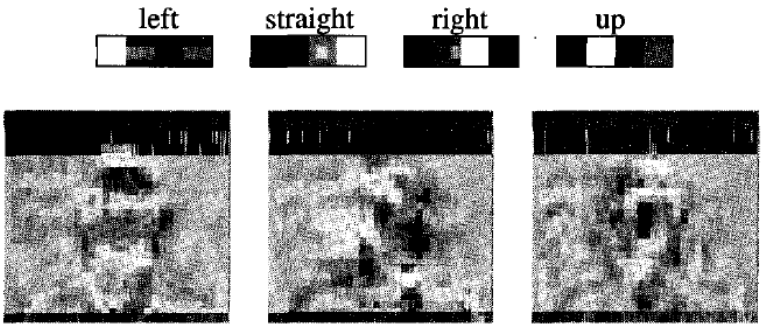
توابع هدف مختلفی را می‌توان از این مجموعه از داده های تصویری یاد گرفت. برای مثال، قرار دادن مجموعه تصاویر در ورودی شبکه می‌توان هدف را تشخیص هویت شخص، جهت صورت وی، جنسیت شخص، عینک دودی زدن وی و ... قرار داد. تمامی این توابع هدف را می‌توان با دقت بالا از این مجموعه داده یاد گرفت و شدیداً توصیه می‌شود که یادگیری این توابع هدف را خود خواننده امتحان کند. در ادامه‌ی این بخش ما به یک هدف یادگیری بخصوص می‌پردازیم: سمت صورت شخص (شامل چپ، راست، تمام رخ و بالا).



30 × 32 resolution input images



Network weights after 1 iteration through each training example



Network weights after 100 iterations through each training example

شکل ۴.۱۰ یادگیری یک شبکه‌ی عصبی مصنوعی برای تشخیص سمت صورت.

www.farabar.net

در اینجا شبکه ای با ابعاد ۹۶۰×۳×۴ بر روی لایه‌ی خاکستری تصویر چهره‌ی افراد آموزش داده می‌شود تا سمت چهره‌ی شخص را تشخیص دهد (چپ، راست، تمام رخ، بالا). بعد از یادگیری روی ۲۶۰ تصویر چهره، شبکه به دقت ۹۰٪ بر روی دسته‌ی تست مجزایی رسید. وزن‌های شبکه بعد از یک و صد بار تکرار حلقه‌ی یادگیری در بالا نشان داده شده‌اند. هر واحد خروجی (چپ، راست، تمام رخ، بالا) چهار وزن دارد که وزن‌هایشان با مربع‌های تیره (منفی) و روشن (مثبت) نشان داده شده است. چپ‌ترین مربع مربوط به وزن w_0 است که مقدار آستانه‌ی واحد را نشان داده و سه مربع دیگر وزن‌های ورودی واحد را از واحدهای پنهان نشان می‌دهد. وزن‌های مربوطه‌ی ورودی هر یک از سه واحد پنهان از نقاط نیز به طور نقطه ای در مکان خودشان نشان داده شده‌اند.

٤.٧.٢ انتخاب‌های طراحی

در اعمال الگوریتم Backpropagation به هر مسئله تعدادی انتخاب طراحی انجام می‌شود. این انتخاب‌های طراحی را برای مسئله‌ی مطرح یادگیری سمت چهره به صورت زیر خلاصه می‌کنیم. با وجود اینکه تلاشی برای طراحی بهینه در این مسئله انجام نمی‌شود، طراحی مطرح شده با دقت قابل توجهی تابع هدف را یاد می‌گیرد. بعد از یادگیری بر روی دسته ای ۲۶۰ عضوی از تصاویر، دقت دسته بندی بر روی دسته‌ی تست مجزا ۹۰٪ است. در حالی که، معمولاً دقت بدست آمده در حدس یکی از این چهار حالت ۲۵٪ است.

کد گذاری ورودی. با معلوم بودن اینکه ورودی شبکه باید نمایشی از تصویر باشد یکی از انتخاب‌های کلیدی تصمیم گیری نوع کد گذاری ورودی شبکه است. برای مثال می‌توانستیم تصویر را برای تشخیص لبه های رنگ‌ها پیش پردازش کرده یا دیگر خواص ناحیه ای تصویر را استخراج کرده و سپس به عنوان ورودی به شبکه بدهیم. یکی از مشکلات چنین ورودی‌هایی ایجاد تعداد متغیری از ویژگی‌ها (لبه‌ها) است در حالی که شبکه عصبی تعداد معلومی واحد ورودی دارد. انتخاب طراحی این قسمت استفاده از مجموعه نقاط ثابت ۳۰×۳۲ عکس و قرار دادن یک واحد ورودی برای هر نقطه است. بازه‌ی ۰ تا ۲۵۵ شدت رنگ نیز به طور خطی به بازه‌ی ۰ تا ۱ نگاشت شد تا مشابه لایه‌ی پنهان بازه‌ی خروجی واحد بین صفر تا یک باشد. تصویر ۳۰×۳۲ نقطه ای، در حقیقت، یک خلاصه ای از تصویر اصلی ۱۲۰×۱۲۸ نقطه ای موجود است که هر یک از نقاط با میانگین گیری چهار نقطه‌ی متناظر در تصویر با کیفیت بالا تر است. با استفاده از این کاهش کیفیت تعداد ورودی‌های شبکه به تعداد قابل کنترل‌تری تبدیل می‌شود، و بنابراین پیچیدگی محاسباتی نیز در عین حفظ دقت لازم برای دسته بندی درست تصاویر کاهش می‌یابد. با توجه به شکل ۴.۱ سیستم ALVINN نیز از کاهش دقت مشابهی برای ورودی شبکه استفاده می‌کند. یکی از نکات جالب این است که ALVINN بجای استفاده از میانگین نقاط محدوده‌ی مربوطه در تصویر با کیفیت بالا تر فقط نقطه ای تصادفی را در آن محدوده انتخاب کرده و شدت رنگ آن‌را به عنوان شدت رنگ مربوطه در نظر می‌گیرد. انگیزه‌ی این کار در ALVINN این است که محاسبات لازم برای ایجاد تصاویر کم دقت تر به طور قابل توجهی با این روش کاهش می‌یابد. این ویژگی هنگامی که شبکه لازم است تعداد زیادی از تصاویر را در ثانیه پردازش کرده و خودرو را کنترل کند بیشتر قابل توجه می‌شود.

کد گذاری خروجی. شبکه‌ی عصبی می‌بایست یکی از چهار ویژگی مربوطه‌ی جهت صورت شخص (چپ، راست، تمام رخ، بالا) را خروجی دهد. توجه داشته باشید که ما می‌توانستیم این چهار جهت را با یک خروجی و نسبت دادن ۰.۲، ۰.۴، ۰.۶، ۰.۸ برای چهار مقدار مربوطه این مقادیر خروجی را کد سازی کنیم. در مقابل، ما چهار واحد خروجی مجزا برای نمایش هر یک از جهات صورت در نظر گرفته‌ایم و واحدی که بیشترین مقدار را داشته باشد خروجی شبکه در نظر گرفته خواهد شد. این نوع کد گذاری گاهی کد گذاری ۱ از n [1] نیز نامیده می‌شود. دو انگیزه برای انتخاب کد گذاری ۱ از n بجای در نظر گرفتن یک خروجی وجود دارد. ابتدا اینکه درجه‌ی آزادی بیشتری برای شبکه برای نمایش شبکه‌ی هدف باقی خواهد گذاشت (در لایه‌ی خروجی n برابر وزن موجود است). دوم اینکه در کد گذاری ۱ از n تفاوت بزرگ‌ترین مقدار خروجی و دومین (بزرگ‌ترین) مقدار خروجی را می‌توان به عنوان درجه‌ی اطمینان پیش بینی شبکه دانست (دسته بندی‌های مبهم ممکن است

[1] 1-of-n output encoding

به خروجی‌های نزدیک و حتی مساوی بیانجامد). یکی دیگر از انتخاب‌های طراحی این است که مقدار خروجی این چهار واحد چه باید باشند؟ یکی از انتخاب‌های واضح استفاده از چهار مقدار <1,0,0,0> برای کد سازی جهت چپ، <0,1,0,0> برای جهت راست و ... است. در مقابل می‌توان بجای مقادیر ۰ و ۱ از مقادیر ۰.۱ و ۰.۹ استفاده کرد، <0.9,0.1,0.1,0.1> بردار مربوطه‌ی جهت چپ خواهد بود. یکی از دلایل پرهیز از ۰ و ۱ این است که واحد های سیگموید نمی‌توانند این خروجی‌ها را با مقادیر محدود وزن‌ها ایجاد کنند. اگر سعی کنیم شبکه را برای مقادیر ۰ و ۱ آموزش دهیم شیب نزول مجبور به رشد بدون مرز وزن‌ها خواهد بود، در مقابل می‌توان با وزن‌های محدود به مقادیر ۰.۹ و ۰.۱ رسید.

ساختار گراف شبکه. همان طور که قبلاً هم توصیف شد، Backpropagation را می‌توان به هر گراف بدون دور واحد های سیگموید اعمال کرد. بنابراین، گزینه‌ی طراحی دیگری نیز که با آن مواجهیم انتخاب تعداد واحد های شبکه و چگونگی ارتباط بین آن‌هاست. متداول‌ترین ساختار شبکه ساختار لایه ای است که تمامی واحدهای یک لایه به تمامی واحدهای لایه‌ی بعد متصلند. در طراحی فعلی از این ساختار متداول با دو لایه واحد سیگموید (یک لایه‌ی پنهان و یک لایه‌ی خروجی) استفاده کرده‌ایم. استفاده از یک یا دو لایه سیگموید متداول است و در مواقع خاص از سه لایه نیز استفاده می‌کنند. استفاده از تعداد لایه‌ی بیشتر متداول نیست زیرا که آموزش شبکه را بسیار کند می‌کند، همچنین شبکه ای با سه لایه سیگموید می‌تواند انواع بسیار زیادی از توابع را نمایش دهد (به قسمت ۴.۶.۲ مراجعه کنید). اگر بخواهیم از بین شبکه های تک سویه با یک لایه‌ی پنهان استفاده کنیم، مسئله اصلی تعین تعداد واحد های پنهان شبکه خواهد بود. در مثال آورده شده در شکل ۴.۱۰ تنها از سه واحد پنهان استفاده شده است، و مجموعه خروجی دقت ۹۰٪ دارد. در آزمایش‌های دیگر، که از ۳۰ واحد پنهان استفاده شد دقت یک تا دو درصد بالاتر رفت. با وجود اینکه دقت کلی سازی برای این دو آزمایش تنها میزان کمی اختلاف دارد، اما آزمایش دوم به مقدار زمان آموزش قابل توجه بیشتری نیاز داشت. با مجموعه ۲۶۰ عکس آموزشی، بر روی یک سیستم خاص (sun sparc5) شبکه‌ی ۳۰ واحده زمان تقریبی یک ساعت را برای آموزش نیاز داشت در حالی که شبکه ۳ واحده تنها در حدود پنج دقیقه آموزش یافت. در بسیاری از کاربردها، تعداد واحد های پنهان لازم برای یادگیری تابع هدف با دقت خاص ثابت است و واحد های پنهان بیشتر بر روی دقت کلی سازی تأثیری ندارند، از روش‌های (cross-validation) برای تعیین تعداد تکرار لازم برای الگوریتم شیب نزول استفاده می‌شود. اگر از چنین روش‌های استفاده نگردد، در بعضی موارد استفاده از تعداد واحد پنهان بیشتر تمایل شبکه به (overfit) را افزایش داده و دقت کلی سازی را کم می‌کند.

دیگر پارامتر های الگوریتم یادگیری. در این آزمایش‌های یادگیری ضریب یادگیری η مقدار 0.3 و تکانه مقدار 0.3 را داشته است. مقادیر اولیه‌ی کم برای هر دو پارامتر دقت تعمیم نسبتاً مساوی‌ای را نتیجه خواهند داد اما زمان آموزش را افزایش خواهند داد. در مقابل اگر این پارامترها را بزرگ در نظر بگیریم، شبکه به شبکه ای با مقدار خطای قابل قبول روی دسته‌ی آموزشی میل نخواهد کرد. در تمامی این آزمایشات از شیب نزول تنها[1] استفاده شده است (در مقابل تخمین احتمالی شیب نزولی که در جدول ۴.۲ معرفی شد). مقادیر وزن‌های خروجی به مقادیر کوچک نزدیک به صفر مقدار دهی اولیه شده‌اند، اما مقادیر وزن‌های ورودی به صفر مقدار دهی اولیه شده‌اند زیرا که با این روش تصور هوشمندانه تری از وزن‌های آموزشی می‌توان داشت (به شکل ۴.۱۰ رجوع کنید)، این انتخاب در تعمیم تأثیر نخواهد گذاشت. تعداد تکرار آموزش با تقسیم داده های موجود به دسته‌ی آموزشی و دسته‌ی تایید مجزا تعیین شده است. از شیب نزول برای مینیمم کردن خطای مجموعه‌ی آموزشی استفاده شده و بعد از هر ۵۰ تکرار یک بار کارایی شبکه بر روی دسته‌ی تایید بررسی شده است. شبکه‌ی انتخاب شده‌ی نهایی یکی از شبکه‌هایی است که بیشترین دقت را روی دسته‌ی تایید داشته است. برای توجیه و توضیح این روش به قسمت ۴.۶.۵ رجوع کنید. دقت نهایی گزارش شده (۹۰٪ برای شبکه‌ی شکل ۴.۱۰) بر روی دسته‌ی سومی از نمونه های تست انجام شده است که هیچ دخالتی در آموزش نداشته‌اند.

[1] full gradient decend

۴.۷.۳ نمایش پنهان یاد گرفته شده

بررسی مقادیر وزن‌های ۲۸۹۹ ارتباط شبکه از نظر تحلیلی جالب است. شکل ۴.۱۰ مقادیر وزن‌های نظیر این ارتباط‌ها را بعد از یک و صد بار تکرار حلقه‌ی آموزش برای کل تصاویر آموزشی را نشان می‌دهد.

برای درک این نمودارها، ابتدا به مستطیل اول زیر تصویر توجه کنید. هر یک از مستطیل‌ها وزن‌های یکی از چهار واحد خروجی شبکه را نشان می‌دهد (چپ، راست، تمام رخ، بالا). چهار مربع هر یک از این مستطیل‌ها چهار وزن هر واحد خروجی را نشان می‌دهد، وزن w_0 که مقدار آستانه‌ی واحد را مشخص می‌کند در سمت چپ قرار دارد و سه وزن دیگر به ترتیب وزن‌های مربوطه‌ی واحد های پنهان را نشان می‌دهند. شدت رنگ مربع‌ها نشان دهنده‌ی مقدار وزن نظیر است، سفید روشن وزن مثبت بزرگ و سیاه تیره نشان دهنده‌ی وزن منفی بزرگ است، خاکستری نیز مقادیر میانی وزن را نشان می‌دهد. برای مثال، خروجی واحد بالا مقدار وزن آستانه‌ی w_0 نزدیک به صفر، وزن مثبتی برای واحد پنهان اول و وزن بزرگ منفی‌ای برای واحد پنهان دوم دارد.

مقادیر وزن‌های شبکه بعد از ۱۰۰ بار تکرار شیب نزول برای تمامی نمونه های آموزشی در بالای شکل نمایش داده شده‌اند. توجه دارید که چپ‌ترین واحد پنهان وزن‌های بسیار متفاوتی نسبت به وزن‌های پس از یک تکرار دارد، البته دو واحد دیگر نیز تغییر وزن داشته‌اند. درک نسبی این کد گذاری در این مجموعه‌ی محدود از وزن‌ها خیلی هم سخت نیست. برای مثال، فرض کنید واحد خروجی‌ای که جهت صورت راست را مشخص می‌کند را در نظر بگیرید. این واحد وزن مثبت بزرگی از واحد پنهان دوم و وزن منفی بزرگی از واحد پنهان سوم دارد. با بررسی وزن‌های این دو واحد، می‌توان به آسانی فهمید که با چرخش صورت فرد به سمت راست (چپ ما)، صورت روشن فرد به طور تقریبی با وزن‌های مثبت قوی این واحد پنهان هم مکان خواهد شد و موی تیره‌ی فرد به طور تقریبی با وزن‌های منفی هم مکان خواهد شد که باعث بزرگ بودن خروجی این واحد خواهد شد. تصویر مشابهی می‌تواند باعث خروجی دادن واحد پنهان سوم نزدیک به صفر شود، زیرا که صورت روشن شخص با وزن‌های منفی بزرگ هم مکان خواهد شد.

۴.۸ مباحث پیشرفته‌ی شبکه های عصبی مصنوعی

۴.۸.۱ گزینه های مختلف برای تابع خطا

همان طور که قبلاً نیز گفته شد، شیب نزول را می‌توان برای مینیمم کردن هر تابع مشتق پذیر E به کار برد. الگوریتم اصلی Backpropagation خطا را به فرم مجموع مربعات اختلافات با تابع هدف تعریف می‌کند، با این وجود تعریف‌های دیگری برای اعمال شروط دیگر برای این خطا پیشنهاد می‌شود. برای هر تعریف E که به کار می‌بریم برای بدست آوردن گرادیان مشتق بگیریم. در زیر توابع خطای مرسوم آورده شده است:

- اضافه کردن جمله‌ی خطا برای اندازه‌ی وزن‌ها. همان طور که قبلاً نیز توضیح داده شد می‌توان با اضافه کردن جمله‌ی جدیدی به E اضافه کرد که با افزایش بردار وزن‌ها افزایش یابد . این عمل باعث می‌شود تا جستجوی شیب نزول در بین بردارها، بردارهایی که اندازه‌ی کوچک‌تری دارند را ارجح بداند و متناسباً (با در ناحیه‌ی خطی بودن واحد های سیگموید و کم بودن پیچیدگی) خطر overfit کم خواهد شد. پس یکی از روش‌های تعریف E به صورت زیر خواهد بود:

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2+\gamma\sum_{i,j}w_{ji}^2$$

پس در رابطه تغییر وزن‌ها در Backpropagation نیز باید همین تغییر را اضافه کرد، تمامی رابطه دست نخورده باقی خواهد ماند و فقط در هر تکرار در عدد ثابت (1-2γη) ضرب می‌شود. پس استفاده از این تعریف E معادل استفاده از استراتژی weight decay است (تمرین ۴.۱۰).

- اضافه کردن جمله ای برای خطاها در شیب یا مشتق تابع هدف. گاهی علاوه بر مقادیر تابع هدف، مقادیر مشتق تابع هدف نیز در نمونه های آموزشی وجود دارد. برای مثال، (simard et al. 1992) کاربردی را برای تشخیص کاراکتر معرفی می‌کند که در آن از مشتقات نمونه های آموزشی نیز استفاده شده است، در این کاربرد شبکه را مجبور می‌کند که نسبت به جابجایی حروف در صفحه بی تفاوت باشد. (Mitchell and Thrun 1993) نیز متدهایی را برای محاسبه‌ی مشتقات آموزشی بر اساس دانش قبلی ارائه می‌کنند. در هر دوی این سیستم‌ها (که در فصل ۱۲ آمده‌اند)، تابع خطا با اضافه شدن جمله ای که اختلاف مشتقات آموزشی و مشتقات واقعی شبکه است تغییر می‌یابد. مثالی از چنین تابع خطایی در زیر آمده،

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}\left[(t_{kd}-o_{kd})^2+\mu\sum_{j\in inputs}\left(\frac{\partial t_{kd}}{\partial x_d^j}-\frac{\partial o_{kd}}{\partial x_d^j}\right)^2\right]$$

در اینجا x_d^j نشان دهنده‌ی واحد ورودی j ام برای نمونه‌ی آموزشی d است. بنابراین، $\frac{\partial t_{kd}}{\partial x_d^j}$ مشتق آموزشی که چگونگی تغییر خروجی هدف t_{kd} را با تغییر x_d^j نشان می‌دهد. به صورت مشابه $\frac{\partial o_{kd}}{\partial x_d^j}$ نیز نشان دهنده‌ی مشتق واقعی شبکه یاد گرفته شده است. ثابت μ نیز وزن نسبی مقادیر نمونه های آموزشی و مشتقات آموزشی را مشخص می‌کند.

- مینیمم کردن cross entropy شبکه برای مقادیر تابع هدف. یادگیری یک تابع احتمالی، مثل پیش بینی اینکه درخواست کننده وامی، وام را کامل برگرداند را بر اساس ویژگی‌هایی چون سن درخواست کننده و موجودی حسابش را در نظر بگیرید. با وجود اینکه نمونه های آموزشی فقط مقادیر منطقی تابع هدف را در بر دارد. (۱ یا صفر، بسته به اینکه درخواست کننده وام را برگردانده یا خیر). بهترین نمایش تابع هدف، مدل کردن خروجی برای احتمال بازگرداندن وام است، به جای اینکه یاد بگیریم که خود مقادیر ۰ یا ۱ را برای هر نمونه واقعی یاد بگیریم. در چنین شرایطی که در آن هدف یادگیری تخمین احتمالات است، می‌توان نشان داد که بهترین تخمین زننده‌ی احتمال شبکه‌هایی هستند که مقدارcross entropy را با تعریف زیر مینیمم کنند،

$$-\sum_{d\in D}t_d log o_d+(1-t_d)\log(1-o_d)$$

در اینجا o_d تخمین احتمال خروجی شبکه برای نمونه‌ی آموزشی d است و t_d نیز مقدار هدف برای نمونه‌ی آموزشی d است. فصل ۶ شرایط و دلیل اینکه محتمل‌ترین فرضیه‌ی شبکه فرضیه ای است که cross entropy را مینیمم می‌کند را بررسی کرده و قانون شیب نزول را برای این معیار و واحد های سیگموید پیدا خواهد کرد. همچنین در آنجا شرایط اینکه محتمل‌ترین فرضیه همان فرضیه‌ی است که مجموع خطاهای مربعی را مینیمم می‌کند را بررسی خواهیم کرد.

- با تغییر موثر تابع خطا می‌توان اشتراک وزن‌ها[1] یا بستن به هم[2] را برای واحدهای مختلف ورودی یا خروجی ایجاد کرد. ایده‌ی اصلی اجبار وزن‌های مختلف شبکه به داشتن مقادیر یکی است، معمولاً این کار توسط کاربرد بخاطر دانش قبلی در مورد مسئله صورت می‌گیرد. برای مثال (Waibel et al. 1989) و (Lang et al. 1990) کاربردی از شبکه های عصبی برای تشخیص صحبت را مطرح می‌کنند که در آن ورودی‌های شبکه عناصر فرکانسی صحبت در زمان‌های مختلف در پنجره‌ی زمانی ۱۴۴ میلی ثانیه است. یکی از فرض‌هایی که می‌توان انجام داد این است که عناصر فرکانسی که صدای مشخصی هستند (برای مثال صدای "eee") را باید مستقل از زمان دقیق آن در طول ۱۴۴ میلی ثانیه تشخیص داد. برای اعمال این قید، واحدهای مختلفی که ورودی از قسمت‌های مختلف پنجره‌ی زمانی دریافت می‌کنند باید وزن‌های مشترکی داشته باشند. اثر کلی این قید در فضای فرضیه های ممکن، کم کردن ریسک overfit و بهبود احتمال تعمیم به وضعیت‌های مشاهده نشده است. چنین اشتراک وزن‌هایی معمولاً با آموزش جداگانه‌ی وزن‌های مشترک و جایگزینی میانگینشان به جای آن‌ها صورت می‌گیرد. حاصل فرایند این است که اشتراک وزن‌ها به سمت مینیمم کردن تابع خطای دیگری میل می‌کنند که با تابع خطای اصلی یکسان نیست.

۴.۸.۲ دیگر متد های مینیمم کردن خطا

شیب نزول یکی از اصلی‌ترین متد های پیدا کردن فرضیه ای با مینیمم کردن تابع خطاست، اما این متد همیشه مؤثرترین متد نیست. بعضی مواقع در آموزش شبکه های پیچیده، backpropagation برای همگرا شدن به ده‌ها هزار تکرار حلقه‌ی تغییر وزن‌ها دارد. به همین دلیل، تعدادی الگوریتم بهینه سازی وزن‌ها ارائه شده و مورد مطالعه قرار گرفته است. برای مشاهده‌ی دیگر روش‌ها، بهتر است متد تغییر وزنی را با دو انتخاب در نظر بگیریم: انتخاب یک جهت برای تغییر بردار وزن‌ها و انتخاب طولی برای حرکت به آن سمت. در backpropagation این جهت با عکس گرادیان انتخاب می‌شود و طولی که حرکت می‌کنیم با ثابت ضریب یادگیری η معلوم می‌گردد.

یکی از متد های بهینه سازی، که جستجوی خطی[3] نامیده می‌شود روشی متفاوت برای انتخاب طول تغییر وزن ارائه می‌کند. در کل، زمانی که خطی برای جهت تغییر وزن‌ها انتخاب می‌شود، طول تغییر با پیدا کردن مینیمم تابع خطا بر روی این خط انتخاب می‌شود. توجه دارید که این کار ممکن است باعث تغییر بسیار بزرگ یا بسیار کوچکی، متناسب با فاصله‌ی مینیمم تابع خطا بر روی این خط، در وزن‌ها شود. روش دیگری که با ایده‌ی جستجوی خطی ایجاد شده، روش مکمل گرادیان[4] نام دارد. در این جا، سری‌ای از جستجو های خطی برای جستجوی مینیممی در سطح خطا انجام می‌شود. در مرحله‌ی اول این سری جستجو جهت عکس گرادیان انتخاب می‌شود. در هر مرحله جهتی جدید انتخاب شده که تغییر در آن جهت باعث تغییر در جهت‌های قبلی نگردد و عنصر خطای گرادیان که صفر شده صفر باقی خواهد ماند.

با وجود اینکه استفاده از متد های دیگر گاهی در سرعت آموزش شبکه تأثیر دارند، اما متدهایی چون مکمل گرادیان تأثیر خاصی بر روی خطای تعمیم شبکه‌ی خروجی ندارند. تنها تأثیر بر روی خطای نهایی تفاوت بین فرایند های مینیمم سازی در افتادن در مینیمم‌های نسبی متفاوت است. (Bishop 1996) بحث کاملی درباره‌ی متدهای بهینه سازی برای آموزش شبکه های عصبی انجام می‌دهد.

---

[1] weight sharing
[2] tying together
[3] line search
[4] conjugate gradient

۴.۸.۳ شبکه های دور دار

تا به الآن فقط به شبکه‌هایی پرداختیم که متناسب با گراف‌های بدون دور بودند. شبکه های دور دار[1] شبکه های عصبی‌ای هستند که برای داده های سری‌های زمانی استفاده می‌شوند و خروجی شبکه در زمان t ورودی زمان t+1 خواهد بود. در چنین شرایطی، حلقه ای در شبکه موجود است. برای درک بهتر، فرض کنید که می‌خواهیم با استفاده از شاخص‌های سهام در هر روز x(t) متوسط قیمت سهام را برای روز بعدی y(t+1) پیش بینی کنیم. با داشتن سری‌ای از این اطلاعات، یکی از راه حل‌های بسیار ساده استفاده از شبکه‌ی تک سویه و استفاده از x(t) ها برای پیش بینی y(t+1) هاست. شبکه ای مثل آنچه در شکل ۴.۱۱ قسمت a آمده است.

یکی از مشکلات این راه این است که مقدار y(t+1) فقط با توجه به x(t) پیش بینی می‌شود و هیچ تأثیری از مقادیر قبلی x نخواهد پذیرفت. در حالی که این تأثیر بسیار حیاتی است، برای مثال، فرض کنید که متوسط قیمت سهام روز بعد به میزان تغییر شاخصی بین امروز و دیروز وابسته است. با وجود اینکه این مشکل با اضافه کردن مقادیر x(t-1) حل می‌شود اما نمی‌توان از این راه حل برای اضافه کردن تمامی مقادیر گذشته‌ی x به سیستم استفاده کرد. شبکه‌ی دور داری که در شکل (b) 4.11 نشان داده شده راه حلی برای اساسی این مشکل است. در این شبکه ما واحد پنهان اضافی b و واحد ورودی جدید c(t) را به شبکه اضافه کرده‌ایم. مقدار c(t) تعریف شده تا همیشه مقدار واحد b را در زمان t-1 داشته باشد. پس مقدار ورودی c(t) در هر پله‌ی زمان دقیقاً همان مقدار واحد پنهان b در پله‌ی قبلی زمان است. چنین ساختاری باعث می‌شود تا شبکه رفتاری با توجه به گذشته انجام دهد، واحد b اطلاعات لازم را برای آینده ذخیره می‌کند و واحد c نیز از گذشته خبر می‌دهد. چون هر بار که b محاسبه می‌شود به عنوان ترکیبی از x(t) و c محاسبه می‌شود پس اطلاعات مربوط به داده های قدیمی‌تر را نیز در خود ذخیره کرده است. از ساختار شبکه های دور دار دیگری نیز می‌شد برای این مثال استفاده کرد. برای مثال، می‌توان چندین لایه بین ورودی و واحد b قرار داد و یا اینکه می‌توان از چندین حلقه به جای یک حلقه استفاده کرد.

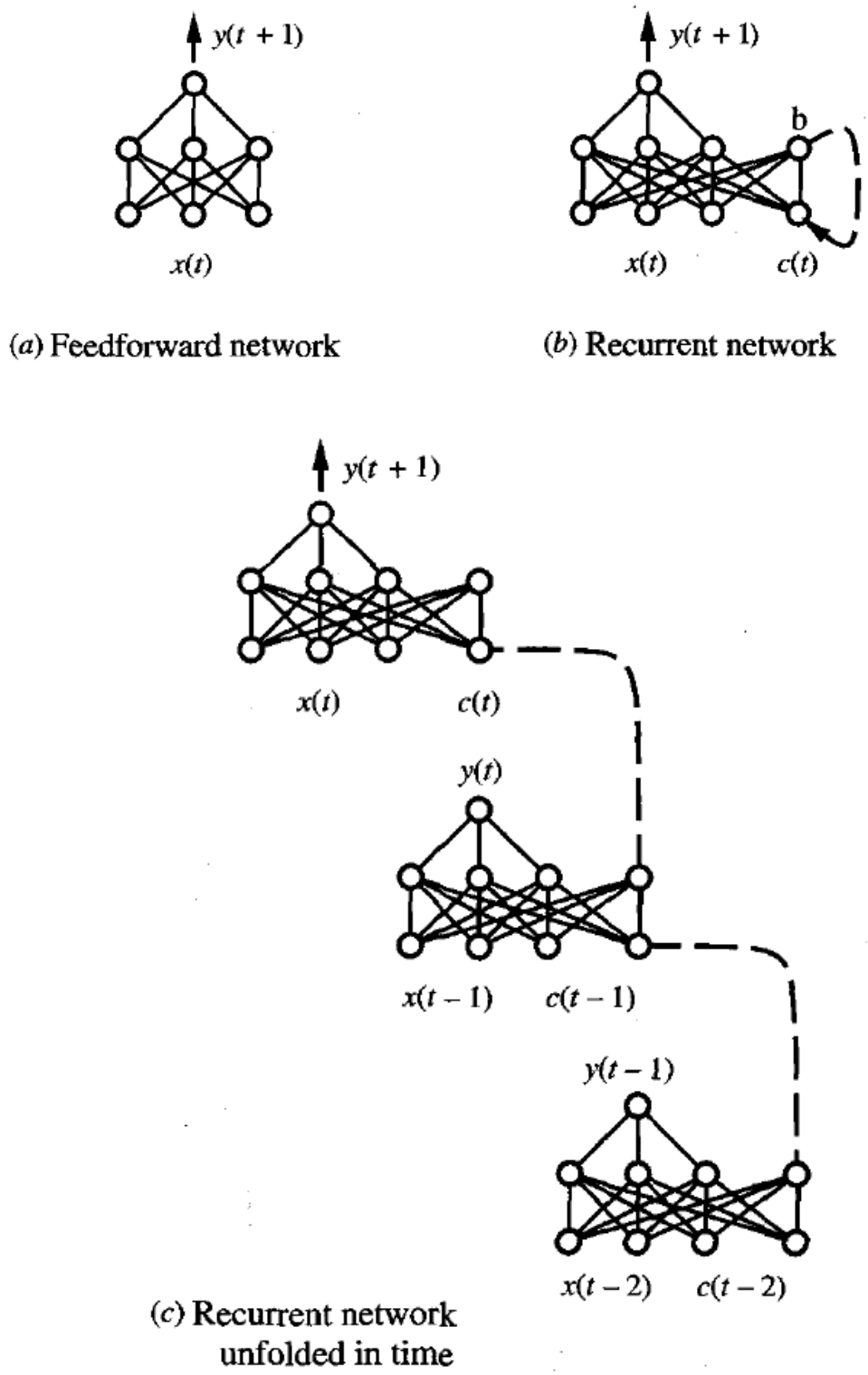
چگونه می‌توان شبکه های دور دار را آموزش داد؟ انواع مختلفی از شبکه های دور دار وجود دارد و متد های بسیاری برای آموزش آن‌ها پیشنهاد شده است. (برای اطلاعات بیشتر به Jordan 1986; Elman 1990; Mozer 1995; Williams and Zipser 1995 رجوع کنید). بسیار جالب است که بدانیم می‌توان با یک تغییر کوچک در Backpropagation شبکه‌هایی نظیر شکل (b) 4.11 را آموزش داد. برای درک این تغییر، شکل (c) 4.11 را در نظر بگیرید که ساختار تا نشده ی[2] شبکه را نشان می‌دهد. در اینجا به جای حلقه‌هایی در طول زمان از کپی‌های بسیاری از همان شبکه استفاده کرده‌ایم. توجه دارید که این ساختار شبکه‌ی جدید هیچ دوری ندارد. بنابراین می‌توان شبکه‌ی تا نشده را با استفاده از Backpropagation آموزش داد. در واقع در عمل فقط یک کپی از شبکه آموزش داده می‌شود و یک دسته وزن خواهیم داشت. بنابراین بعد از آموزش شبکه‌ی تا نشده می‌توان مقدار هر وزن را میانگین وزن‌های نظیر در تمامی کپی‌ها دانست. (Mozer 1995) این فرایند را با جزئیات توضیح داده است. در کل، شبکه های دور دار سخت‌تر از شبکه های ساده آموزش داده می‌شوند و تعمیم‌های قابل اطمینانی نیز نمی‌دهند. با این وجود چون قابلیت تعمیم را افزایش می‌دهند همچنان جزو شبکه های مهم محسوب می‌شوند.

---

[1] recurrent networks
[2] unfolded

Telegram: @Farabar_BI

www.farabar.net



شکل ۴.۱۱ شبکه های دور دار.

www.farabar.net

٤.٨.٤ ساختار شبکه‌ی پویا

تا این لحظه به آموزش شبکه های عصبی به عنوان پیدا کردن وزن‌ها برای شبکه ای با ساختار گرافی ثابت پرداخته‌ایم. متد های بسیاری درباره‌ی پویا[1] بودن ساختار شبکه ارائه شده که در صورت نیاز، شبکه‌ها می‌توانند افزایش یا کاهش واحد و یا ارتباط (گره) داشته باشد تا قدرت تعمیم و موثر بودن آموزش را افزایش دهد.

یکی از این ایده‌ها شروع از شبکه بدون هیچ واحد پنهان، و افزایش واحد های پنهان مطابق با نیاز، تا میزان خطا را تا حد قابل قبولی کاهش دهد. الگوریتم Cascade-Correlation (Fahlman, Lebiere 1990) چنین الگوریتمی است. این الگوریتم در ابتدا شبکه ای می‌سازد که هیچ واحد پنهانی ندارد. برای مثال، در مثال تشخیص چهره فقط ۴ واحد خروجی که هر کدام مستقیماً با تمامی گره های ورودی صفحه‌ی 30x32 در ارتباطند ایجاد می‌کند. بعد از آموزش این شبکه، معلوم می‌شود که مقدار خطای قابل توجهی از باقی می‌ماند، زیرا که تابع هدف را نمی‌توان با شبکه ای تک لایه یاد گرفت. پس بنابراین الگوریتم یک واحد پنهان به شبکه اضافه می‌کند و وزن‌های مربوطه را چنان تعیین می‌کند که رابطه‌ی بین مقدار واحد پنهان و خطای باقی‌مانده‌ی شبکه حداکثر شوند. سپس این واحد جدید نصب می‌شود، تمامی وزن‌هایش ثابت نگه داشته می‌شود و ارتباطی بین آن و تمامی خروجی‌ها ایجاد می‌گردد. دوباره فرایند به حرکت می‌افتد، وزن‌های قدیمی دوباره آموزش داده می‌شوند (وزن‌های واحد پنهان همچنان ثابت می‌ماند). مقدار خطای باقیمانده دوباره چک می‌شود و اگر به اندازه‌ی قابل توجهی بزرگ بود واحدی دیگر به لایه‌ی پنهان اضافه خواهد شد. هر گاه که واحدی جدید به لایه‌ی پنهان اضافه می‌شود، از تمامی ورودی‌های اصلی و تمامی واحد های پنهان قبلی ورودی دریافت می‌کند. به همین منوال شبکه گسترش پیدا خواهد کرد تا خطا از حد آستانه ای کمتر شود و به مقدار قابل قبولی برسد. (Fahlman and Lebiere 1990) با توجه به اینکه همیشه فقط واحد جدید آموزش داده می‌شود نشان دادند که در Cascade-Correlation می‌توان به طور قابل توجهی تعداد آموزش‌ها را کم کرد. یکی از مشکلات کاربردی‌ای که هنگام به کار بردن این الگوریتم پیش می‌آید این است که چون تعداد واحد های اضافه شده نامحدود است پس خیلی ساده مشکل overfit رخ می‌دهد، پس همیشه باید اقدامات لازم را برای جلوگیری از overfit در استفاده از این الگوریتم انجام داد.

راه دیگر برای استفاده از ساختار شبکه‌ی پویا، دقیقاً عکس این Cascade-Correlation است. بجای شروع از ساده‌ترین شبکه‌ها و پیچیده تر کردن آن طی مراحل، با شبکه ای پیچیده شروع می‌کنیم و با معلوم شدن اینکه بعضی ارتباط‌ها مهم نیستند آن‌ها را هرس می‌کنیم. یکی از راه‌های تشخیص اینکه یک ارتباط مهم نیست این است که وزن ارتباط‌های غیر مهم معمولاً نزدیک به صفرند. راه دوم، که در کاربرد موفق‌تر به نظر می‌رسد، این است که ببینیم تغییر کوچکی در مقدار وزن چه تأثیری بر خطای E می‌گذارد. اثر تغییر w بر E ($\frac{\partial E}{\partial w}$) را می‌توان معیاری برای اهمیت ارتباط نظیر دانست. (LeCun et al. 1990) فرایندی را معرفی می‌کند که طی آن شبکه آموزش داده می‌شود و کم اهمیت‌ترین ارتباط‌ها نیز حذف می‌شوند، این فرایند آنقدر تکرار می‌شود که به شرط پایانی خاصی برسیم. به این فرایند، روش بهینه سازی ی صدمات مغز[2] نیز می‌گویند، زیرا که در هر مرحله، الگوریتم سعی می‌کند تا کم اهمیت‌ترین ارتباط‌ها را حذف کند. گفته می‌شود در شبکه های بسیار بزرگ برای تشخیص کاراکتر چنین روشی می‌تواند تعداد ارتباط‌ها را تا ۱/۴ کاهش دهد، و قدرت تعمیم شبکه را کمی بهبود می‌بخشد و بازده آموزش را به طور قابل توجهی بالا می‌برد.

---

[1] dynamic
[2] optimal brain damage

در کل، تکنیک‌های ساختار شبکه های پویا موفق آمیز بوده است. فقط باید دید تا آن‌ها به اندازه‌ی Backpropagation در افزایش قدرت تعمیم قوی هستند یا نه. با این وجود در مواردی نشان داده شده که به میزان قابل توجهی در زمان آموزش تأثیر می‌گذارند.

٤.٩ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر می‌شود:

- شبکه های عصبی مصنوعی متدی کاربردی برای یادگیری توابع حقیقی مقدار و برداری را بر روی ویژگی‌های گسسته و پیوسته ارائه می‌کنند، این متد در مقابل خطای داده های آموزشی مقاوم است. الگوریتم Backpropagation متداول‌ترین متد یادگیری شبکه است و در بسیاری از کارهای یادگیری نظیر تشخیص دستخط و کنترل ربات با موفقیت به کار رفته است.
- فضای فرضیه ای در نظر گرفته شده برای الگوریتم Backpropagation فضای تمامی توابع ممکنی که با تغییر وزن‌های شبکه ثابتی از واحدها و ارتباطها بیان می‌شوند است. شبکه های تک سویه با تعداد کافی واحد در هر لایه که شامل سه لایه واحد می‌شوند قابلیت تخمین هر تابعی را با دقت دلخواه را دارند. حتی شبکه‌هایی با اندازه‌ی واقعی می‌توانند فضای غنی‌ای از توابع غیر خطی را نمایش دهند، به همین دلیل شبکه های تک سویه گزینه‌ی خوبی برای یادگیری توابع گسسته و پیوسته که در حالت کلی فرم کلی مجهولی دارند است.
- Backpropagation فضای تمامی فرضیه های ممکن را با استفاده از شیب نزول و کاهش متناوب خطای بین شبکه و نمونه های آموزشی جستجو می‌کند. شیب نزول به سمت مینیمم نسبی خطای بین شبکه و نمونه های آموزشی برای وزن‌های شبکه میل خواهد کرد. در حالت کلی‌تر، شیب نزول متدی بالقوه مفید برای جستجوی فضاهای فرضیه ای پیوسته‌ی چند متغیره که در آن خطای آموزشی تابعی مشتق پذیر از پارامتر های فرضیه است.
- یکی از فریبنده‌ترین ویژگی‌های Backpropagation قابلیت آن در ایجاد ویژگی‌های جدیدی است که به طور صریح در ورودی شبکه در نظر گرفته نشده است. در کل، لایه های داخلی (پنهان) در یک شبکه‌ی چند لایه یاد می‌گیرند تا ویژگی‌های میانی‌ای را نمایش دهند که برای یادگیری تابع هدف مفید بوده و به طور ضمنی در ورودی‌های شبکه بیان شده‌اند. این قابلیت، برای مثال، در شبکه ای ۸×۳×۸ در قسمت ۴.۶.۴ در کد گذاری منطقی انجام شده برای اعداد ۱ تا ۸ و در مثال تشخیص چهره در قسمت ۴.۷ با ویژگی‌های عکس در لایه‌ی پنهان بیان شده‌اند.
- overfit بر روی داده های آموزشی مشکلی مهم در یادگیری شبکه های عصبی است. overfit به شبکه ای ختم می‌شود که تعمیم ضعیفی روی داده های جدید دارد اما کارایی عالی‌ای روی داده های آموزشی دارد. متدهای Cross-Validation را می‌توان برای تخمین نقطه‌ی ایست مناسب برای جستجو و مینیمم کردن ریسک overfit به کاربرد.
- با وجود اینکه Backpropagation متداول‌ترین الگوریتم برای یادگیری شبکه های عصبی است، اما الگوریتم‌های بسیار دیگری ارائه شده‌اند، این الگوریتم‌ها شامل الگوریتم‌هایی برای اهداف خاص می‌شوند. برای مثال، متدهای شبکه های عصبی حلقه دار، شبکه‌هایی را آموزش می‌دهند که حلقه های مستقیم دارند و الگوریتم‌هایی چون Cascade Correlation ساختار شبکه را علاوه بر وزن‌های شبکه تغییر می‌دهند.

اطلاعات بیشتر در مورد شبکه های عصبی را می‌توانید در فصول دیگر این کتاب بیابید. توجیهی بیزی برای انتخاب معیار خطای مربعی در فصل ۶ ارائه می‌شود، در این فصل همچنین توجیهی برای مینیمم کردن cross-entropy به جای خطای مربعی در شرایط خاص آورده

شده است. نتایج تئوری تعداد نمونه های آموزشی لازم برای رسیدن به شبکه ای قابل اطمینان برای توابع حقیقی و بعد Vapnik-Chervonenkis برای شبکه های خاص در فصل ۷ بحث شده‌اند. در فصل ۵ نیز بحثی در مورد overfit و چگونگی دوری از آن آورده شده. همچنین در فصل ۱۲ متدهایی برای استفاده از دانش قبلی برای بهبود تعمیم دقت شبکه های عصبی مورد بحث قرار گرفته است.

کار بر روی شبکه های عصبی به زمان‌های اولیه‌ی علوم کامپیوتر بر می‌گردد. (McCulloch and Pitts 1943) مدلی برای یک نورون مشابه پرسپترون ارائه کردند، در طی دهه‌ی ۱۹۶۰ کارهای بسیاری بر روی قابل قبول بودن این مدل انجام گرفت. در اوایل دهه‌ی ۶۰ (Windrow and Hoff 1960) شبکه های پرسپترون (که آن‌را adeline می‌نامیدند) و قانون دلتا را مورد بررسی قرار دادند، (Rosenblatt 1962) همگرایی قانون آموزش پرسپترون را ثابت کرد. با این وجود، در اواخر دهه‌ی ۶۰ مشخص شد که پرسپترون تک لایه محدودیت نمایش دارد و الگوریتم موثری نیز برای آموزش شبکه های چند لایه معرفی نشده بود. (Minsky and Papert 1969) نشان دادند که حتی تابع ساده ای چون XOR را نمی‌توان با شبکه های پرسپترون تک لایه نمایش داد و کار بر روی شبکه های عصبی در دهه‌ی ۷۰ متوقف شد.

در اواسط دهه‌ی ۸۰ با اختراع Backpropagation و الگوریتم‌های مربوطه‌ی آموزش شبکه های چند لایه (Rumelhart and McClelland 1986; Parker 1985)، کار بر روی شبکه های عصبی دوباره از سر گرفته شد. اساس این ایده‌ها را می‌توان در کارهای قبلی جست (Werbos 1975). از دهه‌ی ۸۰ Backpropagation به یکی از متداول‌ترین متدهای یادگیری تبدیل شد و بسیاری از روش‌های دیگر مربوطه‌ی شبکه های عصبی مورد مطالعه قرار گرفت. با ظهور رایانه های کم قیمت در همان دوره تحقیقات بر روی الگوریتم‌های محاسباتی‌تر که در دهه‌ی ۶۰ ممکن نبود شروع شد.

کتاب‌های زیادی به مبحث شبکه های عصبی اختصاص یافته است. یکی از کتاب‌های قدیمی اما مفید درباره‌ی متدهای یادگیری پارامتری برای تشخیص الگو توسط (Duda and Hart 1973) نوشته شده است. کتاب (Widrow and Stearns 1985) به پرسپترون‌ها و شبکه های تک لایه‌ی مربوطه و کاربردشان می‌پردازد. (Rumelhart and McClelland 1986) مجموعه‌ی منتخبی از مقالات که علاقه‌ی کار بر روی این متدها را افزایش داد را از اواسط دهه‌ی ۸۰ به بعد جمع آوری کرده‌اند. کتاب‌های اخیر در مبحث شبکه های عصبی شامل (Bishop 1996)، (Chauvin and Rumelhart 1995)، (Freeman and Skapina 1991)، (Fu 1994)، (Hecht-Nielsen 1990) و (Hertz et al. 1991) می‌شود.

تمرینات

۴.۱ مقادیر وزن‌های w_0، w_1 و w_2 را برای پرسپترونی که سطح تصمیمش در شکل ۴.۳ آورده شده تعیین کنید. فرض کنید که این سطح محور x_1 را در ۱- و محور x_2 را در ۲ قطع می‌کند.

۴.۲ پرسپترونی با دو ورودی طراحی کنید که تابع منطقی $A \wedge \neg B$ را نشان دهد. از شبکه ای دو لایه از پرسپترون‌ها برای نشان دادن A XOR B استفاده کنید.

۴.۳ دو پرسپترون با رابطه‌ی مقدار حدی $w_0 + w_1x_1 + w_2x_2 > 0$ را در نظر بگیرید. پرسپترون A وزن‌های زیر را دارد

$$w_0 = 1, \qquad w_1 = 2, \qquad w_2 = 1$$

و پرسپترون B وزن‌های زیر را داراست

$$w_0 = 0, \qquad w_1 = 2, \qquad w_2 = 1$$

تعیین کنید که آیا پرسپترون A از پرسپترون B کلی‌تر است؟ (تعریف کلی‌تر بودن در فصل ۲ آمده است).

۴.۴ از قانون آموزش دلتا برای یک واحد خطی با دو ورودی استفاده کنید و آن‌را برای تناسب با مفهوم هدف $-2 + x_1 + 2x_2 > 0$ آموزش دهید. خطای E را بر حسب تعداد تکرار های آموزش رسم کنید. سطح تصمیم را بعد از ۵، ۱۰، ۵۰، ۱۰۰، ... بار اجرا رسم کنید.

(a) از مقادیر مختلف ثابت برای η استفاده کرده و همچنین از مقدار متغیر η_0/i برای آموزش استفاده کنید. عملکرد کدام حالت بهتر است؟

(b) از افزایش و آموزش دسته ای[1] استفاده کنید. کدام یک زودتر همگرا می‌شود؟ هر دو معیار تعداد تغییر وزن‌ها و کل زمان اجرا را در نظر بگیرید.

۴.۵ قانون شیب نزول را برای تک خروجی‌ای به فرم o به شکل زیر استخراج کنید

$$o = w_0 + w_1x_1 + w_1x_1^2 + w_2x_2 + w_2x_2^2 + \cdots + w_nx_n + w_nx_n^2$$

۴.۶ به طور غیر رسمی توضیح دهید که چرا قانون آموزش دلتا در رابطه‌ی ۴.۱۰ فقط تخمینی از قانون شیب نزول رابطه‌ی ۴.۷ است.

۴.۷ شبکه‌ی عصبی تک سویه ای را در نظر بگیرید که دو ورودی a و b و یک واحد پنهان c و یک واحد خروجی d دارد. این شبکه پنج وزن $(w_{ca}, w_{cb}, w_{c0}, w_{dc}, w_{d0})$ دارد، در این نمایش w_{x0} نشان دهنده‌ی مقدار آستانه‌ی برای واحد x است. این وزن‌ها را با مقادیر اولیه‌ی (.1,.1,.1,.1,.1) مقدار دهی اولیه کنید سپس الگوریتم Backpropagation را به آن‌ها اعمال کنید. فرض کنید که ضریب یادگیری η=.3، تکانه α=.9، وزن‌ها را برای هر نمونه جداگانه تغییر دهید، و نمونه های آموزشی به صورت زیرند:

a	B	D
۱	۰	۱
۰	۱	۰

۴.۸ الگوریتم Backpropagation در جدول ۴.۲ را طوری تغییر دهید که بر روی واحدهایی که از تابع tanh به جای تابع سیگموید استفاده می‌کنند عمل کند. بدین معنا که $o = \tanh(\vec{w}.\vec{x})$ قانون تغییر وزن را برای لایه خروجی و لایه پنهان ارائه کنید.

(راهنمایی $\tanh'(x) = 1 - \tanh^2(x)$).

۴.۹ شبکه‌ی ۸×۳×۸ نشان داده شده در شکل ۴.۷ را در نظر بگیرید. فرض کنید که برای چنین کار مشابهی می‌خواهیم از شبکه ای ۸×۱×۸ کمک بگیریم؛ شبکه ای که فقط یک واحد پنهان دارد. توجه دارید که هشت نمونه‌ی آموزشی شکل ۴.۷ را می‌توان با هشت مقدار برای گره پنهان نظیر کرد (برای مثال ۰.۱، ۰.۲، ... ۰.۸). آیا بنابراین شبکه ای با فقط یک واحد پنهان می‌تواند تابع همانی را بر روی این نمونه های آموزشی یاد بگیرد. راهنمایی: این سؤال را در نظر بگیرید که "آیا مقادیری برای وزن‌های لایه‌ی پنهان وجود دارد که بتواند کد گذاری بالا را در

[1] batch learning

لایه‌ی پنهان ایجاد کند؟" "آیا مقادیری برای وزن‌های خروجی وجود دارد که بتوان ورودی را از این کد گذاری به سادگی استخراج کرد؟" و "آیا شیب نزول می‌تواند چنین وزن‌هایی را پیدا کند؟"

۴.۱۰ تابع خطای جایگزین زیر را به جای رابطه‌ی قسمت ۴.۸.۱ در نظر بگیرید.

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_k - o_{kd})^2 + \gamma\sum_{i,j} w_{ji}^2$$

قانون تغییر شیب نزول را برای این تعریف E محاسبه کنید. نشان دهید که این طرح را می‌توان با ضرب هر وزن در یک ثابت قبل از اعمال قانون شیب نزول جدول ۴.۲ پیاده سازی کرد.

۴.۱۱ از Backpropagation برای کار تشخیص چهره استفاده کنید. برای جزئیات کار از جمله داده های تصویری صورت، کد Backpropagation و کارهای خاص به http://www.cs.cmu.edu/~tom/book.html مراجعه کنید.

۴.۱۲ الگوریتم شیب نزول را در نظر بگیرید که برای یادگیری مفاهیم هدف متناسب با مستطیل‌های موجود در صفحه‌ی x,y به کار می‌رود. هر فرضیه را با گوشه سمت چپ پایین و راست بالا متناسب با llx,lly,urx,ury توصیف می‌شود. نقطه‌ی <x,y> تنها زمانی توسط فرضیه‌ی <llx,lly,urx,ury> مثبت دسته بندی می‌شود که نقطه‌ی <x,y> درون مستطیل قرار داشته باشد. از تعریف خطای E آمده در فصل استفاده کنید. آیا می‌توانید نسخه ای بازبینی شده از شیب نزول را ارائه دهید که چنین فرضیه های مستطیلی‌ای را یاد بگیرد. توجه دارید که E بر حسب llx,lly,urx,ury پیوسته نیست، مثل حالت پرسپترون. (راهنمایی: دو راه استفاده شده برای پرسپترون را در نظر بگیرید: (۱) تغییر قانون دسته بندی به صورتی که تابع پیش بینی تابعی پیوسته بر حسب ورودی‌ها باشد و (۲) تعریف تابع خطای دیگری، مثل فاصله تا مرکز مستطیل، برای استفاده در قانون دلتا برای آموزش پرسپترون). آیا الگوریتمتان به فرضیه ای با خطای مینیمم که در آن نمونه های مثبت و منفی با مستطیلی قابل تقسیم باشند میل می‌کند؟ چه زمانی نمی‌توان با مستطیل چنین کاری کرد؟ آیا مشکلات مینیمم‌های نسبی رخ می‌دهند؟ الگوریتم شما چه رابطه ای با متد های نمادین ای که برای عطف ویژگی‌ها استفاده می‌شوند دارد؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

ابر صفحه ای	Hyperplane
اندیس	Index
بدترین حالت	worse case
برانگیخته	Excited
برنامه نویسی خطی	linear programming
بیزی	Bayesian
پهنای شبکه	Network width
تابع فشرده ساز	squashing function
تابع منطق	logistic function
تا نشده ی	Unfolded
ترتیب ساده به پیچیده	simple-to-complex
تک سویه	feed-forward

penalty term	جمله‌ی خطا
Interpolation	درون یابی
True	درست
linearly separable	دسته بندی پذیر خطی
Artificial neural networks	شبکه های عصبی مصنوعی
termination condition	شرط پایانی
Gradient Descent	شیب نزول
Incremental gradient descend	شیب نزول افزایشی
learning rate	ضریب یادگیری
False	غلت
Inhibited	غیر برانگیخته
perceptron rule	قانون پرسپترون
delta rule	قانون دلتا
generalization accuracy	قدرت تعمیم شبکه
Gradient	گرادیان
Node	گره
Sequential	ماشین‌های ترتیبی
validation set	مجموعه‌ی تایید
Threshold	مقدار آستانه
linear unit	واحد خطی
Edge	یال

فصل پنجم: ارزیابی فرضیه‌ها

ارزیابی تجربی دقت فرضیه‌ها اساس یادگیری ماشین است. این فصل معرفی‌ای بر متد های تخمین دقت فرضیه‌ها با تاکید بر سه سؤال زیر ارائه می‌کند. اول اینکه با در دست داشتن دقت فرضیه بر روی مجموعه‌ی محدودی از داده‌ها، این تقریب چقدر به دقت بر روی نمونه‌های جدید نزدیک است؟ دوم اینکه با دانستن اینکه یک فرضیه از فرضیه های دیگر دقت بیشتری بر روی تعدادی نمونه دارد، چقدر احتمال دارد که این فرضیه در کل از فرضیه های دیگر دقت بیشتری داشته باشد؟ سوم اینکه زمانی که داده‌ها محدود است بهترین روش برای استفاده‌ی این داده‌ها هم برای آموزش و هم برای ارزیابی چیست؟ زیرا که تعداد محدود نمونه‌ها ممکن است معرف توزیع کلی نمونه‌ها نباشد و در بدست آوردن دقت فرضیه بر روی تمامی نمونه‌ها گمراه کننده باشند. متد های آماری، با فرض‌هایی که درباره‌ی توزیع داده‌ها انجام می‌دهند، اجازه می‌دهند تا حداکثر اختلاف بین دقت مشاهده شده بر روی داده های موجود و دقت واقعی روی کل توزیع داده‌ها را محاسبه کنیم.

5.1 انگیزه

در بسیاری از موارد ارزیابی فرضیه‌ی یاد گرفته شده با حداکثر دقت ممکن بسیار مهم است. یکی از دلایل این اهمیت، تشخیص قابل استفاده بودن فرضیه است (مشخص شدن این است که آیا این فرضیه را به کار ببریم یا خیر). برای مثال، زمانی که از یک پایگاه داده‌ی محدود برای بررسی تأثیر داروها استفاده می‌کنیم، دانستن دقت فرضیه‌ی یاد گرفته شده بسیار مهم است. دلیل دوم اهمیت ارزیابی فرضیه‌ها این است که ارزیابی فرضیه‌ها عنصر داخلی بسیاری از الگوریتم‌های یادگیری است. برای مثال، در هرس کردن درخت‌های تصمیم برای حل کردن مشکل overfit باید تأثیر هرس بر دقت درخت حاصل را در هر مرحله بدانیم. بنابراین درک وجود خطای ذاتی در تخمین دقت درخت‌ها قبل از هرس بعد از هرس اهمیت بسیار دارد.

تخمین دقت یک فرضیه هنگامی که تعداد داده‌ها زیاد است بسیار ساده خواهد بود. با این حال، زمانی که لازم است با تعداد محدودی داده فرضیه ای را یاد بگیریم و ارزیابی کنیم دو مشکل اساسی پیش می‌آید:

- بایاس در تخمین. اول اینکه دقت فرضیه بر روی نمونه های آموزشی اغلب معیار ضعیفی برای دقت فرضیه بر داده های جدید است. زیرا که فرضیه از همین داده‌ها به وجود آمده است، پس این نمونه های تخمینی بایاس دار و خوش‌بینانه از دقت فرضیه بر داده های جدید می‌زنند. این مشکل بیشتر مواقعی پیش می‌آید که فضای فرضیه ای، فضایی کامل است و به فرضیه اجازه می‌دهد بر روی نمونه های آموزشی overfit شود. برای بدست آوردن تخمینی بدون بایاس از دقت فرضیه بر روی داده های جدید، معمولاً فرضیه را بر روی دسته نمونه های مجزایی از نمونه های آموزشی (دسته‌ی تست) می‌سنجیم.
- اختلاف در تخمین. دوم اینکه اگر دقت فرضیه را بر روی دسته‌ی تست که بایاس ندارند بسنجیم، با این حال امکان دارد که این دقت بدست آمده با دقت واقعی اختلاف داشته باشد، این اختلاف به چگونگی انتخاب دسته‌ی تست وابسته است. با کاهش اندازه‌ی دسته‌ی تست این میزان خطای احتمالی نیز افزایش می‌یابد.

در این فصل به متد های ارزیابی فرضیه های یاد گرفته شده، متدهای مقایسه‌ی دقت دو فرضیه، و متد های مقایسه‌ی دقت دو الگوریتم یادگیری مختلف هنگامی با داده‌ها محدودند می‌پردازیم. اکثر این مباحث بر پایه‌ی قوانین پایه ای آماری و تئوری نمونه برداری[1] هستند، البته در طول این فصل فرض شده که خواننده هیچ اطلاعات قبلی‌ای در مورد مباحث پیچیده‌ی آماری ندارد. تحقیق بر روی تست‌های آماری بررسی فرضیه‌ها خیلی وسیع است. این فصل خلاصه ای مقدمه ای از این تحقیقات آماری با تاکید بر قسمت‌هایی که بیشترین رابطه را با یادگیری و تخمین و مقایسه‌ی دقت فرضیه‌ها دارد را نیز ارائه می‌کند.

5.2 تخمین دقت فرضیه‌ها

زمانی که دقت یک فرضیه را تخمین می‌زنیم هدف دقت فرضیه برای دسته بندی نمونه های جدید است، علاوه بر این علاقه داریم که احتمال خطا در تخمین این دقت را نیز بسنجیم (چه میزان خطایی را باید در این تخمین در نظر گرفت).

در تمام طول این فصل از نماد گذاری ذیل برای مسایل یادگیری استفاده خواهیم کرد. فضای نمونه های X (برای مثال مجموعه‌ی کل افراد جامعه) وجود دارد که تابع هدف‌های متفاوتی (مثل افرادی که می‌خواهند امسال تخته اسکی جدید بخرند) روی آن‌ها تعریف می‌شوند. ما فرض می‌کنیم که تعداد تکرار اعضای مختلف X مساوی نیست. راه حل ساده برای در نظر گرفتن این فرض این است که توزیع احتمال مجهول $\mathcal{D}$ را که بر روی X تعریف شده برای تعداد تکرار نمونه‌ها در نظر بگیریم (این تابع توزیع ممکن است برای افراد 19 ساله خیلی بیشتر از افراد 109 ساله باشد). توجه داشته باشید که $\mathcal{D}$ هیچ اطلاعاتی در مورد مثبت یا منفی بودن نمونه x به ما نمی‌دهد؛ این توزیع فقط احتمال برخورد با نمونه x را به ما می‌دهد. کار یادگیری، یادگیری تابع هدف f با استفاده از فضای فرضیه های ممکن H است. نمونه های آموزشی تابع هدف f توسط یک معلم به یادگیر داده می‌شود. معلم جداگانه بر اساس توزیع $\mathcal{D}$ نمونه‌ها را انتخاب می‌کند سپس نمونه‌ی x را با مقدار تابع هدف f(x) به یادگیر می‌دهد.

برای تصور، تابع هدف "افرادی که می‌خواهند امسال تخته اسکی جدید بخرند" را با نمونه های آموزشی‌ای که از تحقیقی که از افرادی که به پیست اسکی وارد می‌شوند بدست آمده در نظر بگیرید. در این مثال، فضای نمونه های X کل افراد جامعه است، این نمونه‌ها با ویژگی‌هایی نظیر سن، شغل، تعداد دفعات اسکی در سال و غیره توصیف می‌شوند. توزیع $\mathcal{D}$ برای هر فرد x احتمال اینکه فرد بعدی‌ای باشد که وارد پیست می‌شود را می‌دهد. تابع هدف $f:X\rightarrow\{0,1\}$ افراد را بر اساس اینکه قصد خرید تخته اسکی جدید دارند دسته بندی می‌کند.

[1] sampling theory

با این نماد گذاری کلی ما به دنبال جواب دو سؤال زیر هستیم:

1. برای فرضیه‌ی h و یک مجموعه نمونه n عضوی که اعضایش با توزیع احتمال $\mathcal{D}$ انتخاب شده‌اند، بهترین تخمین دقت h بـر روی نمونه های جدیدی که با همان توزیع انتخاب می‌شوند چقدر است؟
2. خطای احتمالی این تقریب دقت چقدر است؟

5.2.1 خطای نمونه ای و خطای واقعی

برای جواب به سؤال‌های مطرح شده، لازم است که ابتدا تفاوت بین دو دقت یا خطا را بیان کنیم. یکی نسبت خطای فرضیه بر روی نمونه های موجود است. دیگری نسبت خطای فرضیه بر روی کل توزیع نامعلوم $\mathcal{D}$ از نمونه‌هاست. این دو خطا را به ترتیـب خطـای نمونـه ای[1] و خطـای واقعی[2] می‌نامیم.

خطای نمونه ای یک فرضیه بر اساس مجموعه نمونه های S از X نسبتی از S است که فرضیه اشتباه دسته بندی می‌کند:

تعریف: خطای نمونه ای ($error_S(h)$) برای فرضیه‌ی h بر اساس تابع هدف f و مجموعه نمونه های S به شکل زیر تعریف می‌شود:

$$error_S(h) \equiv \frac{1}{n}\sum_{x\in S}\delta(f(x),h(x))$$

در این رابطه n تعداد نمونه های S و δ(f(x),h(x)) اگر $f(x) \neq h(x)$ یک است و در غیر این صورت صفر می باشد.

خطای واقعی یک فرضیه احتمال این است که فرضیه نمونه ای که با توزیع $\mathcal{D}$ انتخاب شده را اشتباه دسته بندی می‌کند.

تعریف: خطای واقعی ($error_{\mathcal{D}}(h)$) برای فرضیه‌ی h و بر اساس تابع هدف f و توزیع $\mathcal{D}$ احتمال این است که h نمونه‌ی انتخـابی بـا توزیع $\mathcal{D}$ را اشتباه دسته بندی کند.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x\in\mathcal{D}}[f(x) \neq h(x)]$$

در این رابطه $\Pr_{x\in\mathcal{D}}$ احتمال را برای x ی که با توزیع $\mathcal{D}$ انتخاب شده باشد نشان می دهد.

همیشه ما قصد داریم که $error_{\mathcal{D}}(h)$ را برای فرضیه پیدا کنیم، زیرا که این میزان خطایی که در دسته بندی نمونه های جدید وجـود دارد را بیان می‌کند. با این وجود، ما فقط می‌توانیم مقدار $error_S(h)$ را با داشتن مجموعه‌ی S اندازه گیـری کنـیم. سـؤال اصـلی ایـن اسـت کـه "$error_S(h)$ چقدر برای تخمین $error_S(h)$ مناسب است؟"

[1] sample error
[2] true error

5.2.2 بازه های اطمینان برای فرضیه های گسسته مقدار

چگونه به سؤال "$error_S(h)$ با چه میزان دقت $error_{\mathcal{D}}(h)$ را تخمین می‌زند؟" در زمانی که h تابعی گسسته مقدار است پاسخ می‌دهیم؟ به عبارت دقیق‌تر، فرض کنید که می‌خواهیم خطای واقعی را برای فرضیه‌ی گسسته مقدار h را با استفاده از مجموعه نمونه های S در شرایط زیر تعیین کنیم:

- مجموعه‌ی S شامل n نمونه ای است که مستقل از یکدیگر و مستقل از h هستند که با توجه به $\mathcal{D}$ انتخاب شده‌اند.
- N≥ 30
- فرضیه‌ی h، r تعداد نمونه را اشتباه دسته بندی می‌کند ($error_s(h)=r/n$)

در چنین شرایطی، تئوری آمار به ما اجازه می‌دهد تا نتایج زیر را بگیریم:

1. بدون داشتن اطلاعات بیشتر، محتمل‌ترین مقدار $error_{\mathcal{D}}(h)$، $error_S(h)$ است.
2. با احتمال تقریباً 95٪ خطای واقعی $error_{\mathcal{D}}(h)$ در بازه‌ی زیر است:

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)\big(1-error_S(h)\big)}{n}}$$

برای تصور، مجموعه‌ی داده های S را با n = 40 نمونه و فرضیه‌ی h را با r =12 خطا بر روی این نمونه‌ها در نظر بگیرید. در این مثال، خطای نمونه ای $error_S(h)$=12/40=0.3 است. بدون داشتن اطلاعات بیشتر، بهترین تخمین برای $error_{\mathcal{D}}(h)$ همان مقدار 0.3 است. با این وجود، انتظار نمی‌رود که این تخمین، تخمین کاملی از خطای واقعی باشد. اگر دسته‌ی دیگری از نمونه‌ها مثل S' که 40 نمونه دارد داشته باشیم، قاعدتاً انتظار داریم که $error_{S'}(h)$ تقریباً با $error_S(h)$ مساوی باشد. اختلاف احتمالی این دو مقدار به چینش دو مجموعه‌ی S و S' وابسته است. در واقع اگر این آزمایش را بارها تکرار کنیم و در هر بار تکرار از مجموعه‌ی 40 نمونه ای S_i استفاده کنیم، به این نتیجه خواهیم رسید که تقریباً در 95٪ این آزمایشات بازه‌ی محاسبه شده خطای واقعی را شامل می‌شود. به همین دلیل به این بازه، بازه‌ی اطمینان 95 درصدی تخمین خطای واقعی $error_{\mathcal{D}}(h)$ می‌گویند. در مثال فعلی که در آن r = 12 و n = 40 است، بازه‌ی 95٪ بر اساس داده های بالا 0.30±(1.96·.07)=0.30±.14 خواهد بود.

رابطه ای که در بالا برای بازه‌ی اطمینان 95 درصدی بیان شد را می‌توان برای اطمینان N درصدی تعمیم داد. مقدار ثابت 1.96 که در تعریف رابطه آمده برای اطمینان 95 درصدی است، با تغییر دادن این ثابت، z_N، می‌توان بازه‌ی اطمینان N% را محاسبه کرد. رابطه‌ی کلی محاسبه‌ی بازه‌ی اطمینان N% برای خطای $error_{\mathcal{D}}(h)$ در زیر آمده است:

$$error_S(h) \pm z_N\sqrt{\frac{error_S(h)\big(1-error_S(h)\big)}{n}} \qquad (5.1)$$

در این رابطه ثابت z_N بر حسب درصد اطمینان تغییر می‌کند. مقادیر نظیر بعضی درصدها در جدول 5.1 آمده است.

خطای N درصدی:	٪50	٪68	٪80	٪90	٪95	٪98	٪99

ثابت z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

جدول 5.1 مقادیر z_N برای بازه‌ی دوطرفه‌ی اطمینان N%.

بنابراین همان طور که به راحتی محاسبه شد، بازه‌ی اطمینان $error_{\mathcal{D}}(h)$ (برای r = 12 و n = 40) (1.96·0.07)±0.30 است، به راحتی می‌توان بازه‌ی اطمینان 68% را نیز محاسبه کرد، این بازه (1.0·0.07)±0.30 است. توجه دارید که منطقی است که بازه‌ی 68% از بازه‌ی 95% کوچک‌تر باشد زیرا که احتمال وجود $error_{\mathcal{D}}(h)$ با کاهش طول بازه کاهش می یابد.

رابطه‌ی 5.1 نحوه‌ی محاسبه‌ی بازه های اطمینان یا ستون‌های خطا[1] را برای تخمین $error_{\mathcal{D}}(h)$ بر اساس مقدار $error_S(h)$ نشان می‌دهد. در استفاده از این رابطه همیشه باید در نظر داشت که این مقادیر فقط برای فرضیه های گسسته مقدار مطرح می‌شود و مجموعه‌ی نمونه‌ی S به صورت تصادفی با همان توزیع احتمالی که نمونه های جدید با آن انتخاب می‌شوند انتخاب شده و همچنین فرض می‌شود که داده‌ها از فرضیه ای که با آن تست می‌شوند مستقلند. همچنین باید در نظر داشت که این رابطه فقط تخمین بازه‌ی اطمینان است، با این وجود این تخمین زمانی که تعداد نمونه های S بیش از 30 است و $error_S(h)$ خیلی نزدیک 0 یا 1 نیست دقت خوبی دارد. به عبارت دقیق‌تر این رابطه زمانی که نامساوی زیر صادق است دقت خوبی دارد:

$$n\, error_S(h)(1 - error_S(h)\,) \geq 5$$

در بالا خلاصه‌ی فرایند محاسبه‌ی بازه های اطمینان برای فرضیه های گسسته مقدار آورده شده است. قسمت بعدی توجیه آماری این فرایند را بیان می‌کند.

5.3 اساس تئوری نمونه برداری

در این قسمت ایده های اصلی آماری و تئوری نمونه برداری را معرفی خواهیم کرد، این ایده‌ها شامل توزیع‌های احتمال، امید ریاضی، واریانس، توزیع‌های دو جمله ای و نرمال و بازه های یک طرفه و دو طرفه می‌شود. آشنایی ابتدایی با این مفاهیم برای درک ارزیابی فرضیه‌ها و الگوریتم‌های یادگیری اساسی است. از آن مهم‌تر اینکه این مفاهیم محیطی برای درک مشکلات یادگیری ماشین مثل overfit و رابطه‌ی بین تعمیم موفق و تعداد نمونه‌ها آموزشی فراهم می‌کنند. خوانندگانی که از قبل با این مفاهیم آشنایی دارند می‌توانند بدون لطمه وارد شدن به همبستگی کتاب بدون خواندن این قسمت رد شوند. مفاهیم کلیدی‌ای که در این بخش معرفی می‌شوند خلاصه‌وار در جدول 5.2 آمده‌اند.

- متغیر تصادفی را می‌توان نام یک آزمایش با خروجی تصادفی در نظر گرفت. مقدار این متغیر خروجی آزمایش است.
- توزیع احتمال برای متغیر تصادفی Y احتمال $\Pr(Y = y_i)$ را که احتمال اینکه متغیر تصادفی Y مقدار y_i را داشته باشد برای تمامی y_i ها مشخص می کند.
- مقدار امید، یا میانگین، متغیر تصادفی Y به صورت E[Y]=$\sum_i \Pr(Y = y_i)$ تعریف می‌شود. معمولاً از نماد μ_Y برای نمایش E[Y] استفاده می‌شود.
- واریانس متغیر تصادفی Y به صورت Var(Y)=$E[(Y - \mu_Y)^2]$ تعریف می‌شود. واریانس پهنای توزیع را حول میانگین بیان

[1] error bars

می‌کند.

- انحراف از معیار متغیر تصادفی Y به صورت $\sqrt{Var(Y)}$ تعریف می‌شود. از نماد σ_Y نیز برای نمایش انحراف معیار متغیر تصادفی Y استفاده می‌کنند.
- توزیع دو جمله ای احتمال مشاهده‌ی r بار مشاهده‌ی شیر در پرتاب n سکه‌ی مستقل را به شرط اینکه در هر پرتاب احتمال شیر آمدن p باشد را معلوم می‌کند.
- توزیع نرمال، توزیع احتمالی زنگی شکل است که توزیع احتمال بسیاری از پدیده های طبیعی است.
- قضیه حد مرکزی قضیه ای است که می‌گوید مجموع تعداد زیادی توزیع یکسان از یک متغیر تصادفی تقریباً توزیع نرمال خواهد داشت.
- از تخمین زننده متغیر تصادفی Y برای تخمین پارامتر p از مجموعه ای از نمونه‌ها استفاده می‌شود.
- بایاس تخمینی متغیر تصادفی Y برای تخمین زننده‌ی پارامتر p با کمیت (E[Y]-p) سنجیده می‌شود. تخمین زننده ای بدون بایاس است که این کمیت برایش صفر باشد.
- بازه‌ی اطمینان N% برای تخمین پارامتر p بازه این است که با احتمال N%، p را در بر خواهد گرفت.

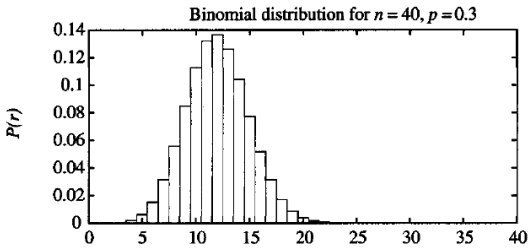
جدول 5.2 تعاریف و حقایق پایه ای آمار

5.3.1 تخمین خطا و تخمین ویژگی‌های توزیع دو جمله ای

اختلاف بین خطای واقعی و خطای نمونه ای دقیقاً چه رابطه ای با اندازه‌ی مجموعه‌ی نمونه‌ها دارد؟ این سؤال نمونه ای از یک مشکل آماری است: مشکل کلی خطا در تخمین ویژگی‌های کلی جامعه با داشتن مجموعه ای تصادفی از اعضای جامعه. در مسئله‌ی ما این ویژگی اشتباه دسته بندی شدن توسط فرضیه‌ی h است.

جواب این سؤال در توجه به این حقیقت است که اندازه گیری خطای نمونه ای از این طریق، آزمایشی با خروجی تصادفی است. زیرا که ابتدا مجموعه‌ی S را با n نمونه‌ی مستقل با توزیع $\mathcal{D}$ تصادفی انتخاب می‌کنیم و خطای نمونه ای $error_S(h)$ را از روی این مجموعه محاسبه می‌کنیم. همان طور که در قسمت قبلی هم گفته شد، اگر آزمایش را به دفعات زیاد و هر دفعه با مجموعه‌ی S_i که n نمونه دارد تکرار کنیم، انتظار خواهیم داشت که مقادیر مختلف $error_{S_i}(h)$ مساوی نباشند. در چنین شرایطی، می‌توانیم بگوییم که $error_{S_i}(h)$ (خروجی i امین آزمایش) یک متغیر تصادفی است. در کل، می‌توان به متغیر تصادفی به چشم آزمایشی با خروجی تصادفی نگاه کرد. مقدار متغیر تصادفی نتیجه‌ی مشاهده شده‌ی آزمایش تصادفی است.

فرض کنید که می‌خواهیم k آزمایش تصادفی برای اندازه گیری متغیر های تصادفی $error_{S_1}(h)$، $error_{S_2}(h)$، ...، $error_k(h)$ انجام دهیم. و فرض کنید خروجی این آزمایش‌ها را در نموداری مستطیلی و بر اساس تعداد تکرار میزان خطا رسم می‌کنیم. با افزایش مقدار k، نمودار مذکور به نمودار توزیع جدول 5.3 نزدیک خواهد شد. این نمودار توزیع احتمال خاصی به نام توزیع دو جمله ای را نشان می‌دهد.



توزیع احتمال دو جمله ای برای r بار شیر آمدن در آزمایشی که n سکه‌ی مجزا پرتاب می‌شوند، احتمال شیر آمـدن در هـر کـدام از سکه‌ها p است. تابع توزیع به شکل زیر تعریف می‌شود:

$$P(r) = \frac{n!}{r!\,(n-r)!}\, p^r (1-q)^{n-r}$$

اگر متغیر تصادفی X از توزیع دو جمله ای پیروی کند:

- احتمال اینکه Pr(X=r) (X مقدار r را بگیرد) با P(r) مشخص می‌شود.
- امید X ، یا همان میانگین X، E[X] از رابطه‌ی زیر بدست خواهد آمد:

  E[X]=np

- واریانس X، Var(X) از رابطه‌ی زیر بدست خواهد آمد:

  Var(X)=np(1-p)

- انحراف معیار X، σ_X از رابطه‌ی زیر بدست خواهد آمد:

$$\sigma_X = \sqrt{np(1-p)}$$

برای n های به اندازه‌ی کافی بزرگ توزیع دو جمله ای تقریباً نزدیک به توزیع نرمال با همان واریانس و میانگین خواهد بود (جـدول 5.4). توصیه می‌شود که فقط زمانی که $np(1-p) \geq 5$ باشد از این تقریب استفاده کرد.

جدول 5.3 توزیع دو جمله ای

5.3.2 توزیع دو جمله ای

یکی از روش‌های خوب درک یادگیری توزیع دو جمله ای بررسی مسئله‌ی ذیل است. سکه ای معیوب (کـج) بـه شـما داده مـی‌شـود و از شـما خواسته می‌شود تا احتمال اینکه سکه پس از پرتاب شیر بیاید را حساب کنید. بیایید احتمال شیر آمدن سکه‌ی معیوب را با p نشان دهـیم. شـما سکه را n بار پرتاب می‌کنید، از این n بار r بار شیر می‌آید. یک تخمین منطقی از p مقدار r/n است. توجه دارید که اگر این آزمایش را تکـرار کنیم و n بار سکه را پرتاب کنیم، انتظار نمی‌رود که تعداد شیر های آمده دقیقاً r قبلی باشد، پس بنابراین مقدار p بدست آمده نیـز بـا مقـدار p قبلی یکی نخواهد بود. توزیع دو جمله ای احتمال وقوع مقدار r (بین 0 تا n) را در n پرتاب مشخص می‌کند، این احتمال با فرض اینکه تمامی پرتاب‌ها مستقل و احتمال شیر آمدن دقیقاً p است محاسبه می‌شود.

جالب است که بدانیم، تخمین p از یک دسته پرتاب مشابه تخمین $error_{\mathcal{D}}(h)$ بر اساس یک دسته نمونه است. شیر بودن یک پرتاب مشابه اشتباه دسته بندی شدن یک نمونه‌ی تصادفی (با توزیع $\mathcal{D}$) است. احتمال p یا همان احتمال شیر آمدن یک پرتاب مشابه احتمال اشتباه دسته بندی شدن یک نمونه تصادفی (یا همان $error_{\mathcal{D}}(h)$) است. تعداد r یا همان تعداد شیرها در n پرتاب نیز مشابه تعداد دسته بندی‌های اشتباه نمونه‌ها از n نمونه‌ی تصادفی است. بنابراین r/n مشابه $error_S(h)$ است و مسئله‌ی تخمین p در سکه نیز مشابه مسئله‌ی تخمین $error_{\mathcal{D}}(h)$ در فرضیه‌ها است. توزیع دو جمله ای فرم کلی توزیع احتمال را برای متغیر تصادفی r مشخص می‌کند حال فرقی نمی‌کند که این r تعداد شیرها باشد یا تعداد دسته بندی‌های اشتباه. فرم دقیق‌تر توزیع دو جمله ای به تعداد نمونه‌ها و p (یا همان $error_{\mathcal{D}}(h)$) وابسته است.

شرایطی که توزیع دو جمله ای در آن صادق است:

1. مبنای کار یک آزمایش (مثل پرتاب سکه) است که خروجی‌اش به عنوان متغیر تصادفی (مثل Y) است. مقدار تصادفی Y می‌تواند فقط دو مقدار داشته باشد (برای مثال Y=1 برای شیر و Y=0 برای خط)
2. احتمال اینکه Y=1 شود در هر آزمایش مبنا به طور مستقل مقدار ثابت p است. پس بنابراین احتمال Y=0 (1-p) خواهد بود. معمولاً p مجهول است و هدف یافتن تخمینی از p است.
3. سری‌ای از آزمایش‌های مبنا پشت سر هم انجام می‌شود (مثل پرتاب‌های سکه) و سری‌ای از متغیر های تصادفی هم ارزش مثل Y_1، Y_2، ...، Y_n را ایجاد می‌کند. اگر R تعداد آزمایش‌ها که در آن‌ها Y_i=1 است در نظر بگیریم خواهیم داشت:

$$R \equiv \sum_{i=1}^{n} Y_i$$

4. احتمال اینکه متغیر تصادفی R مقدار r باشد (احتمال اینکه دقیقا r بار شیر بیاید) بر اساس توزیع دو جمله ای به صورت زیر است:

$$\Pr(R = r) = \frac{n!}{r!\,(n-r)!} p^r (1-p)^{n-r} \qquad (5.2)$$

نموداری از این رابطه در جدول 5.3 آمده بود.

توزیع دو جمله ای احتمال تعداد خطای r در میان n نمونه را مشابه احتمال r بار شیر آمدن در میان n آزمایش پرتاب سکه توصیف می‌کند.

5.3.3 میانگین و واریانس

دو خاصیتی که معمولاً در مورد متغیر های تصادفی مطرح می‌شود امید (مقدار انتظاری یا میانگین) و واریانس است. مقدار امید، میانگین مقادیر تصادفی بدست آمده بعد از آزمایش‌های بسیار است. به عبارت دقیق‌تر:

تعریف: اگر Y یک متغیر تصادفی باشد که مقادیر y_1، ...، y_n را بپذیرد امید Y، E[Y] به صورت زیر تعریف می شود:

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i) \qquad (5.3)$$

برای مثال اگر متغیر تصادفی Y مقدار 1 را با احتمال 0.7 و مقدار 2 را با احتمال 0.3 بپذیرد مقدار امید (1·0.7+2·0.3=1.3) خواهد بـود. در توزیع دو جمله ای این تعریف به شکل زیر تغییر شکل پیدا می‌کند:

$$E[Y] = np \tag{5.4}$$

در این رابطه n و p پارامتر های توزیع دو جمله ای در رابطه‌ی 5.2 هستند.

کمیت دوم مطرح "پهنا[1]" یا "میزان پخشی[2]" توزیع احتمال است و میزان دور بودن احتمالی متغیر تصادفی از میانگین را نشان می‌دهد.

تعریف: واریانس یک متغیر تصادفی Y، Var[Y] به فرم زیر تعریف می‌شود:

$$Var[Y] \equiv E[(Y - E[Y])^2] \tag{5.5}$$

واریانس مجموع مربعات خطای انتظاری را با استفاده از امید Y، E[Y]، پیدا می‌کند. جزر واریانس را انحراف معیار Y مـی‌نامنـد و بـا σ_Y نشـان می‌دهند.

تعریف: انحراف معیار متغیر تصادفی Y، σ_Y، به صورت زیر تعریف می‌شود:

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]} \tag{5.5}$$

در شرایطی که متغیر تصادفی Y توزیع دو جمله ای داشته باشد، واریانس و انحراف از معیار به فرم زیر خواهند بود:

$$Var[Y] = np(1-p)$$

$$\sigma_Y = \sqrt{np(1-p)} \tag{5.7}$$

5.3.4 تخمین زننده‌ها، بایاس و واریانس

حال که نشان داده‌ایم که متغیر تصادفی $error_S(h)$ از توزیع دو جمله ای پیروی می‌کند، به سؤال اصلی بر می‌گردیم: فرق خطای نمونه ای و خطای واقعی چیست؟

بیایید $error_S(h)$ و $error_D(h)$ را با استفاده از رابطه‌ی 5.2 که توزیع دو جمله ای را بیان می‌کند توصیف کنیم. داریم که

$$error_S(h) = \frac{\mathrm{r}}{\mathrm{n}}$$

$$error_{\mathcal{D}}(h) = p$$

[1] width
[2] spread

در این رابطه n تعداد نمونه های مجموعه‌ی S و r تعداد دسته بندی‌های اشتباه h از مجموعه‌ی S است و p نیز احتمال دسته بندی اشتباه h از نمونه ای انتخاب شده با توزیع $\mathcal{D}$ است.

متخصصان $error_S(h)$ را تخمین زننده ای[1] از خطای واقعی $error_D(h)$ می‌نامند. در کل، تخمین زننده‌ی یک مقدار تصادفی برای تخمین ویژگی‌های جمعیت آن متغیر تصادفی به کار می‌رود. اولین سؤالی که درباره‌ی هر تخمین زننده مطرح می‌شود این است که آیا تخمین زننده در میانگین تخمین درستی به ما می‌دهد؟ بایاس تخمین را به عنوان اختلاف بین مقدار امید تخمین زننده و مقدار واقعی متغیر تصادفی تعریف می‌کنیم.

تعریف: بایاس تخمین[2] برای تخمین زننده‌ی Y از پارامتر p به صورت

$$E[Y] - p$$

تعریف می‌شود.

اگر مقدار بایاس تخمین زننده صفر باشد می‌گوییم که Y یک تخمین زننده‌ی بدون بایاس از p است. توجه داشته این حالتی است که پس از تعداد زیادی آزمایش تصادفی میانگین مقدار تصادفی به امید تخمین زننده میل کند.

آیا $error_S(h)$ تخمین زننده‌ی بدون بایاسی از $error_{\mathcal{D}}(h)$ است؟ بله، زیرا که مقدار امید r در توزیع دو جمله ای np است (رابطه‌ی 5.4). حال چون که n ثابت است، پس مقدار امید r/n همان p است.

دو نکته‌ی قابل توجه در بایاس تخمینی وجود دارد. اول، همان طور که در ابتدای این فصل نیز گفته شد، بررسی فرضیه‌ها بر روی نمونه های آموزشی، تخمینی بایاس دار از خطای فرضیه به ما می‌دهد، این دقیقا همان نکته ای است که بایاس تخمین به آن اشاره می‌کند. برای اینکه $error_S(h)$ تخمینی بدون بایاس از $error_D(h)$ به ما بدهد، باید فرضیه‌ی h و نمونه های S باید مستقل باشند. دوم اینکه این مفهوم نباید با بایاس استقرایی که در فصل 2 بیان شد اشتباه گرفته شود. بایاس تخمینی یک مقدار عددی است در حالی که بایاس استقرایی دسته ای از پیش فرض‌ها[3] است.

ویژگی مهم دیگر هر تخمین زننده مقدار واریانس آن است. با داشتن انتخاب بین تخمین زننده های بدون بایاس مختلف، قابل درک است که تخمین زننده ای را انتخاب کنیم که کمترین مقدار واریانس را داشته باشد. با تعریفی که از واریانس ارائه شد، این انتخاب باعث می‌شود که خطای انتظاری بین تخمین و مقدار واقعی به کمترین مقدار برسد.

برای تصور این مفاهیم، فرض کنید که می‌خواهیم فرضیه ای را بررسی کنیم که r=12 خطا بر روی نمونه‌هایی با تعداد n=40 دارد. اگر $error_S(h)$ یک تخمین زننده‌ی بدون بایاس از $error_{\mathcal{D}}(h)$ باشد، و داشته باشیم $error_S(h) = \frac{r}{n} = 0.3$. واریانس این تخمین مستقیماً به واریانس مقدار r وابسته است، زیرا که n عددی ثابت است. حال چون r با توزیع دو جمله ای انتخاب می‌شود برای واریانس از رابطه‌ی 5.7 داریم: np(1-p). متأسفانه هنوز مقدار p مجهول است، اما می‌توان بجای آن از تخمین r/n مان از p استفاده کنیم. پس

[1] estimator
[2] estimation bias
[3] assertion

واریانس r خواهد بود 40·0.3(1-0.3)=8.4 پس مقدار انحراف معیار $\sqrt{8.4} \approx 2.9$. پس انحراف معیار r/n، 2.9/40=.07 خواهد بود. به طور خلاصه، $error_S(h)$ در این مثال مقدار امید 0.30 با انحراف معیار تقریبا 0.07 است (تمرین 5.1).

در کل، با داشتن خطای r در n نمونه‌ی موجود مستقل، از رابطه‌ی زیر بدست می‌آید:

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}} \tag{5.8}$$

که با تخمین r/n=$error_S(h)$ برای p به رابطه‌ی زیر تبدیل می‌شود:

$$\sigma_{error_S(h)} = \sqrt{\frac{error_S(h)\big(1-error_S(h)\big)}{n}} \tag{5.9}$$

۵.3.5 بازه‌ی اطمینان

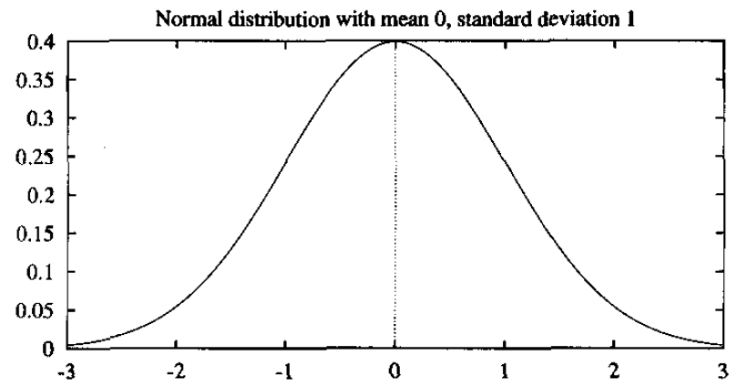
یکی از راه‌های معمول توصیف عدم قطعیت یک تخمین توجه به بازه و احتمالی است که انتظار می‌رود که مقدار واقعی در این بازه باشد است. چنین تخمینی، تخمین بازه‌ی اطمینان نامیده می‌شود.

تعریف: بازه‌ی اطمینان N% برای پارامتر p بازه ای است که N% احتمال می‌رود که شامل p باشد.

برای مثال، اگر مثل مثال بالا r=12 و n=40 باشد و نمونه‌ها نیز از فرضیه مستقل باشند، می‌توان گفت که با احتمال 95٪ مقدار $error_D(h)$ در بازه‌ی 0.30±0.14 است.

بازه های اطمینان $error_D(h)$ چگونه بدست می‌آیند؟ جواب در این حقیقت نهفته است که توزیع احتمال دو جمله ای بر $error_S(h)$ حاکم است. میانگین این توزیع مقدار $error_D(h)$ است و انحراف معیار نیز از رابطه‌ی 5.9 بدست می‌آید. بنابراین، برای بدست آوردن بازه‌ی 95٪ فقط نیاز است که بازه را حول مقدار میانگین $error_D(h)$ بگیریم تا 95٪ از کل احتمال را در بر بگیرد. این بازه، بازه ای حول $error_D(h)$ که در 95% موارد $error_S(h)$ درون آن قرار می گیرد.

برای عدد معلوم N چگونه می‌توان اندازه‌ی بازه‌ی N% احتمال را بدست آورد؟ متأسفانه، این محاسبه برای توزیع احتمال دو جمله ای زمان بر است. خوشبختانه، با وجود زمان بری، در اکثر موارد تقریب خوبی برای بازه بدست می‌آید، زیرا که با تعداد نسبتاً زیاد نمونه توزیع به توزیع نرمال میل می‌کند. توزیع نرمال (که در جدول 5.4 نیز آمده) شاید خوش تعریف‌ترین توزیع احتمال باشد. همان طور که در جدول 5.4 نیز نشان داده شده، توزیع نرمال، توزیعی زنگی شکل حول میانگین μ و با انحراف معیار σ است. زمانی که تعداد n نسبتاً زیاد باشد، توزیع دو جمله ای به توزیع نرمالی با همان میانگین و انحراف معیار میل می‌کند.



توزیع نرمال (یا توزیع گوس)، توزیعی زنگی شکل است که توسط رابطه‌ی زیر تعریف می‌شود:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

اگر متغیر تصادفی X از توزیع نرمال پیروی کند:

- احتمال اینکه X در بازه‌ی (a,b) باشد از رابطه‌ی زیر بدست خواهد آمد:

$$\int_a^b p(x)dx$$

- امید یا میانگین X، E[X]:

$$E[X] = \mu$$

- واریانس X، Var(X):

$$Var(X) = \sigma^2$$

- انحراف معیار X، σ_X

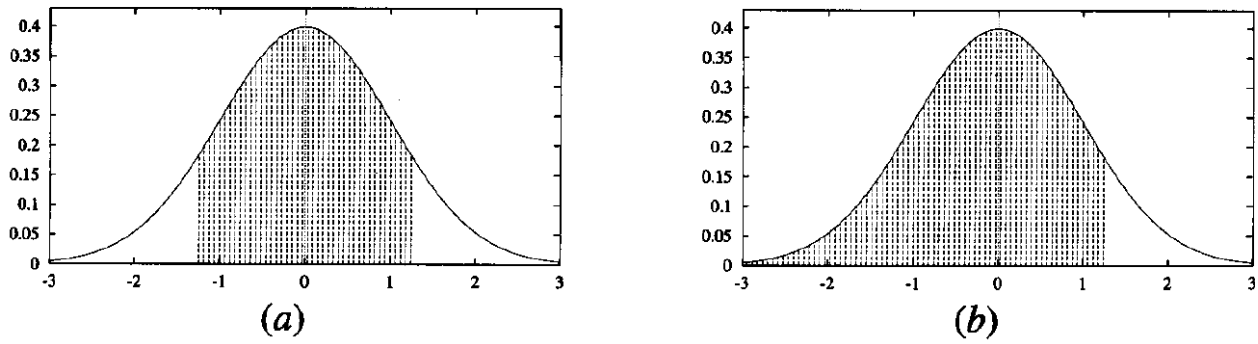
$$\sigma_X = \sigma$$

قضیه‌ی حد مرکزی[1] (در بخش 5.4.1) نشان می‌دهد که مجموع متغیر های تصادفی را با توزیع دلخواه می‌توان با توزیع نرمال بررسی کرد.

جدول 5.4 توزیع نرمال یا توزیع گوس.

یکی از دلایلی ترجیح توزیع نرمال این است که می‌توان به راحتی بازه ای که N% احتمال را در بر می‌گیرد پیدا کرد. این بازه دقیقا همان بازه‌ی N% احتمال ماست و در عمل جدول 5.1 نیز از این حقیقت بدست آمده. ثابت z_N که در جدول 5.1 آمده بود، نصف پهنای بازه‌ی اطمینان است (فاصله‌ی بین میانگین و یکی از طرفین بازه) که بر مقدار انحراف معیار تقسیم می‌شود. شکل (a) 5.1 این بازه را برای $z_{.80}$ نشان می‌دهد.

[1] central limit



شکل 5.1 توزیع نرمال با میانگین 0 و انحراف معیار 1.

(a) با احتمال 80٪ متغیر تصادفی در بازه‌ی از دو طرف محدود [-1.28,1.28] قرار می‌گیرد. توجه داشته باشید که $z_{.80} = 1.28$ *پس با احتمال 10٪ متغیر تصادفی در سمت راست و با احتمال 10٪ متغیر تصادفی در سمت چپ این بازه قرار می‌گیرد. (b) با احتمال 90٪ متغیر تصادفی در بازه‌ی از یک طرف محدود [-∞,1.28] قرار می‌گیرد.*

به طور خلاصه، اگر متغیر تصادفی Y از توزیع نرمال با میانگین μ و انحراف معیار σ پیروی کند، مقدار تصادفی y برای Y به احتمال N% در بازه‌ی زیر قرار می‌گیرد:

$$\mu \pm z_N \sigma \tag{5.10}$$

به طور مشابه، مقدار میانگین μ با احتمال N% در بازه‌ی زیر قرار می‌گیرد:

$$y \pm z_N \sigma \tag{5.11}$$

این واقعیت را می‌توان به سادگی با واقعیت‌های کلی قبلی ذکر شده در مورد بازه‌ی N% در توابع گسسته مقدار ترکیب کرد (رابطه‌ی 5.1). ابتدا اینکه می‌دانیم $error_S(h)$ از توزیع دو جمله ای پیروی می‌کند که میانگین آن $error_D(h)$ است و انحراف معیارش نیز از رابطه‌ی 5.9 بدست می‌آید. دوم اینکه می‌دانیم که برای زمانی که تعداد n به اندازه‌ی کافی بزرگ باشد توزیع دو جمله ای را می‌توان با تقریب خوبی با توزیع نرمال تقریب زد. سوم اینکه رابطه‌ی 5.11 روش پیدا کردن بازه‌ی اطمینان N% را با توجه به توزیع نرمال مشخص می‌کند. بنابراین، با جایگزینی میانگین و انحراف معیار $error_S(h)$ در رابطه‌ی 5.11 برای توابع گسسته مقدار به رابطه‌ی 5.1 می‌رسیم.

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)\left(1 - error_{S(h)}\right)}{n}}$$

دو نکته‌ی مهم در تخمین این رابطه به شرح زیرند:

1. در تخمین انحراف معیار σ برای $error_S(h)$، ما از $error_S(h)$ به جای $error_D(h)$ استفاده کردیم (در نتیجه گیری رابطه‌ی 5.9 از رابطه‌ی 5.8)
2. توزیع دو جمله ای را با توزیع نرمال تخمین زده‌ایم.

این دو تقریب تا زمانی که n≥30 و np(1-p)≥5 تقریب‌های خوبی هستند. برای مقادیر کمتر n بهتر است از جدولی با مقادیر توزیع دو جمله ای به جای توزیع نرمال استفاده کنیم.

5.3.6 بازه های یک طرفه و دو طرفه

توجه دارید که بازه‌ی اطمینان ذکر شده مرز دوطرفه[1] دارد؛ زیرا که کمیت تخمین زده شده را هم از بالا و هم از پایین محدود کرده است. در بعضی موارد، علاقه‌ی ما فقط به یکی از این دو مرز است (مرز یک طرفه[2]). برای مثال ممکن است جواب سؤال "احتمال اینکه $error_D(h)$ حداقل از U بیشتر باشد؟" برایمان مهم باشد. چنین سؤال‌هایی که مرز یک طرفه دارند طبیعتاً زمانی ایجاد می‌شوند که ما به محدود کردن حداکثر خطای h علاقه داریم و اینکه خطا از آن مقدار بسیار کوچک‌تر باشد برایمان مهم نیست.

تغییر کوچکی در فرایند بالا آن‌را به روش پیدا کردن مرز یک طرفه تبدیل می‌کند. نکته‌ی اساسی این تغییر این است که توزیع نرمال حول میانگینش متقارن پخش شده است. به همین خاطر می‌توان هر بازه‌ی اطمینان با مرز دوطرفه را به بازه ای با مرز یک طرفه تبدیل کرد (مثل شکل (b) 5.1). اگر بازه‌ی اولیه اطمینان %100(1-α) داشته باشد، بازه های یک طرفه‌ی فقط محدود از بالا اطمینان %100(1-α/2) خواهند داشت. بازه‌ی یک طرفه‌ی فقط محدود از پایین نیز همین اطمینان را خواهد داشت. در اینجا α احتمال این است که متغیر تصادفی خارج بازه‌ی مزبور باشد. به عبارت دیگر، α احتمال این است که متغیر تصادفی در قسمت‌هایی از شکل (a) 5.1 قرار بگیرد که هاشور نخورده‌اند. متناسباً مقدار α/2 نیز احتمال این است که متغیر تصادفی در قسمت هاشور نخورده‌ی شکل (b) 5.1 قرار بگیرد.

برای تصور، دوباره فرض کنید که فرضیه ای با r=12 خطا بر روی مجموعه ای با n=40 نمونه داریم که نمونه‌ها از فرضیه مستقلند. همان طور که بالاتر نیز گفته شد، این اطلاعات بازه‌ی 95% (دوطرفه) ی 0.30±0.14 را مشخص می‌کند. در اینجا، 100(1-α)=95% پس α=0.05. بنابراین بدون اضافه کردن هیچ پیش فرض اضافه ای، می‌توانیم بگوییم که با اطمینان 100(1-α/2)=97.5%، $error_{\mathcal{D}}(h)$ حداکثر 0.30+0.14=.44 است. بنابراین، مرزی یک طرفه برای $error_{\mathcal{D}}(h)$ با اطمینان دو برابر نسبت به مرز دوطرفه خواهیم داشت (تمرین 5.3).

5.4 روش کلی برای استخراج بازه های اطمینان

در قسمت قبل چگونگی بدست آوردن بازه های تخمین برای حالت خاص: تخمین $error_{\mathcal{D}}(h)$ برای توابع گسسته مقدار h بر پایه‌ی مجموعه ای با n نمونه توضیح داده شد. روشی که آنجا ارائه شد روشی کلی‌تر برای استفاده در تعداد کثیری از مشکلات تخمین شرح می‌دهد. در کل، می‌توان مسئله‌ی فوق را تخمین میانگین (مقدار امید) یک مجموعه بر اساس زیر مجموعه ای با n عضو دانست. فرایند کلی مراحل زیر را در بر می‌گیرد:

1. معلوم کردن پارامتر p ای که از مجموعه می‌خواهیم تخمین بزنیم، برای مثال $error_{\mathcal{D}}(h)$.
2. تعریف تخمین زننده‌ی Y، مثل ، مثل $error_S(h)$. بهتر است این تخمین زننده واریانس کم داشته و بدون بایاس باشد.
3. مشخص کردن توزیع احتمال $\mathcal{D}_Y$ که کار تخمین زننده‌ی Y را کنترل می‌کند. مشخص کردن این توزیع شامل مشخص کردن واریانس و میانگین نیز می‌شود.

---

[1] two sided bound
[2] one side bound

4. مشخص کردن بازه‌ی N% با پیدا کردن مقادیر آستانه‌ی L و U که N% از جرم احتمال توزیع $\mathcal{D}_Y$ بین دو مقدار L و U قرار بگیرد.

در بخش‌های بعدی این فصل، از این روش کلی برای مسائل تخمین مختلفی که در یادگیری ماشین مطرح است استفاده می‌کنیم. با این وجود، ابتدا بیایید نتیجه‌ی اساسی قضیه‌ی حد مرکزی را بررسی کنیم.

5.4.1 قضیه‌ی حد مرکزی

یکی از حقیقت‌هایی که برای پیدا کردن بازه های اطمینان مورد استفاده قرار می‌گیرد قضیه‌ی حد مرکزی است. دوباره شرایط کلی، n متغیر تصادفی مستقل $Y_1, \dots, Y_n$ که از توزیع احتمال مجهول خاصی پیروی می‌کنند (مثل n بار پرتاب یک سکه) را در نظر بگیرید. فرض کنید μ میانگین این توزیع مجهول باشد و σ نیز انحراف معیار آن باشد. این متغیر های Y_i مستقل و به طور یکسان توزیع[1] شده‌اند زیرا که هر کدام از آن‌ها آزمایش مجزایی را توصیف می‌کند، و هر کدام از همان توزیع احتمال پیروی می‌کنند. در تخمین میانگین μ تابع توزیع حاکم بر Y_i ها از همان تعریف میانگین استفاده می‌کنیم $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$. (مثل نسبت شیرها به کل پرتاب‌ها). قضیه‌ی حد مرکزی می‌گوید که توزیع احتمال حاکم بر $\bar{Y}_n$ مستقل از اینکه توزیع احتمال Y_i ها چه باشد با n→∞ به توزیع نرمال میل می‌کند. علاوه بر این توزیعی که $\bar{Y}_n$ را کنترل می‌کند میانگین μ و انحراف معیار $\frac{\sigma}{\sqrt{n}}$ خواهد داشت. به عبارت دیگر،

قضیه‌ی 5.1 قضیه‌ی حد مرکزی. فرض کنید که دسته ای متغیر تصادفی مستقل و به طور یکسان توزیع شده‌ی $Y_1, \dots, Y_n$ را داریم که هر کدام از این متغیر های تصادفی با توزیع احتمالی که میانگین μ و واریانس σ^2 دارد کنترل می‌شوند. اگر $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$ تعریف کنیم، و n→∞ توزیع احتمال حاکم بر

$$\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

به توزیع نرمال با میانگین صفر و انحراف معیار 1 میل خواهد کرد.

این حقیقت بسیار جالبی است، توزیع احتمال حاکم بر $\bar{Y}_n$ بدون دانستن توزیع احتمال حاکم بر Y_i ها معلوم می‌گردد! علاوه بر آن، قضیه‌ی حد مرکزی روشی برای پیدا کردن واریانس و میانگین تابع توزیع Y_i ها از واریانس و میانگین تابع توزیع $\bar{Y}_n$ ارائه می کند.

قضیه‌ی حد مرکزی، حقیقتی پر کاربرد است، زیرا که به هر صورت که یک تخمین زننده برای تخمین میانگین چیزی تعریف کنیم ($error_S(h)$ تخمین زننده‌ی میانگین خطاست)، برای n های به اندازه‌ی کافی بزرگ آن‌را می‌توان با توزیع نرمال تخمین زد. حال اگر واریانس این توزیع نرمال (تخمینی) را بدانیم می‌توانیم با استفاده از رابطه‌ی 5.11 برای محاسبه‌ی بازه های اطمینان استفاده کنیم. یک تقریب متداول این است که زمانی می‌توانیم از تقریب نرمال استفاده کنیم که داشت باشیم $n \geq 30$. توجه دارید که در قسمت قبلی از چنین توزیع نرمالی برای تخمین توزیع دوجمله ای که $error_S(h)$ را توصیف می‌کرد استفاده کردیم.

[1] identically distributed

5.5 تفاوت خطا های دو فرضیه

حالتی را تصور کنید که دو فرضیه‌ی h_2 و h_1 را برای تابع هدف گسسته مقداری داریم. فرضیه‌ی h_1 بر روی مجموعه‌ی S_1 که شامل n_1 نمونه است، و فرضیه‌ی h_2 بر روی مجموعه‌ی S_2 که شامل n_2 نمونه است بررسی شده است. فرض کنید که می‌خواهیم تفاوت بین خطای واقعی این دو فرضیه را تخمین بزنیم.

$$d \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

در اینجا ما از فرایند چهار مرحله ای ارائه شده در ابتدای بخش 5.4 برای بدست آوردن بازه‌ی اطمینان برای d استفاده می‌کنیم. با معلوم کردن d به عنوان پارامتری که می‌خواهیم تخمین بزنیم، باید یک تخمین زننده معرفی کنیم. تنها انتخاب ممکن و واضح برای تخمین زننده‌ی d اختلاف بین خطا های نمونه ای است که با $\hat{d}$ نشان می‌دهیم:

$$\hat{d} \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

با وجود اینکه اینجا اثبات نمی‌کنیم اما می‌توان نشان داد که $\hat{d}$ تخمینی بدون بایاس از d ارائه می‌کند؛ $E[\hat{d}] = \mathrm{d}$.

اما $\hat{d}$ از چه توزیع احتمالی پیروی می‌کند؟ با استفاده از آنچه در قسمت‌های قبلی گفته شد، اگر n_2 و n_1 به اندازه‌ی کافی بزرگ باشند (هر دو بزرگ‌تر از 30 باشند) هر دو متغیر تصادفی $error_{S_2}(h_2)$ و $error_{S_1}(h_1)$ از توزیع نرمال پیروی خواهند کرد. چون تفاوت دو توزیع نرمال نیز توزیعی نرمال است، $\hat{d}$ نیز توزیعی نرمال با میانگین d خواهد داشت. همچنین می‌توان نشان داد که واریانس این توزیع مجموع واریانس توزیع‌های $error_{S_2}(h_2)$ و $error_{S_1}(h_1)$ است. با استفاده از رابطه‌ی 5.9 برای تخمین واریانس هر یک از این دو توزیع داریم که:

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)\left(1 - error_{S_1}(h_1)\right)}{n_1} + \frac{error_{S_2}(h_2)\left(1 - error_{S_2}(h_2)\right)}{n_2} \quad (5.12)$$

حال که توزیع احتمالی که $\hat{d}$ را کنترل می‌کند را مشخص کرده‌ایم، به راحتی می‌توان بازه های اطمینان را که از $\hat{d}$ برای d بدست می‌آید مشخص کرد. برای متغیر تصادفی $\hat{d}$ که از توزیع نرمالی با میانگین d و واریانس σ^2 پیروی می‌کند بازه‌ی اطمینان $\hat{d} \pm z_N \sigma$ را خواهیم داشت. با استفاده از واریانس تخمینی $\sigma_{\hat{d}}^2$ که در بالا محاسبه شد این بازه‌ی اطمینان تخمینی برای d به صورت زیر خواهد بود:

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)\left(1 - error_{S_1}(h_1)\right)}{n_1} + \frac{error_{S_2}(h_2)\left(1 - error_{S_2}(h_2)\right)}{n_2}} \quad (5.13)$$

در این رابطه z_N مقادیری است که از جدول 5.1 استخراج می‌شود. رابطه‌ی بالا بازه‌ی اطمینان دوطرفه ای برای تخمین اختلاف بین خطاهای دو فرضیه به ما می‌دهد. بعضی مواقع ممکن است علاقه‌ی ما به بازه‌ی یک طرفه باشد، محدود کردن بزرگ‌ترین اختلاف خطاها در یک محدوده‌ی خاص. این بازه‌ی اطمینان دوطرفه را می‌توان با همان فرایند قسمت 5.3.6 به بازه های یک طرفه تبدیل کرد.

با این وجود که بررسی بالا در حالتی انجام گرفته که دو فرضیه‌ی h_1 و h_2 روی مجموعه داده های مستقل تست شده‌اند، اما گاهی استفاده از این بازه‌ی اطمینان (رابطه‌ی 5.13) در جایی که h_1 و h_2 هر دو بر روی مجموعه‌ی S (که هنوز از هر دو فرضیه‌ی h_1 و h_2 مستقل است) قابل قبول است. در این حالت می‌توان $\hat{d}$ را به فرم زیر تعریف کرد:

$$\hat{d} \equiv error_S(h_1) - error_S(h_2)$$

این اختلاف در $\hat{d}$ جدید معمولاً کمتر از اختلاف رابطه‌ی 5.12 است، زیرا که دو مجموعه‌ی S_1 و S_2 هر دو S در نظر گرفته شده‌اند. دلیل این کاهش استفاده از یک مجموعه‌ی نمونه ای S برای ارزیابی اختلاف اثر اختلافات تصادفی بین ترکیب S_1 و S_2 را حذف خواهد کرد. در این حالت، بازه‌ی اطمینان رابطه‌ی 5.13 در کل بازه ای محافظه کارانه، اما هنوز درست، خواهد بود.

5.5.1 تست فرضیه[1]

بعضی مواقع، علاقه‌ی ما بیشتر به احتمال درستی یک حدس است تا اینکه بازه های اطمینان را برای پارامتر های حدس داشته باشیم. برای مثال، فرض کنید، علاقه‌ی ما به این سؤال که "با چه احتمالی داریم $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$؟" است. با توجه به آنچه در قسمت‌های گذشته گفتیم، فرض کنید که دو فرضیه‌ی h_1 و h_2 را بر روی دو مجموعه‌ی مستقل S_1 و S_2 با اندازه‌ی مساوی 100 تست می‌کنیم و داریم، $error_{S_1}(h_1) = .30$ و $error_{S_2}(h_2) = .20$، بنابراین اختلاف مشاهده شده $\hat{d} = .10$ خواهد بود. البته با توجه به اختلافات تصادفی در داده های نمونه ای ممکن است چنین نتایجی حتی زمانی که $error_{\mathcal{D}}(h_1) \leq error_{\mathcal{D}}(h_2)$ است نیز مشاهده شود. احتمال اینکه داشته باشیم $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$ با داشتن اینکه اختلاف نمونه ای $\hat{d} = .10$ را داریم چقدر است؟ یا به طور معادل احتمال اینکه $d > 0$ باشد به شرط اینکه $\hat{d} = .10$ چقدر است؟

توجه دارید که احتمال اینکه $\Pr(d > 0)$ مشابه احتمال این است که $\hat{d}$، d را به اندازه‌ی 10 . بیشتر تخمین زده باشد. به عبارت دیگر، احتمال این است که $\hat{d}$ در بازه‌ی اطمینان یک طرفه‌ی $\hat{d} < d + .10$ قرار بگیرد است. در این رابطه d میانگین توزیع احتمال حاکم بر $\hat{d}$، است پس می‌توان رابطه را به صورت $\hat{d} < \mu_{\hat{d}} + .10$ بازنویسی کرد.

به طور خلاصه احتمال $\Pr(d > 0)$ مساوی این احتمال است که $\hat{d}$ در بازه‌ی اطمینان یک طرفه‌ی $\hat{d} < \mu_{\hat{d}} + .10$ قرار داشته باشد است. از آنجایی که در قسمت قبلی توزیع احتمال حاکم بر $\hat{d}$ را محاسبه کرده‌ایم، می‌توان احتمال اینکه $\hat{d}$ در بازه‌ی اطمینان یک طرفه‌ی قرار گیرد با جرم احتمال توزیع $\hat{d}$ در این بازه اندازه گیری خواهد شد.

بیایید این محاسبه را با بازنویسی دوباره‌ی $\hat{d} < \mu_{\hat{d}} + .10$ با انحراف از معیار شروع کنیم. با استفاده از رابطه‌ی 5.12 می‌توان بدست آورد که $\sigma_{\hat{d}} \approx .061$، پس می‌توان بازه‌ی اطمینان را به فرم زیر نوشت،

$$\hat{d} < \mu_{\hat{d}} + 1.64\sigma_{\hat{d}}$$

میزان احتمال متناسب با این بازه‌ی یک طرفه در توزیع نرمال چند است؟ با توجه به جدول 5.1، می‌توان بدست آورد که بازه ای تا 1.64 برابر انحراف حول میانگین برای بازه‌ی دوطرفه احتمال 90٪ دارد. بنابراین، بازه‌ی اطمینان دو طرفه احتمال 95٪ را خواهد داشت.

---

[1] Hypothesis testing

بنابراین، با داشتن مشاهده‌ی $\hat{d} = .10$، احتمال اینکه $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$ تقریباً 95 . است. در واژگان ادبیات آماری، می‌گوییم که این فرضیه که "$error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$" را با اطمینان 95 . می‌پذیریم. یا به طور مشابه ممکن است بگوییم که فرضیه متضاد[1] (یا فرضیه‌ی تهی[2]) را با احتمال 05.=(1-.95) رد می‌کنیم.

5.6 مقایسه‌ی الگوریتم‌های یادگیری

بعضی مواقع مقایسه‌ی عملکرد دو الگوریتم یادگیری L_A و L_B برای ما از مقایسه‌ی دو فرضیه اهمیت بیشتری دارد. آزمون مناسب برای مقایسه‌ی الگوریتم‌های یادگیری چیست و چگونه می‌توان معلوم کرد که تفاوت‌های بدست آمده از نظر آماری قابل توجهند؟ با وجود اینکه بحث‌ها هنوز در این مبحث از یادگیری ماشین داغ است اما ما در اینجا روشی خاص را معرفی خواهیم کرد. بحث در مورد دیگر متد های جایگزین را می‌توانید در (Ditterich 1996) پیدا کنید.

مثل قبل، کار را با تعیین پارامتری که می‌خواهیم تخمین بزنیم آغاز می‌کنیم. فرض کنید قصد داریم مشخص کنیم که کدام یک از دو الگوریتم L_A یا L_B به طور متوسط برای یادگیری تابع هدف f معلوم متناسب‌ترند. یکی از راه‌های تعریف "به طور متوسط" بررسی کارایی نسبی دو الگوریتم بر روی تمامی مجموعه نمونه های n عضوی ممکن که با استفاده از توزیع احتمال نمونه ای $\mathcal{D}$ است. به عبارت دیگر، قصد داریم که مقدار امید خطای بین دو فرضیه را تخمین بزنیم

$$\underset{S\subset\mathcal{D}}{\mathrm{E}}\left[error_{\mathcal{D}}\big(L_A(S)\big) - error_{\mathcal{D}}\big(L_B(S)\big)\right] \qquad (5.14)$$

در این رابطه L(S) فرضیه ای است که با استفاده از متد L از نمونه های S بدست می‌آید، $S\subset\mathcal{D}$ نیز به این معناست که مقدار امید بر روی نمونه های S که با توزیع $\mathcal{D}$ انتخاب می‌شوند محاسبه می‌شود. عبارت بالا مقدار امید اختلاف بین خطا های یادگیری دو متد L_A و L_B را نشان می دهد.

البته در عمل مجموعه ای محدود از نمونه‌ها D_0 در دسترس است و بررسی بین دو متد را بر روی این مجموعه‌ی محدود انجام می‌دهیم. در چنین شرایطی، یکی از روش‌های ساده‌ی تخمین کمیت بالا تقسیم D_0 به دسته‌ی آموزشی S_0 و دسته‌ی تست T_0 است. داده های آموزشی را می‌توان برای آموزش در هر دو روش L_A و L_B به کار برد و از دسته‌ی تست می‌توان برای مقایسه‌ی دقت هر کدام از فرضیه های یاد گرفته شده استفاده کرد. به عبارت دیگر، ما کمیت زیر را محاسبه می‌کنیم:

$$error_{T_0}\big(L_A(S_0)\big) - error_{T_0}\big(L_B(S_0)\big) \qquad (5.15)$$

توجه داشته باشید که این تخمین زننده و کمیت رابطه‌ی 5.14 دو تفاوت کلیدی دارند. ابتدا اینکه در این رابطه از $error_{T_0}(h)$ برای تخمین مقدار $error_{\mathcal{D}}(h)$ استفاده شده است. دوم اینکه در این رابطه فقط تفاوت بین خطاها برای یک مجموعه‌ی آموزشی S_0 به جای کل مجموعه های آموزشی ممکن توزیع $\mathcal{D}$ استفاده شده است.

---

[1] opposite hyposesis
[2] null hyposesis

یکی از راه های بهبود این تخمین زننده‌ی رابطه‌ی 5.15 تقسیم داده های در چندین مرحله D_0 بـه مجموعـه هـای کوچـک‌تـر و اسـتفاده از میانگین خطای بدست آمده از دسته‌ی تست در آزمایش‌های مختلف است. این کار به فرایند نشـان داده شـده در جـدول 5.5 بـرای مقایسـه‌ی خطاهای دو متد یادگیری بر اساس داده های ثابت D_0 می‌انجامد. این فرایند تقسیم ابتدا داده‌ها را به k زیر مجموعه‌ی هم اندازه که هـر کـدام حداقل 30 نمونه دارند تقسیم می‌کند. سپس الگوریتم یادگیری را k بار آموزش می‌دهد و آزمایش می‌کند، در هر یک از این k بار یکی از k زیر مجموعه به عنوان مجموعه‌ی تست مورد استفاده قرار می‌گیرد و بقیه داده‌ها نیز مجموعه‌ی آموزشی خواهند بود. به این ترتیـب، الگـوریتم‌هـای یادگیری بر روی k مجموعه‌ی مجزای تست بررسی می‌شوند و میانگین اختلاف در خطا های $\bar{\delta}$ به عنوان یک تخمین زننده برای اختلاف بـین دو الگوریتم یادگیری انتخاب می‌شود.

کمیت $\bar{\delta}$ که از فرایند جدول 5.5 بدست می‌آید را می‌توان به عنوان تخمینی از کمیت مطلوب رابطه‌ی 5.14 به حساب آورد. حتی می‌توان به $\bar{\delta}$ به دید تخمینی از کمیت زیر نگاه کرد:

$$\underset{S\subset D_0}{\mathrm{E}}\left[error_{\mathcal{D}}\big(L_A(S)\big) - error_{\mathcal{D}}(L_B(S))\right] \tag{5.16}$$

در این رابطه S مجموعه ای دلخواه از $\frac{k-1}{k}|D_0|$ نمونه است که با توزیع یکنواخت از D_0 انتخاب شده‌اند. تنها تفاوت بین این کمیت و کمیت اصلی ما در رابطه‌ی 5.14 این است که این کمیت مقدار امید را بر روی زیرمجموعه ای از داده های موجود D_0 پیدا می‌کند به جـای اینکـه از تمامی نمونه‌ها با توزیع $\mathcal{D}$ استفاده کند.

---

1. داده های موجود D_0 را به k دسته‌ی مجزای $T_1, T_2, \dots, T_k$ با اندازه های مساوی تقسیم کن، اندازه‌ی هر مجموعه بایـد حـداقل 30 باشد.
2. برای تمامی مقادیر i با شروع از 1 و کمتر مساوی از k:

   از T_i برای دسته‌ی تست و از بقیه‌ی داده‌ها برای دسته‌ی آموزشی S_i استفاده کن.

$$S_i \leftarrow \{D_0 - T_i\}$$
$$h_A \leftarrow L_A(S_i)$$
$$h_B \leftarrow L_B(S_i)$$
$$\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$$

3. مقدار $\bar{\delta}$ را از تعریف زیر خروجی بده:

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=1}^{k}\delta_i \tag{T5.1}$$

جدول 5.5 فرایند تخمین تفاوت بین خطاهای بین دو متد یادگیری $\boldsymbol{L_A}$ *و* $\boldsymbol{L_B}$. بازه های اطمینان این تخمین در متن آورده شده اند.

بازه‌ی اطمینان N% برای تخمین کمیت رابطه‌ی 5.16 با $\bar{\delta}$ به فرم زیر است:

$$\bar{\delta} \pm t_{N,k-1}s_{\bar{\delta}} \tag{5.17}$$

در این رابطه $t_{N,k-1}$ ثابتی است که نقش z_N را در تعریف قبلی بازه‌ی اطمینان بازی می‌کند، $s_{\bar{\delta}}$ نیز تخمین انحراف معیار توزیع حاکم بـر $\bar{\delta}$ است. در کل، $s_{\bar{\delta}}$ به صورت زیر تعریف می‌شود:

$$s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)}\sum_{i=1}^{k}\left(\delta_i - \bar{\delta}\right)^2} \qquad (5.18)$$

توجه دارید که ثابت $t_{N,k-1}$ دو اندیس دارد. اندیس اول درصد اطمینان بازه را مشخص می‌کند، مشابه z_N. پارامتر دوم که درجه‌ی آزادی[1] نیز نامیده می‌شود و معمولاً با υ نمایش داده می‌شود، این پارامتر تعداد فرایند های تصادفی مستقل که برای تولید مقدار تصادفی $\bar{\delta}$ انجام می‌شود را نشان می‌دهد. در شرایط حاضر، درجه‌ی آزادی همان k-1 است. مقادیر ثابت t در جدول 5.6 آورده شده، توجه دارید که با k→∞ مقدار $t_{N,k-1}$ به ثابت z_N میل می کند.

توجه دارید که فرایندی که برای مقایسه‌ی دو متد یادگیری در اینجا آورده شد دقت هر دو فرضیه‌ی یاد گرفته شده را بر اساس یک دسته‌ی تست مشترک بررسی می‌کند. این با چیزی که در قسمت 5.5 در مورد مقایسه‌ی فرضیه‌ها با دسته تست‌های مستقل گفته شد در تضاد است. به تست‌هایی که فرضیه‌ها بر روی مجموعه های مشابهی تست می‌شوند تست‌های جفت[2] می‌گویند. تست‌های جفت معمولاً بازه های اطمینان کوچک‌تری ایجاد می‌کنند زیرا که در خطاهای مشاهده شده در تست‌های جفت فقط بخاطر اختلاف در فرضیه‌هاست. در مقابل، زمانی که فرضیه‌ها بر روی داده های مجزایی تست می‌شوند، تفاوت ناشی از ترکیب دو مجموعه‌ی نمونه ممکن است بر روی تست تأثیر بگذارد.

	درجه‌ی اطمینان N			
	90%	95%	98%	99%
υ=2	2.92	4.30	6.96	9.92
υ=5	2.02	2.57	3.36	4.03
υ=10	1.81	2.23	2.76	3.17
υ=20	1.72	2.09	2.53	2.84
υ=30	1.70	2.04	2.46	2.75
υ=120	1.66	1.98	2.36	2.62
υ=∞	1.64	1.96	2.33	2.58

جدول 5.6 مقادیر $t_{N,\nu}$ برای بازه های اطمینان دو طرفه. با $\nu\to\infty$، $t_{N,\nu}$ به z_N میل می کند.

[1] number of degrees of freedom
[2] paired tests

5.6.1 تست‌های جفتی t[1]

در بالا، فرایندی برای مقایسه‌ی دو متد یادگیری با مجموعه‌ی ثابتی از داده‌ها ارائه شد. در این بخش توجیه آماری برای این فرایند و بازه های اطمینان روابط 5.17 و 5.18 را مورد بحث قرار می‌دهیم. در اولین بار خواندن این کتاب می‌توانید بدون از دست دادن پیوستگی مطالب این قسمت را نخوانید.

بهترین راه برای درک توجیه بازه های تخمینی اطمینان که در رابطه‌ی 5.17 آورده شده در نظر گرفتن مسئله‌ی تخمینی زیر است:

- دسته ای از مقادیر تصادفی مشاهده شده‌ی مستقل و به طور یکسان توزیع شده داریم.
- سعی داریم که میانگین μ را برای توزیع حاکم بر Y_i ها را پیدا کنیم.
- تخمین زننده‌ی مورد استفاده در این مسئله مقدار $\bar{Y}$ است:

$$\bar{Y} = \frac{1}{k}\sum_{i=1}^{k} Y_i$$

این مسئله‌ی تخمین میانگین μ که بر اساس مقدار $\bar{Y}$ صورت می‌گیرد، بسیار کلی است. برای مثال، این حالت کلی مسئله‌ی تخمین $error_{\mathcal{D}}(h)$ با استفاده از $error_S(h)$ را نیز شامل می‌شود. (در این مسئله، Y_i ها 1 یا 0 بودند (بنا به اینکه نمونه درست دسته بندی می‌شود یا خیر) و $error_{\mathcal{D}}(h)$ همان میانگین بود که می‌خواستیم تخمین بزنیم). تست t، که در رابطه‌ی 5.17 و 5.18 نیز آورده شده است، حالت خاصی از این مسئله است، حالتی که Y_i ها از توزیع نرمال پیروی می‌کنند.

حال فرم ایده آل زیر را برای متد جدول 5.5 را برای مقایسه‌ی کارایی دو الگوریتم یادگیری در نظر بگیرید. فرض کنید که به جای داشتن مجموعه‌ی ثابت D_0، می‌توانیم نمونه های آموزشی جدید را بر اساس توزیع احتمال $\mathcal{D}$ دریافت کنیم. در کل، متد ایده آل فرایند جدول 5.5 در هر بار تکرار حلقه از یک دسته‌ی آموزشی جدید S_i و دسته‌ی تست جدید T_i را که با توزیع $\mathcal{D}$ (همان توزیع D_0) ایجاد شده استفاده می‌کند. این متد ایده آل کاملاً با فرم مسئله‌ی تخمینی بالا تطبیق دارد. در کل، معیار δ_i در این فرایند متناسب با متغیرهای تصادفی به طور یکسان توزیع شده Y_i ها هستند. میانگین μ این توزیع‌ها مقدار امید اختلاف بین خطاها بین دو متد یادگیری را نشان می‌دهد (رابطه‌ی 5.14). میانگین نمونه ای $\bar{Y}$ کمیتی است $\bar{\delta}$ که توسط حالت ایده آل این متد اندازه گیری می‌شود. علاقه‌ی ما به جواب این سؤال است که "میانگین $\bar{\delta}$ تا چه میزان تخمین خوبی از μ به ما می‌دهد؟"

ابتدا توجه داشته باشید که اندازه‌ی مجموعه های تست T_i طوری انتخاب می‌شود که هر یک از مجموعه‌ها حداقل 30 نمونه داشته باشد. به همین دلیل، هر کدام از δ_i ها (طبق قضیه‌ی حد مرکزی) توزیعی تقریباً نرمال خواهند داشت. بنابراین، با حالت خاصی روبرو هستیم که در آن توزیع حاکم بر Y_i ها همگی تقریباً نرمال هستند. می‌توان نشان داد که در کل، زمانی که توزیع حاکم بر هر یک Y_i توزیعی نرمال است، توزیع حاکم بر میانگین نمونه ای $\bar{Y}$ توزیعی نرمال خواهد بود. با این دانش که $\bar{Y}$ توزیعی نرمال دارد، می‌توان از آنچه پیش‌تر در مورد بازه های اطمینان گفته شد برای این متغیر تصادفی استفاده کرد (رابطه‌ی 5.11 که برای توزیع احتمال‌های با توزیع نرمال صادق بود). متأسفانه، این رابطه نیاز به انحراف معیار دارد، که در حال حاضر برای ما متغیری مجهول است.

---

[1] Paired t tests

تست t دقیقا برای چنین شرایطی به وجود آمده است، شرایطی که در آن هدف تخمین میانگین نمونه ای مجموعه ای از متغیر های تصادفی با توزیع‌های نرمال به طور یکسان توزیع شده است. در چنین شرایطی، می‌توان از بازه های اطمینان رابطه‌ی 5.17 و 5.18 که با نماد گذاری جدید به شکل زیر نمایش داده می‌شوند استفاده کرد:

$$\mu = \bar{Y} \pm \mathrm{t}_{\mathrm{N,k-1}} \mathrm{s}_{\bar{Y}}$$

که در این رابطه $\mathrm{s}_{\bar{Y}}$ انحراف معیار میانگین نمونه است:

$$\mathrm{s}_{\bar{Y}} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (Y_i - \bar{Y})^2}$$

و $\mathrm{t}_{\mathrm{N,k-1}}$ ثابتی مشابه ثابت قبلی z_N است. در واقع ثابت $\mathrm{t}_{\mathrm{N,k-1}}$ ثابتی است که ویژگی‌های ناحیه ای از توزیع احتمال موسوم به توزیع t را مشابه ثابت z_N که ناحیه ای از توزیع احتمال نرمال را مشخص می‌کرد مشخص می‌کند. توزیع t مشابه توزیع نرمال توزیعی زنگی شکل است با این تفاوت که پهنای بیشتری دارد تا واریانس $\mathrm{s}_{\bar{Y}}$ را داشته باشد تا بتواند انحراف معیار واقعی $\sigma_{\bar{Y}}$ را تخمین بزند. توزیع t با میل کردن متغیر k به سمت بینهایت به توزیع نرمال (و متناسباً $\mathrm{t}_{\mathrm{N,k-1}}$ نیز به z_N) میل خواهد کرد. این منطقی است زیرا که انتظار داریم که $\mathrm{s}_{\bar{Y}}$، با افزایش k به سمت مقدار واقعی انحراف معیار $\sigma_{\bar{Y}}$ میل کند و همچنین زیرا که زمانی که انحراف معیار را دقیقا داریم می‌توانیم از z_N استفاده کنیم.

5.6.2 نکات کاربردی

توجه دارید که بحث بالا استفاده از تخمین بازه‌ی اطمینان رابطه‌ی 5.17 را در حالتی که علاقه ما به میانگین نمونه ای $\bar{Y}$ برای تخمین میانگین مجموعه‌ی k عضوی متغیرهایی مستقل با توزیع نرمال را توجیه می‌کند. این رابطه برای متد ایده آل مطرح شده در بالا ایجاد شده است، در این ایده آل دسترسی بینهایت به نمونه های آموزشی تابع هدف به فرض‌های قبلی اضافه شده. در عمل، با داشتن مجموعه‌ی محدود D_0 از نمونه های آموزشی و متد عملی جدول 5.5، این توجیه کاملاً برقرار نیست. در کل، مسئله اینجاست که تنها راه ایجاد δ_i های جدید باز ترکیب D_0 با تقسیم ان به مجموعه های آموزشی و تست با ترکیب‌های مختلف است. بنابراین، δ_i ها نیز از یکدیگر مستقل نخواهند بود، زیرا که آن‌ها از مجموعه نمونه های آموزشی‌ای که اشتراک دارند و از مجموعه‌ی محدود D_0 انتخاب شده‌اند (به جای اینکه با توزیع احتمال کامل $\mathcal{D}$ انتخاب شوند).

هنگامی که تنها مجموعه‌ی محدود D_0 از نمونه های در دسترس است، چندین متد را می‌توان برای باز ترکیب D_0 به کاربرد. جدول 5.5 متدی به نام k-fold را که در آن مجموعه‌ی D_0 را به k زیر مجموعه‌ی هم اندازه تقسیم می‌کند. در این روش، هر نمونه‌ی D_0 دقیقا در یک مجموعه‌ی تست استفاده و k-1 بار به عنوان نمونه‌ی آموزشی مورد استفاده قرار می‌گیرد. راه حل دیگر متداول انتخاب تصادفی حداقل 30 نمونه از D_0 به عنوان مجموعه‌ی تست و استفاده از بقیه‌ی نمونه‌ها برای آموزش است، این متد را می‌توان به تعداد دلخواه تکرار کرد. این متد تصادفی این مزیت را دارد که می‌توان برای کوچک کردن بازه های اطمینان به اندازه‌ی دلخواه، آن‌را بینهایت بار تکرار کرد. در مقابل، متد k-fold با تعداد داده های موجود با دو شرط اینکه هر نمونه تنها یک بار برای تست به کار برده می‌شود و تعداد داده های دسته‌ی تست حتماً باید بیشتر 30 باشند محدود می‌شود. با این وجود در متد تصادفی دیگر دسته های تست مستقل از همدیگر و بر اساس توزیع احتمال $\mathcal{D}$ نخواهند

بود. در مقابل، دسته های تست ایجاد شده در روش k-fold از یکدیگر مستقل خواهند بود زیرا که هر نمونه تنها در یک دسته‌ی تست حضور دارد.

به طور خلاصه، هیچ فرایندی در مقایسه‌ی متد های یادگیری بر اساس داده های محدود تمامی ویژگی‌هایی که ما می‌خواهیم را ندارد. پس باید در نظر داشت که مدل‌های آماری در تست الگوریتم‌های یادگیری زمانی که تعداد داده های موجود محدود است به ندرت تمامی ویژگی‌های مورد نظر را خواهند داشت. با این وجود، این مدل‌ها بازه های اطمینان را که می‌توانند کمک بزرگی در تفسیر آزمایش‌های مقایسه‌ی متد های یادگیری است را ارائه می‌کنند.

5.7 خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر می‌شود:

- نظریه‌ی آمار مبنایی برای تخمین از خطای واقعی $(error_D(h))$ فرضیه‌ی h بنا بر مشاهداتش از خطای مشاهده شده $(error_S(h))$ بر روی نمونه‌ی S ارائه می‌کند. برای مثال، اگر h یک فرضیه‌ی گسسته مقدار باشد و تعداد داده های نمونه‌ی S بیش از 30 باشد که به طور مستقل از یکدیگر انتخاب شده‌اند، آنگاه بازه‌ی اطمینان N% برای خطای $error_D(h)$ تقریباً بازه‌ی زیر خواهد بود:

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)\big(1 - error_S(h)\big)}{n}}$$

در این رابطه مقدار z_N از جدول 5.1 تعیین می‌شود.

- در کل، مشکل تخمین بازه‌ی اطمینان با تعیین پارامتری که باید تخمین زده شود $(error_D(h))$ و یک تخمین زننده $(error_S(h))$ برای این کمیت انجام می‌گیرد. چون تخمین زننده یک متغیر تصادفی است $(error_S(h))$ وابسته به مجموعه نمونه‌ی تصادفی S است)، آن‌را می‌توان با تابع توزیع احتمال حاکم نشان داد. بازه های اطمینان را می‌توان با پیدا کردن بازه ای از این تابع توزیع که N% حجم احتمال را در بر بگیرد پیدا کرد.
- یکی از دلایل خطا در دقت فرضیه تخمین زننده بایاس تخمین است. اگر Y یک تخمین زننده‌ی پارامتر p باشد، بایاس تخمین Y خطای بین p و مقدار امید Y خواهد بود. برای مثال اگر S داده های آموزشی برای ساخت فرضیه‌ی h باشد، آنگاه $error_S(h)$ نیز تخمین بایاس داری از خطای واقعی $error_D(h)$ خواهد بود.
- دلیل دوم خطا واریانس تخمین است. حتی با تخمین زننده‌ی بدون بایاس نیز مقدار مشاهده شده تخمین زننده در آزمایش‌های متفاوت با هم متفاوت است. واریانس σ^2 ی توزیع حاکم بر خواص تخمین زننده تعیین می‌کند که این مقدار از مقدار واقعی چقدر می‌تواند متفاوت باشد. این واریانس با افزایش تعداد نمونه های داده کاهش می‌یابد.
- مقایسه‌ی کارایی دو الگوریتم یادگیری نیز یک مسئله‌ی تخمین است، که آن زمانی که زمان و داده های آموزشی نامحدودند، بسیار ساده است اما هنگامی که منابع محدود می‌شوند این مسئله کمی سخت‌تر می‌گردد. یکی از راه‌های حل این مسئله که در این فصل

توضیح داده شده اعمال این دو الگوریتم به دو مجموعه‌ی مختلف از داده‌ها و مقایسه‌ی فرضیه های یاد گرفته شده با استفاده از بقیه‌ی داده‌هاست، در انتها نیز می‌توان از میانگین نتایج به عنوان اختلاف دو الگوریتم یاد کرد.

- در بسیاری موارد در نظر گرفته شده در اینجا، اشتقاق بازه‌ی اطمینان با فرض‌ها و تخمین‌هایی انجام گرفته است. برای مثال، بازه‌ی اطمینان مذکور در بالا برای $error_D(h)$ شامل تخمین توزیع دوجمله ای با توزیع نرمال، تخمین واریانس این توزیع و فرض اینکه توزیع احتمال حاکم بر نمونه‌ها ثابت است انجام می‌گیرد. با چنین شرایطی بازه های اطمینان فقط تخمینی از بازه‌ی اطمینان خواهند بود اما با این حال آن‌ها اطلاعات مفیدی برای طراحی و بررسی نتایج یادگیری ماشین به ما می‌دهند.

تعاریف کلیدی آماری این فصل در جدول 5.2 به طور خلاصه آورده شده است.

در بحث یافتن آماری میانگین و بررسی درستی فرضیه‌ها دریایی از اصطلاحات وجود دارد. در حالی که این فصل فقط به مفاهیم اولیه‌ی آماری می‌پردازد، می‌توانید نکات بیشتر آماری را در بسیاری از مقالات و کتب دیگر پیدا کنید. (Billingsley et al. 1986) معرفی بر مباحث آمار مربوطه ارائه می‌کند. دیگر متون آماری شامل (DeGroot 1986) و (Casella and Berger 1990) می‌شوند. (Duda and Hart 1973) نیز بررسی‌ای از این مباحث در قالب پیدا کردن عددی الگوها ارائه می‌کنند.

(Segre et al. 1991 1996)، (Etzioni and Etzioni 1994)، (Gordon and Segre 1996) بررسی‌های مهم آماری برای ارزیابی الگوریتم‌های یادگیری‌ای که کارایی‌شان با کاهش میزان محاسبات سنجیده می‌شوند ارائه می‌کنند.

(German et al. 1992) معیار مینیمم کردن بایاس و واریانس را به طور همزمان بررسی می‌کند. تحقیق بر روی بهترین راه برای یادگیری و مقایسه‌ی فرضیات بر روی تعداد محدود داده همچنان ادامه دارد. برای مثال (Dietterich 1996) مشکلات استفاده از چندین تست t جفت با استفاده از قسمت‌های مختلف داده های موجود به عنوان دسته های آموزشی و تست را بررسی می‌کند.

تمرینات

5.1 فرض کنید که فرضیه ای را بررسی می‌کنید که r = 300 خطا بر روی یک نمونه‌ی S با تعداد n=1000 نمونه‌ی تصادفی دارد. انحراف معیار $error_S(h)$ چقدر است؟ چگونه می‌توان این انحراف معیار را با انحراف معیار انتهای بخش 5.3.4 مقایسه کرد؟

5.2 فرضیه‌ی یاد گرفته شده‌ی h برای مفهومی منطقی را در نظر بگیرید. هنگامی که h بر روی مجموعه ای از 100 نمونه بررسی می‌شود 83 تای آن‌ها را درست دسته بندی می‌کند. انحراف معیار و بازه‌ی 95٪ اطمینان را برای خطای $error_D(h)$ بیابید.

5.3 فرض کنید که فرضیه‌ی h بر روی نمونه ای مستقل با n=65 دارای خطای r=10 است. بازه‌ی دوطرفه‌ی 90٪ اطمینان برای خطای واقعی چقدر است؟ بازه‌ی 95٪ اطمینان یک طرفه چقدر است (یعنی با احتمال 95٪ داریم $error_S(h) \leq U$)؟ بازه‌ی اطمینان 90٪ درصد یک طرفه چقدر است؟

5.4 رابطه ای کلی برای حد بالا و حد پایین بازه‌ی اطمینان یک طرفه‌ی N درصد برای خطاهای مختلف بین دو فرضیه با داده های مختلف ارائه دهید. (راهنمایی: رابطه‌ی بخش 5.5 را تغییر دهید)

5.5 توضیح دهید که چرا تخمین بازه‌ی اطمینان رابطه‌ی 5.17 به تخمین کمیت رابطه‌ی 5.16 اعمال می‌شود و چرا نمی‌توان آن‌را به 5.14 اعمال کرد؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

تست‌های جفت	paired tests
ارزیابی فرضیه‌ها	evaluating hypotheses
بازه‌ی اطمینان	confidence interval
بایاس تخمین زننده	estimator bias
به طور یکسان توزیع شده‌اند	identically distributed
تخمین زننده‌ی بدون بایاس	unbiased estimator
تئوری نمونه برداری	sampling theory
خطای نمونه ای	sample error
خطای واقعی	true error
خوش تعریف	well suited
درجه‌ی آزادی	number of degrees of freedom
قضیه‌ی حد مرکزی	central limit
مرز دو طرفه	two sided bound
مرز یک طرفه	one side bound

فصل ششم: یادگیری بیزی

استدلال بیز روشی احتمالی برای استنتاج ارائه می‌کند. این روش بر اساس این فرض است که کمیت‌های مورد نظر از توزیع‌های احتمال پیروی می‌کنند و تصمیم گیری بهینه را می‌توان با استدلال بر این توزیع‌های احتمال و داده های مشاهده شده انجام داد. اهمیت این روش در یادگیری ماشین این است که روشی کمّی برای ارزیابی مدارک فرضیه‌ها ارائه می‌کند. استدلال بیزی اساس الگوریتم‌های یادگیری‌ای که با استفاده از احتمالات کار می‌کنند است. همچنین استدلال بیز چارچوبی[1] برای بررسی عملیات دیگر الگوریتم‌هایی که از احتمالات استفاده نمی‌کنند ایجاد می‌کند.

٦.١ معرفی

متد های یادگیری بیزی به دو دلیل به مطالعه‌ی ما در مورد یادگیری ماشین مربوط می‌شود. اول اینکه الگوریتم‌های یادگیری بیزی احتمال صریح هر فرضیه، مثل دسته بندی کننده‌ی ساده‌ی بیز[2]، را محاسبه می‌کنند، این نوع روش‌ها از پر کاربردترین روش‌ها در حل بعضی از مسائل یادگیری هستند. برای مثال، (Michie 1994) تحقیقی در مورد تفاوت‌های دسته بندی کننده‌ی ساده‌ی بیز و دیگر الگوریتم‌های یادگیری، از جمله یادگیری درختی و شبکه های عصبی، ارائه می‌کند. این تحقیقات نشان می‌دهد که کارایی دسته بندی کننده‌ی ساده‌ی بیز در بعضی ویژگی‌ها ضعیف‌تر از دیگر الگوریتم‌ها و در بعضی ویژگی‌ها بهتر است. در این فصل دسته بندی کننده‌ی ساده‌ی بیز را به همراه چند مثال بررسی می‌کنیم. در کل، کاربرد این الگوریتم را روی مسئله‌ی دسته بندی متونی مثل مقالات خبری الکترونیک بررسی می‌کنیم. برای چنین کارهای یادگیری‌ای دسته بندی کننده‌ی ساده‌ی بیز یکی از بهترین الگوریتم شناخته شده است.

دلیل دوم اهمیت متد های بیز در مطالعه‌ی ما در یادگیری ماشین زمینه‌ی مساعدی است که این متدها برای درک الگوریتم‌های یادگیری‌ای که مستقیماً با احتمالات کار نمی‌کنند ایجاد می‌کند. برای مثال، در این فصل، ما الگوریتم‌هایی چون Find-S و Candidate-Elimination،

[1] framework
[2] naive Bayes classifier

را که در فصل ۲ آمده بود، برای مشخص کردن شروطی که خروجی، محتمل‌ترین فرضیه‌ی سازگار با نمونه های آموزشی باشد را بررسی خواهیم کرد. همچنین با بررسی‌ای بیزی توجیهی برای یکی از انتخاب‌های کلیدی الگوریتم‌های یادگیری شبکه های عصبی (انتخاب تابع خطای مجموع مربعات خطا برای جستجوی فضای شبکه های عصبی ممکن) ارائه خواهیم کرد. همچنین در این بخش اشتقاق تابع خطای جایگزینی را محاسبه می‌کنیم، cross-entropy، این معیار زمانی که تابع هدف احتمالات را پیش بینی می‌کند از معیار مجموع خطاهای مربعی کارامد تر است. از نظری بیزی بایاس استقرایی الگوریتم‌های درختی که به درخت‌های کوچک‌تر علاقه دارند و قانون کوتاه‌ترین طول توضیح را بررسی خواهیم کرد. آشنایی پایه ای با روش‌های بیزی برای درک بسیاری از الگوریتم‌های یادگیری ماشین اهمیت بسزایی دارد. ویژگی‌های متد های یادگیری بیزی شامل موارد زیر است:

- هر نمونه‌ی آموزشی می‌تواند احتمال تخمینی اینکه فرضیه درست است را کم یا زیاد کند. این حقیقت باعث می‌شود که روش‌های بیزی نسبت به الگوریتم‌هایی که کاملاً فرضیه را با نمونه های غیر سازگار رد می‌کنند انعطاف‌تر پذیر تر باشند.
- می‌توان از دانش قبلی به همراه داده های مشاهده شده برای تعیین احتمال نهایی درستی فرضیه‌ها استفاده کرد. در یادگیری بیزی، دانش قبلی با (۱) احتمال اولیه‌ی هر فرضیه و (۲) توزیع احتمال روی داده های تعیین شده برای هر فرضیه‌ی ممکن تعیین می‌شود.
- متد های بیزی می‌توانند برای فرضیه‌ها احتمالاتی را پیش بینی کنند (برای مثال، فرضیه‌ی "این بیمار ذات‌الریه با احتمال ۹۳٪ کاملاً بهبود خواهد یافت).
- نمونه های جدید را می‌توان با ترکیب پیش بینی‌های چندین فرضیه، (هر کدام با وزن احتمالشان) دسته بندی کرد.
- حتی هنگامی که اثبات می‌شود که متد های بیزی محاسباتی غیر قابل پیش بینی انجام می‌دهند، با این حال معیار استانداردی برای دیگر متد های عملی یادگیری مطرح می‌کنند.

یکی از مشکلات عملی کاربرد متد های بیزی نیاز آن‌ها به داشتن دانش اولیه از بسیاری از احتمالات است. هنگامی که این اطلاعات به طور دقیق در دسترس نیست، گاهی آن‌ها را با استفاده از دانش قبلی، داده های موجود قبلی، و فرض‌هایی درباره‌ی فرم توزیع تخمین می‌زنیم. دومین مشکل عملی کاربرد این متدها هزینه‌ی محاسباتی قابل توجه آن‌ها برای تعیین فرضیه‌ی بهینه‌ی بیز در حالت کلی است (که رابطه خطی با تعداد فرضیه های ممکن دارد). در حالت‌های خاص خاصی این هزینه‌ی محاسباتی به طور قابل توجهی کاهش می‌یابد.

ادامه‌ی این فصل به شکل زیر ساختار بندی شده است. بخش ۶.۲ قضیه‌ی بیز را معرفی کرده و محتمل‌ترین[1] و فرضیه ای با حداکثر احتمال ثانویه[2] را تعریف خواهد کرد. چهار زیر بخش این بخش این چارچوب[3] را برای بررسی چندین مشکل و الگوریتم یادگیری که در فصل‌های گذشته مطرح شد به کار می‌برند. برای مثال، نشان می‌دهیم که چندین الگوریتم مطرح شده با چه فرض‌هایی محتمل‌ترین فرضیه را خروجی می‌دهند. بخش‌های بعدی تعدادی از الگوریتم‌های یادگیری که منحصر با احتمالات کار می‌کنند را معرفی خواهند کرد. این الگوریتم‌ها شامل دسته بندی کننده‌ی بهینه‌ی بیز، الگوریتم گیبز و دسته بندی کننده‌ی ساده‌ی بیز می‌شود. بالاخره درباره‌ی شبکه‌ی باور بیز[4] بحث خواهیم کرد و روشی جدید برای یادگیری بر اساس استدلال احتمالی و الگوریتم EM که الگوریتمی پرکاربرد در یادگیری در حضور متغیر های غیر قابل مشاهده است را بررسی خواهیم کرد.

---

[1] maximum likelihood
[2] maximum a posteriori probability hypotheses
[3] framework
[4] Bayesian belief network

۶.۲ قضیه‌ی بیز

در یادگیری ماشین، گاهی سعی داریم که از میان فضای فرضیه های H بهترین فرضیه‌ی سازگار با نمونه های آموزشی D را پیدا کنیم. چندین راه برای تعریف "بهترین" در این جمله وجود دارد، یکی از این تعاریف "محتمل‌ترین" است، با در دست داشتن داده های D بدون نیاز به هیچ اطلاعات اولیه‌ی دیگر نمی‌توان محتمل‌ترین فرضیه را انتخاب کرد. قضیه‌ی بیز متدی مستقیم برای محاسبه‌ی احتمالات فرضیه های موجود در H ارائه می‌کند. به عبارت دیگر، قضیه‌ی بیز روشی برای محاسبه‌ی احتمال یک فرضیه بر اساس احتمال قبلی‌اش، احتمال مشاهده‌ی داده های سازگار با فرض درستی این فرضیه و احتمال خود داده های مشاهده شده ارائه می‌کند.

برای تعریف دقیق قضیه‌ی بیز، ابتدا بیایید نشانه گذاری‌ها را معرفی کنیم. برای نشان دادن احتمال اولیه‌ی فرضیه‌ی h، احتمال قبل از مشاهده‌ی داده های آموزشی، از P(h) استفاده می‌کنیم. به P(h) احتمال اولیه‌ی[1] h نیز می‌گویند، این احتمال از اطلاعات قبلی‌ای که در مورد احتمال درستی فرضیه‌ی h داریم تأثیر می‌پذیرد. به طور مشابه از P(D) برای نمایش احتمال اولیه‌ی مشاهده‌ی نمونه های آموزشی D استفاده می‌کنیم (مثلاً احتمال مشاهده‌ی D بدون داشتن هیچ اطلاعات قبلی در مورد اینکه با چه فرضیه‌هایی سازگار است). برای نشان دادن احتمال مشاهده‌ی D در جایی که فرضیه‌ی h درست است از P(D|h) استفاده می‌کنیم. در حالت کلی، از p(x|y) برای نشان دادن احتمال x با فرض وقوع y استفاده می‌کنیم. در مسائل یادگیری ماشین، علاقه‌ی ما به احتمال P(h|D) است که در آن h یک فرضیه و D نمونه های آموزشی مشاهده شده هستند. به P(h|D) احتمال ثانویه‌ی[2] h نیز می‌گویند، زیرا که اطمینان ما به فرضیه‌ی h بعد از مشاهده‌ی نمونه های آموزشی D را نشان می‌دهد. توجه داشته باشید که احتمال ثانویه P(h|D) بر خلاف احتمال اولیه P(h) که از نمونه های آموزشی مستقل است، از نمونه های آموزشی D تأثیر می‌پذیرد.

قضیه بیز، اساس متد های یادگیری بیز است زیرا که راهی برای محاسبه‌ی احتمال ثانویه P(h|D) از P(h)، P(D) و P(D|h).

قضیه‌ی بیز:

$$p(h|D) = \frac{p(D|h)P(h)}{p(D)} \tag{6.1}$$

همان طور که انتظار می‌رود، بر اساس قضیه‌ی بیز P(h|D) با افزایش P(h) و P(D|h) افزایش می‌یابد. همچنین منطقی است که P(h|D)، با افزایش P(D) کاهش بیابد، زیرا که هر چه که احتمال مشاهده‌ی D به طور مستقل از h بالا تر رود دیگر D مدرکی برای درستی h نخواهد بود.

در بسیاری از مسائل یادگیری، یادگیر مجموعه‌ی فرضیه‌هایی مثل H را در نظر می‌گیرد و در بین آن‌ها به دنبال محتمل‌ترین فرضیه‌ی $h \in H$ با توجه به نمونه های آموزشی D می‌گردد (یا حداقل یکی از محتمل‌ترین فرضیه‌ها). هر کدام از این محتمل‌ترین، فرضیه با حداکثر

[1] prior probability
[2] posterior probability

احتمال ثانویه[1] یا (MAP) نامیده می‌شود. فرضیه های MAP را می‌توان با استفاده از قضیه‌ی بیز برای محاسبه‌ی احتمال ثانویه‌ی هر فرضیه مشخص کرد. به صورت دقیق‌تر، زمانی می‌گوییم که فرضیه‌ی h_{MAP} یک فرضیه‌ی MAP است که

$$h_{MAP} = \arg\max_{h\in H} P(h|D)$$

$$= \arg\max_{h\in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h\in H} P(h|D)\, P(h) \qquad (6.2)$$

توجه داشته باشید که مرحله آخر عبارات بالا P(D) چون ثابتی است و h بر آن تأثیری ندارد حذف می‌شود.

در بعضی موارد، فرض می‌کنیم که هر فرضیه در H احتمال اولیه‌ی مساوی‌ای دارد (برای هر h_j و h_i در H داریم که $P(h_i) = P(h_j)$. در این شرایط می‌توان رابطه‌ی ۶.۲ را بیشتر ساده کرد و کافی است که فقط عبارت P(D|h) را برای پیدا کردن محتمل‌ترین فرضیه در نظر بگیریم. P(D|h) گاهی محتمل بودن داده های D برای h نیز نامیده می‌شود و هر فرضیه ای که P(D|h) را ماکزیمم کند (ML)[2] ، h_{ML} نامیده می‌شود.

$$h_{ML} \equiv \arg\max_{h\in H} P(D|h) \qquad (6.3)$$

برای مشخص شدن رابطه با مسائل یادگیری ماشین، ابتدا قضیه‌ی بیز را با توجه به نمونه های D و فضای فرضیه ای H معرفی کردیم. در واقع قضیه‌ی بیز کلی‌تر از آنچه در بالا گفته شد است. از قضیه‌ی بیز می‌توان برای هر زیر مجموعه‌ی H که ناسازگارند (اشتراک ندارند) استفاده کرد (مثل "آسمان آبی است" و "آسمان آبی نیست"). در این فصل، در اکثر موارد فرض خواهیم کرد که H فضای فرضیه ای که تابع هدف را شامل می‌شود است و D نمونه های آموزشی هستند. در مواقع دیگر فرض می‌کنیم که H مجموعه‌ی دیگر ناسازگاری با یکدیگر از فرضیه‌هاست و D نیز مجموعه‌ی دیگری از داده‌هاست.

۶.۲.۱ یک مثال

برای تصور قانون بیز، فرض کنید که مسئله ای برای تشخیص بیماری داریم، دو فرضیه‌ی ممکن برای بیماری وجود دارد: (۱) بیمار نوع خاصی از سرطان دارد و (۲) بیمار آن نوع سرطان را ندارد. داده های موجود یک تست آزمایشگاهی است که دو خروجی ممکن دارد: $\oplus$ (مثبت) و $\ominus$ (منفی). دانش قبلی داریم در کل جمعیت حاضر در آزمایش فقط 008. این بیماری را دارند. علاوه بر آن نتیجه‌ی آزمایش همیشه قطعی نیست و احتمال خطا وجود دارد. تست آزمایشگاهی در ۹۸٪ مواردی که بیمار بیماری را دارد نتیجه‌ی مثبت درست می‌دهد و در ۹۷٪ مواردی که بیمار بیماری را ندارد نتیجه‌ی منفی درست می‌دهد. در بقیه‌ی موارد آزمایش نتیجه‌ی اشتباه می‌دهد. آنچه در بالا گفته شد را خلاصه‌وار می‌توان به صورت زیر نشان داد:

$$P(cancer) = .008, \qquad P(\neg cancer) = .992$$

[1] Maximum A Posteriori
[2] maximum likelihood

$$P(\oplus\,|cancer) = .98, \quad P(\ominus\,|\neg cancer) = .02$$

$$P(\oplus\,|\neg cancer) = .03, \quad P(\ominus\,|\neg cancer) = .97$$

فرض کنید که بیمار جدیدی پذیرش می‌شود و نتیجه‌ی آزمایشش مثبت است. حال با چه احتمالی می‌توان گفت که بیمار سرطان دارد؟ فرضیه با حداکثر احتمال را می‌توان از رابطه‌ی ۶.۲ پیدا کرد:

$$P(\oplus\,|cancer)P(cancer) = (.98).008 = .0078$$

$$P(\oplus\,|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$

پس، داریم $h_{MAP} = \neg cancer$. احتمال ثانویه‌ی فرضیه‌ها را می‌توان با رساندن مجموع دو احتمال به ۱ پیدا کرد ($P(\oplus\,|cancer)P(cancer) =.\frac{0078}{.0078+.0298} = .21$). این مرحله برای این درست است که قضیه‌ی بیز احتمال‌های ثانویه مجموعه‌ی تمامی داده را بدون اشتراک می‌پوشانند (افراز می‌کنند) بیان می‌کند. با وجود اینکه $P(\oplus)$ مستقیماً توسط مسئله داده نشده است، اما می‌توان آن‌را محاسبه کرد زیرا که می‌دانیم مجموع دو احتمال $P(cancer|\oplus)$ و $P(\neg cancer|\oplus)$ یک است (هر بیمار یا سرطان دارد یا سرطان ندارد). توجه داشته باشید که احتمال ثانویه‌ی سرطان نسبت به احتمال اولیه‌ی آن به طور قابل توجهی زیادتر است، اما با این حال محتمل‌ترین فرضیه این است که بیمار سرطان ندارد.

همان طور که در مثال بالا نشان داده شد، نتیجه‌ی تأثیر بیز به شدت به احتمال اولیه وابسته است، برای اینکه بتوان قضیه را به طور مستقیم به کار برد باید احتمالات اولیه معلوم باشند. توجه داشته باشید که در این مثال فرضیه‌ها کاملاً پذیرفته شده یا رد شده نیستند بلکه هر کدام با افزایش داده های مشاهده شده احتمالی پیدا می‌کنند. فرمول اصلی محاسبه‌ی احتمالات در جدول ۶.۱ خلاصه شده است.

۶.۳ قضیه‌ی بیز و یادگیری مفهوم

ارتباط بین قضیه‌ی بیز و مسائل یادگیری مفهوم چیست؟ از آنجایی که قضیه‌ی بیز راهی اصولی برای محاسبه‌ی احتمالات ثانویه‌ی هر یک از فرضیه‌ها بعد از مشاهده‌ی داده های آموزشی ارائه می‌کند، می‌توانیم از آن برای پایه‌ی یک الگوریتم یادگیری ساده استفاده کنیم، الگوریتمی که احتمال هر یک از فرضیه‌ها را محاسبه کرده و محتمل‌ترین فرضیه‌ها را خروجی می‌دهد. در این بخش چنین الگوریتم‌های بدون شعور[1] یادگیری مفهوم بیز را بررسی و با الگوریتم‌های یادگیری مفهوم مقایسه می‌کنیم. همان طور که بعداً نیز خواهیم دید، یکی از نتایج جالب این مقایسه این است که تحت شرایط خاصی چندین الگوریتمی که در فصل‌های گذشته بررسی شدند همان فرضیه ای که یادگیری بدون شعور بیز خروجی می‌دهد را خروجی می‌دهند، با این تفاوت که آن‌ها احتمالات فرضیه‌ها را مشخص نمی‌کنند و فقط محتمل‌ترین را مشخص می‌کنند.

- قانون ضرب[2]: احتمال $P(A\wedge B)$ که احتمال عطف دو اتفاق A و B است را محاسبه کن

$P(A\wedge B)=P(A|B)P(B)=P(B|A)P(A)$

- قانون جمع[1]: احتمال فصل دو اتفاق A و B را محاسبه کن

---

[1] brute-force
[2] Product rule

P(A∨B)=P(A)+P(B)-P(A∧B)

- قضیه بیز[2]: احتمال ثانویه‌ی P(h|D) را محاسبه کن

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- قضیه‌ی مجموع احتمالات[3]: اگر اتفاق‌های $A_1, \dots, A_n$ دو به دو ناسازگار باشند و $\sum_{i=1}^{n} P(A_i) = 1$ خواهیم داشت

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

جدول ۶.۱ خلاصه‌ی فرمول‌های پایه ای احتمال.

۶.۳.۱ یادگیری مفهوم بدون شعور بیز

مسئله‌ی یادگیری مفهومی را که در ابتدای فصل ۲ معرفی شد را در نظر بگیرید. در کل، فرض کنید که یادگیر فضای فرضیه ای محدود H را که شامل فرضیه‌هایی که بر فضای نمونه ای X تعریف شده‌اند است در نظر می‌گیرد و تابع هدف نیز مفهومی به فرم c:X→{0,1} است. مثل معمول، فرض می‌کنیم که به یادگیر دسته ای از نمونه های آموزشی مثل $\{< x_1, d_1 > \cdots < x_m, d_m >\}$ داده می‌شود، در این نمونه های آموزشی x_i عضوی از X و d_i نیز مقدار هدف برای آن x_i است $(d_i = c(x_i))$. برای ساده سازی بحث در این بخش، فرض می‌کنیم که ترتیب نمونه های نمونه‌ها $< x_1 \dots x_m >$ ثابت است پس می‌توان نمونه های آموزشی D را به فرم ساده به صورت $D =< d_1 \dots d_m >$ نوشت. می‌توان نشان داد که این ساده نویسی تأثیری بر نتایج بدست آمده از این قسمت ندارد (تمرین ۶.۴).

می‌توان با استفاده از قضیه‌ی بیز الگوریتم یادگیری مفهوم مستقیمی طراحی کرد که فرضیه با حداکثر احتمال ثانویه را خروجی دهد:

الگوریتم یادگیری بدون شعور MAP [4]

۱. برای هر فرضیه h در H احتمال ثانویه را محاسبه کن،

$$\mathrm{p(h|D)} = \frac{\mathrm{p(D|h)P(h)}}{\mathrm{p(D)}}$$

۲. فرضیه h_{MAP} را که بیشترین احتمال ثانویه را دارد خروجی بده

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

[1] Sum rule
[2] Bayes theorem
[3] Theorem of total probabily
[4] Brute-Force MAP Learning

این الگوریتم ممکن است محاسبات قابل توجهی نیاز داشته باشد، زیرا که قانون بیز را برای تمامی فرضیه های H برای محاسبه‌ی P(h|D) به کار می‌برد. چنین حجم محاسباتی‌ای برای فضای فرضیه‌هایی با اندازه‌ی بالا غیر عملی است، با این وجود الگوریتم هنوز مورد توجه است زیرا که معیاری ارائه می‌کند در حالی که دیگر الگوریتم‌های یادگیری مفهوم هیچ معیاری ارائه نمی‌کنند.

برای آماده سازی یک مسئله برای حل با الگوریتم یادگیری بدون شعور MAP لازم است که مقادیر P(h) و P(D|h) را مشخص کنیم (همان طور که بعداً هم خواهیم دید با مشخص کردن مقادیر ذکر شده مقدار P(D) نیز مشخص می‌شود). اطلاعات اولیه‌ی ما در مورد مسئله با تعیین دو توزیع P(h) و P(D|h) به طور دلخواه مشخص می‌شود. بیایید ابتدا با فرض‌های زیر شروع کنیم :

۱. نمونه های آموزشی D خطا ندارند $(d_i = c(x_i))$
۲. مفهوم هدف c در فضای فرضیه ای H موجود است.
۳. هیچ مدرکی بر برتری یک فرضیه بر فرضیه‌ی دیگر وجود ندارد.

با فرض‌های بالا، چه مقداری باید برای P(h) تعیین شود؟ بدون هیچ اطلاعات قبلی، برتری فرضیه‌ها بر یکدیگر بی دلیل خواهد بود، می‌توانیم احتمال تمامی آن‌ها را مساوی قرار دهیم. علاوه بر ان، چون فرض کرده‌ایم تابع هدف c در H موجود است باید طوری احتمال را پخش کنیم که مجموع احتمال کل H یک باشد. پس خواهیم داشت:

$$P(h) = \frac{1}{|H|} \quad for\ all\ h\ in\ H$$

اما P(D|h) چه احتمالی باید داشته باشد؟ P(D|h) احتمال مشاهده‌ی مقادیر هدف $D =< d_1 \dots d_n >$ است که برای دسته‌ی ثابت نمونه‌ها زمانی که h درست است می‌باشد.(مثلاً زمانی که h همان مفهوم هدف c است). از آنجایی که فرض کردیم داده های آموزشی خطا ندارند، احتمال دیدن d_i اگر $d_i = h(x_i)$ باشد ۱ و اگر $d_i \neq h(x_i)$ باشد۰ است. بنابراین،

$$P(D|h) = \begin{cases} 1 \ \ if\ d_i = h(x_i) \ \ for\ all\ d_i\ in\ D \\ 0 \ \ otherwise \end{cases} \tag{6.4}$$

به عبارت دیگر احتمال مشاهده‌ی D با داشتن h، ۱ است اگر D با h سازگار باشد و در غیر این صورت ۰ است.

با این نوع انتخاب P(h) و P(D|h) حال مسئله را کاملاً برای الگوریتم یادگیری بدون شعور MAP آماده کرده‌ایم. مرحله‌ی اول این الگوریتم که در آن با استفاده از قضیه‌ی بیز احتمال ثانویه‌ی P(h|D) برای تمامی h ها با توجه به نمونه های آموزشی D محاسبه می‌شود را در نظر بگیرید. با توجه به قضیه‌ی بیز داریم،

$$\mathrm{p(h|D)} = \frac{\mathrm{p(D|h)P(h)}}{\mathrm{p(D)}}$$

ابتدا فرض کنید که h با نمونه های آموزشی ناسازگار است. از رابطه‌ی ۶.۴ داریم که P(D|h) صفر است زیرا که h با D ناسازگار است پس داریم که:

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0\ if\ h\ is\ inconsistent\ with\ D$$

پس احتمال ثانویه‌ی فرضیه‌ی ناسازگار با D صفر خواهد بود.

حال فرض کنید که فرضیه‌ی h با D سازگار است. از رابطه‌ی ۶.۴ داریم که P(D|h) یک فرض شده است زیرا که h با D سازگار است. داریم،

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)}$$

$$= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}}$$

$$= \frac{1}{|VS_{H,D}|} \; if\ h\ is\ consistent\ with\ D$$

در این رابطه $VS_{H,D}$ زیر مجموعه ای از H است که با D سازگار است (مثلاً $VS_{H,D}$ می‌تواند همان فضای ویژه ای فصل ۲ باشد که با توجه به D بدست آمده). تشخیص اینکه $P(D) = \frac{VS_{H,D}}{|H|}$ کار ساده ای است زیرا که مجموع P(h|D) برای تمامی فرضیه‌ها باید ۱ باشد و از طرفی تعداد کل فرضیه های سازگار با D در H طبق تعریف $|VS_{H,D}|$ است. می‌توان مقدار P(D) را از قضیه‌ی مجموع احتمال (در جدول ۶.۱) و این حقیقت که فرضیه‌ها دو به دو ناسازگارند $((\forall i \neq j)(P(h_i \wedge h_j) = 0))$ بدست آورد.

$$P(D) = \sum_{h_i \in H} P(D|h_i)P(h_i)$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|}$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|}$$

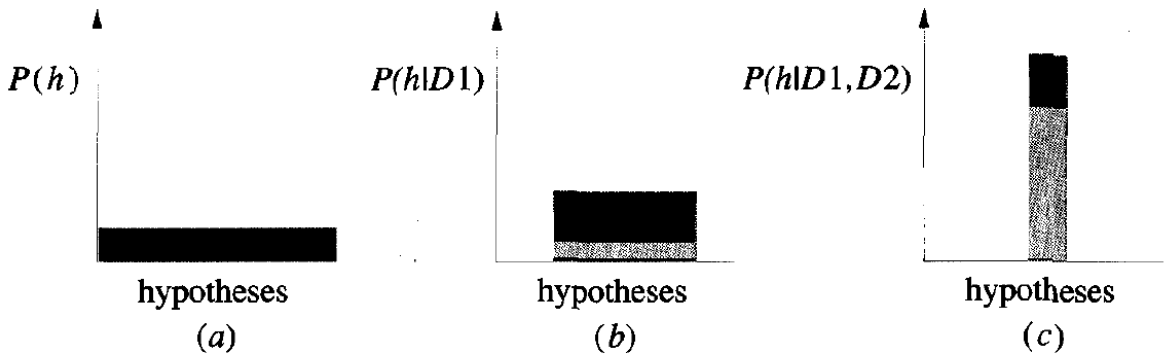
$$= \frac{|VS_{H,D}|}{|H|}$$

به طور خلاصه اینکه با فرض‌هایی که در مورد P(h) و P(D|h) کردیم قضیه‌ی بیز ایجاب می‌کند که P(h|D) به صورت زیر باشد:

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & if\ h\ is\ consistent\ with\ D \\ 0 & otherwise \end{cases} \qquad (6.5)$$

در این رابطه $|VS_{H,D}|$ تعداد فرضیه های H که با D سازگارند است. شکل ۶.۱ سیر تکامل احتمالات را با نمودار نشان می‌دهد. ابتدا (شکل 6.1 (a)) تمامی فرضیه‌ها احتمال یکسانی دارند. با افزایش داده های آموزشی (شکل‌های 6.1 (b) و 6.1 (c)) احتمال ثانویه‌ی فرضیه های ناسازگار صفر می‌شود اما مجموع کل احتمالات ۱ باقی می‌ماند، یعنی احتمال فرضیه‌هایی که صفر می‌شود به طور مساوی در بین فرضیه های دیگر تقسیم می‌شود.

بررسی بالا نشان داد با انتخاب P(h) و P(D|h) تمامی فرضیه های سازگار احتمال ثانویه‌ی مساوی $(1/|VS_{H,D}|)$ خواهند داشت و احتمال فرضیه های ناسازگار صفر خواهد شد. پس با توجه به این بررسی هر فرضیه‌ی سازگار یک MAP (فرضیه با حداکثر احتمال) است.



شکل ۶.۱ تکامل احتمال ثانویه‌ی P(h|D) با افزایش داده های آموزشی.
(a) اولویت یکسان به تمامی فرضیه‌ها داده می‌شود. با افزایش داده‌ها به D1 (b) و سپس به D1∧D2 (c)، احتمال ثانویه‌ی فرضیه های ناسازگار به صفر می‌رسد در حالی که احتمال ثانویه برای فرضیه های فضای ویژه افزایش می‌یابد.

۶.۳.۲ فرضیه های MAP و یادگیرهای سازگار

بررسی‌های بالا نشان می‌دهد که با مفروضات مذکور تمامی فرضیه های سازگار با D فرضیه ای MAP هستند. این عبارت را می‌توان مستقیماً به عبارتی جالب در مورد دسته ای از یادگیرها که یادگیر های سازگار[1] می‌نامیم تفسیر کرد. زمانی می‌گوییم که یک الگوریتم یادگیری یادگیر سازگار است که فرضیه‌ی خروجی هیچ خطایی بر روی داده های آموزشی نداشته باشد. بر اساس بررسی بالا، می‌توان گفت تمامی یادگیر های سازگار فرضیه‌ی خروجیشان یک فرضیه‌ی MAP است، به شرطی که فرض کنیم که توزیع اولیه احتمال روی H یکنواخت باشد $((\forall i,j)P(h_i) = P(h_j))$ و همچنین فرض کنیم که داده های آموزشی قطعی و بدون خطا هستند (P(D|H)=1 اگر D با h سازگار باشد و در غیر این صورت صفر است).

برای مثال، الگوریتم یادگیری مفهوم Find-S را که در فصل ۲ بررسی شد را در نظر بگیرید. Find-S فضای فرضیه ای H را از فرضیه های جزئی‌تر به کلی‌تر جستجو می‌کند تا جزئی‌ترین فرضیه‌ی سازگار را پیدا کند (جزئی‌ترین عضو فضای ویژه). چون Find-S فرضیه ای سازگار را خروجی می‌دهد پس طبق احتمالات مفروض بالا برای P(h) و P(D|h) فرضیه ای MAP را خروجی خواهد داد. البته Find-S هیچ

[1] consistent learner

احتمالی را محاسبه و ارائه نمی‌کند و فقط خاص‌ترین فرضیه‌ی فضای ویژه را پیدا می‌کند. با این وجود، با مشخص کردن توزیع‌های P(h) و P(D|h) به صورتی که فرضیه‌ی خروجی MAP باشد، روشی مفید برای مشخص کردن رفتار Find-S داریم.

آیا توزیع احتمال‌های دیگری برای P(h) و P(D|h) وجود دارد که خروجی FIND-S فرضیه‌ی MAP باشد؟ بله، چون FIND-S خاص‌ترین فرضیه‌ی فضای ویژه را پیدا می‌کند فرضیه‌ی خروجی‌اش با اختصاص توزیع احتمال‌هایی که به سمت فرضیه های خاص‌تر تمایل دارند MAP خواهد بود. به عبارت دقیق‌تر، فرض کنید که $\mathcal{H}$ تمامی توزیع احتمال P(h) روی H است که در آن‌ها داریم $P(h_1) \geq P(h_2)$ اگر h_1 خاص‌تر از h_2 باشد. می‌توان نشان داد با چنین توزیع احتمال‌هایی و توزیع احتمال مذکور برای P(D|h) فرضیه‌ی خروجی FINS-S یک فرضیه‌ی MAP خواهد بود.

خلاصه بحث بالا بدین شکل است، چارچوب بیزی به ما اجازه می‌دهد تا ویژگی‌های رفتاری الگوریتم‌های یادگیری (حتی الگوریتم‌هایی که مقدار احتمال فرضیه را مشخص نمی‌کنند، مثل FIND-S) را مشخص کنیم. با مشخص کردن توزیع احتمال‌های P(h) و P(D|h) به صورتی که فرضیه‌ی خروجی الگوریتم بهینه، MAP، شود، پیش فرض‌هایی که الگوریتم برای نتیجه گیری انجام می‌دهد را می‌توان پیدا کرد.

استفاده از دیدگاه بیزی برای بررسی ویژگی‌های الگوریتم‌های یادگیری بدین صورت عملاً مشابه بررسی بایاس استقرایی یادگیرهاست. در فصل ۲ ما بایاس استقرایی یک الگوریتم را دسته پیش فرض‌هایی مثل B تعریف کردیم که نحوه‌ی استقرای یادگیر را توجیه می‌کند. برای مثال، گفته شد که بایاس استقرایی الگوریتم Candidate-Elimination وجود مفهوم هدف c در مجموعه‌ی فرضیه ای H است. علاوه بر آن نشان دادیم که خروجی این الگوریتم یادگیری را می‌توان از ورودی‌هایش و این پیش فرض استقرایی ضمنی نتیجه گرفت. تفسیری بیزی بالا می‌تواند جایگزینی برای بررسی ویژگی‌های این پیش فرض‌های الگوریتم‌های یادگیری باشد. با این تفاوت که در اینجا به جای مدل کردن الگوریتم با یک سیستم معادل استقرایی، الگوریتم را با سیستم معادل استدلال احتمالی[1] که بر اساس قضیه‌ی بیز کار می‌کند، مدل سازی می‌کنیم. و در اینجا پیش فرض‌هایی که یادگیر فرض می‌کند به فرم "احتمال اولیه های فرضیه‌ها P(h) و قدرت داده‌ها در قبول یا رد فرضیه‌ها P(D|h)" است. تعریف P(h) و P(D|h) که در این قسمت معرفی شد مربوط به دو الگوریتم Candidate-Elimination و FIND-S بود. سیستم استدلال احتمالی‌ای که بر اساس قضیه‌ی بیز کار می‌کند، با این توزیع‌ها در ورودی و خروجی رفتاری مشابه این الگوریتم‌ها از خود نشان خواهد داد.

بحثی که در این بخش انجام شد حالت خاصی از استدلال بیزی بود زیرا که فرض کردیم داده های آموزشی بدون خطایند و فرضیه‌ها نیز قطعی‌اند، یعنی P(D|h) حتماً یکی از دو مقدار ۱ یا ۰ را دارد. همان طور که در قسمت بعدی نیز خواهیم دید، می‌توان یادگیری از نمونه های آموزشی خطا دار را شبیه سازی کرد، فقط کافی است که مقدار P(D|h) مقادیری غیر ۰ و ۱ را نیز داشته باشد، با این تغییر توزیع احتمال P(D|h) خطا را کنترل خواهد کرد.

[1] probabilistic reasoning system

٦.٤ محتمل‌ترین فرضیه‌ها[1] و فرضیه‌هایی که کمترین خطای مربعی[2] را دارند

همان طور که در بالا نیز نشان داده شد تحت شرایطی یک الگوریتم یادگیری فرضیه های MAP را خروجی می‌دهد حتی اگر این الگوریتم از روش بیز یا حتی از محاسبه‌ی احتمال‌ها استفاده نکند.

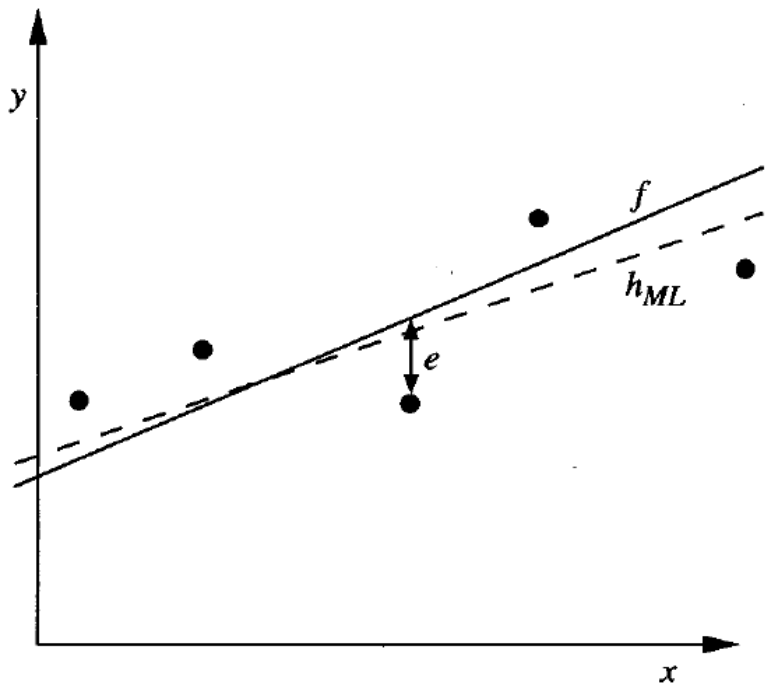
در این بخش، به مسئله ای یادگیری توابع هدف پیوسته مقدار می‌پردازیم، مسئله ای که راه‌های زیادی برای آن مثل شبکه های عصبی، تقریب خطی، و تقریب چند جمله ای ارائه شده است. یک بررسی مستقیم بیزی نشان می‌دهد که در شرایط خاصی هر الگوریتم یادگیری که خطای مربعی بین تخمین و خروجی داده های آموزشی را مینیمم کند یک محتمل‌ترین فرضیه[3] را خروجی می‌دهد. اهمیت این نتیجه در استفاده از این استدلال بیزی (تحت شرایط خاص) برای توجیه بسیاری از شبکه های عصبی و دیگر متدهایی که مجموع خطای مربعی را مینیمم می‌کنند است.

شرایط مسئله‌ی یادگیری تابع هدف پیوسته را در نظر بگیرید، یادگیر L که از فضای نمونه ای X و فضای فرضیه ای H که مجموعه ای از توابع حقیقی مقدار روی نمونه های X استفاده می‌کند (هر h در H تابعی است به فرم $h: X \to \Re$، که در آن $\Re$ مجموعه‌ی اعداد حقیقی است). مسئله ای که یادگیر L با آن مواجه است یادگیری تابع هدف مجهول $f: X \to \Re$ از مجموعه‌ی فرضیه های H است. مجموعه ای از m نمونه‌ی آموزشی در دسترس است، در این مجموعه مقدار تابع هدف هر یک از نمونه‌ها با یک مقدار تصادفی خطا که توزیع نرمال دارد معلوم است. به عبارت دقیق‌تر، هر نمونه‌ی آموزشی زوج مرتبی به فرم $< x_i, d_i >$ است که در آن $d_i = f(x_i) + e_i$. در اینجا $f(x_i)$ خود تابع هدف و مقدار e_i متغیر تصادفی خطاست. فرض می‌شود که مقدار e_i مستقل و دارای توزیع نرمال با میانگین صفر است. هدف یادگیر نیز پیدا کردن محتمل‌ترین فرضیه، یا به صورت معادل، یک فرضیه‌ی MAP است با این فرض که تمامی فرضیه‌ها احتمال اولیه‌ی یکسانی دارند.

با وجود اینکه بررسی‌هایمان را برای یادگیری توابع دلخواه حقیقی مقدار انجام می‌دهیم، مسئله‌ی یادگیری تابع خطی نمونه ای از چنین مسائلی است. شکل ۶.۲ شکل تابع هدف خطی f را به همراه چندین نمونه‌ی آموزشی نشان داده است. خط چین فرضیه‌ی h_{ML} است که کمترین خطای مربعی را دارد، پس محتمل‌ترین فرضیه است. توجه داشته باشید که محتمل‌ترین فرضیه حتماً فرضیه‌ی درست نیست، زیرا که مجموعه های آموزشی محدود و خطا دار هستند.

---

[1] maximum likelihood
[2] least squared error
[3] maximum likelihood



شکل ۶.۲ یادگیری تابع حقیقی مقدار.
تابع هدف f با خط نشان داده شده است. نمونه های آموزشی $< \boldsymbol{x_i}, \boldsymbol{d_i} >$ با فرض اینکه خطایی با توزیع نرمال با میانگین صفر دارند در نظر گرفته شده‌اند. خط چین تابعی خطی را نشان می‌دهد که میزان خطای مربعی را مینیمم می‌کند. بنابراین این فرضیه محتمل‌ترین فرضیه، $\boldsymbol{h_{ML}}$، بر اساس ۵ نمونه‌ی آموزشی موجود است.

قبل از اینکه به اثبات محتمل‌ترین بودن فرضیه‌هایی که خطای مربعی را مینیمم می‌کنند در شرایط مذکور بپردازیم؛ ابتدا بیایید دو مفهوم را از تئوری احتمال مرور کنیم: چگالی احتمال و توزیع نرمال. ابتدا برای بحث روی متغیر های تصادفی پیوسته مثل e، ابتدا باید چگالی احتمال را معرفی کنیم. دلیل اولیه این پیش زمینه‌ها این است که می‌خواهیم مجموع احتمالات روی تمامی مقادیر ممکن متغیر تصادفی یک باشد. در این حالت که متغیر های تصادفی پیوسته هستند، تعیین احتمال را نمی‌توان با نسبت دادن یک احتمال به هر یک از مقادیر ممکن متغیر تصادفی انجام داد. به جای ان، از چگالی احتمال[1] برای مقادیر تصادفی حقیقی مثل e استفاده می‌کنیم و انتگرال روی کل چگالی احتمال را مساوی یک قرار می‌دهیم. در کل از حرف کوچک p برای نشان دادن تابع چگالی احتمال استفاده می‌کنیم و احتمال را با حرف بزرگ P نشان می‌دهیم (گاهی اوقات این مقدار جرم احتمال[2] نیز نامیده می‌شود). چگالی احتمال $p(x_0)$، $\frac{1}{\varepsilon}$ برابر مقدار احتمال اینکه متغیر تصادفی در بازه ای $[x_0, x_0 + \varepsilon)$ زمانی که $\varepsilon \to 0$ قرار بگیرد است.

چگالی احتمال:

$$p(x_0) \equiv \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} P(x_0 \leq x < x_0 + \varepsilon)$$

[1] probability density
[2] probability mass

دوم اینکه e را در مسئله طوری تعریف کردیم که از توزیع احتمال نرمال پیروی می‌کند. توزیع احتمال نرمال، توزیع احتمالی هموار و زنگی شکل است که می‌توان آن‌را با میانگین μ و انحراف معیار σ کاملاً مشخص کرد. برای تعریف دقیق‌تر به جدول ۵.۴ مراجعه کنید.

حال با داشتن این پیش زمینه‌ها می‌توانیم به موضوع اصلی بر گردیم: در شرایط مذکور، نشان می‌دهیم که فرضیه‌هایی که خطای مربعی را مینیمم می‌کنند در واقع همان محتمل‌ترین فرضیه‌ها هستند. ابتدا محتمل‌ترین تابع را با استفاده از رابطه‌ی 6.3 مشخص می‌کنیم، با این تفاوت که توزیع احتمال در این رابطه را با p نشان می‌دهیم.

$$h_{ML} = \arg\max_{h \in H} p(D|h)$$

مثل قبل، فرض می‌کنیم که مجموعه ای از نمونه های آموزشی مثل $< x_1 \dots x_n >$ با مقدار تابع هدفشان D، $D = < d_1 \dots d_2 >$ داریم. در اینجا $d_i = f(x_i) + e_i$. با فرض اینکه نمونه های آموزشی کاملاً مستقل از فرضیه‌ی h هستند P(D|h) را می‌توان بر حسب $p(d_i|h)$ ها نوشت

$$h_{ML} = \arg\max_{h \in H} \prod_{i=1}^{m} p(d_i|h)$$

با دانستن اینکه e_i ها از توزیع نرمال با میانگین صفر و واریانس مجهول σ^2 پیروی می‌کنند، هر d_i نیز باید از توزیع نرمالی با واریانس σ^2 و میانگین $f(x_i)$، به جای صفر، پیروی کند. بنابراین $p(d_i|h)$ را می‌توان به صورت توزیع نرمالی با واریانس σ^2 و میانگین $\mu = f(x_i)$ نوشت. بیایید فرمول این توزیع نرمال را که $p(d_i|h)$ را توصیف می‌کند بنویسیم، ابتدا فرمولی که در جدول ۵.۴ آمده را می‌نویسیم و مقادیر μ و σ^2 را جایگزین می‌کنیم. چون رابطه ای برای d_i با فرض اینکه فرضیه‌ی h توصیف درست از تابع هدف f است می‌نویسیم، خواهیم داشت که $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \arg\max_{h \in H} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-\mu)^2}$$

$$= \arg\max_{h \in H} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-h(x_i))^2}$$

حال از تبدیلی استفاده می‌کنیم که در اکثر محاسبات محتمل‌ترین‌ها متداول است: به جای ماکزیمم کردن مقدار کل عبارت، لگاریتم آن‌را که بسیار ساده تر است ماکزیمم می‌کنیم. زیرا که تابع ln p تابعی یکنواخت و صعودی از p است. بنابراین ماکزیمم کردن ln p باعث ماکزیمم شدن خود p می‌شود.

$$h_{ML} = \arg\max_{h \in H} \sum_{i=1}^{m} ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\left(d_i - h(x_i)\right)^2$$

جمله‌ی اول مستقل از h است و بنابراین می‌توان آن‌را حذف کرد،

$$h_{ML} = \arg\max_{h\in H} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}\left(d_i - h(x_i)\right)^2$$

ماکزیمم کردن این کمیت منفی مشابه مینیمم کردن مقدار مثبت آن است،

$$h_{ML} = \arg\min_{h\in H} \sum_{i=1}^{m} \frac{1}{2\sigma^2}\left(d_i - h(x_i)\right)^2$$

و دوباره می‌توان ثابتی که مستقل از h است را حذف کرد و داریم:

$$h_{ML} = \arg\min_{h\in H} \sum_{i=1}^{m} \left(d_i - h(x_i)\right)^2 \qquad (6.6)$$

رابطه‌ی ۶.۶ نشان می‌دهد که محتمل‌ترین فرضیه h_{ML} فرضیه ای است که مجموع خطا های مربعی بین مقادیر هدف نمونه های آموزشی d_i و پیش بینی فرضیه $h(x_i)$ را مینیمم کند. این نتیجه گیری‌ها با این فرض بود که نمونه های آموزشی، d_i ها، مقادیر تابع هدف بـه اضـافه‌ی مقدار خطای تصادفی با توزیع نرمال و میانگین صفر هستند. همان طور که استخراج عبارت بالا نیز نشان می‌دهد، مقدار جمله‌ی مربعی خطـای $\left(d_i - h(x_i)\right)^2$ مستقیماً از توزیع نرمال ناشی شده است. با استفاده از دیگر توزیع‌های خطا می‌توان تعریف‌های دیگری برای خطـا بدسـت آورد.

توجه داشته باشید که ساختار اشتقاق بالا شامل انتخاب فرضیه ای که لگاریتم محتمـل بـودن (ln p(D|h)) را حـداکثر مـی‌کنـد بـه عنـوان محتمل‌ترین فرضیه نیز می‌شود. همان طور که پیش‌تر نیز گفته شد، این مشابه این است که محتمل بودن (ln p(D|h)) را حداکثر کنیم. این روش کار با لگاریتم محتمل بودن[1] در بسیار از بررسی‌های بیزی مورد استفاده قرار می‌گیرد، زیرا که کار با لگاریتم محتمل بودن بسیار ساده تر از کار با خود محتمل بودن است. البته، همان طور که قبلاً هم گفته شد، محتمل‌ترین فرضیه همیشه فرضیه‌ی *MAP* نیست مگر اینکـه احتمـال اولیه‌ی تمامی فرضیه‌ها مساوی فرض شود.

چرا استفاده از توزیع نرمال برای مدل سازی نویز یا همان خطای نمونه‌ها استفاده می‌کنیم؟ یکی از دلایلی که لازم است حتماً ذکـر شـود، ایـن است که بررسی را از نظر ریاضی بسیار ساده تر می‌کند. دلیل دوم این است که این توزیع توزیعی هموار است و توزیع‌های زنگی شکل تخمـین خوبی برای بسیاری از انواع خطاها در سیستم‌های فیزیکی هستند. در واقع، طبق قضیه‌ی حد مرکزی که در فصل ۵ توضـیح داده شـد، مجمـوع تعداد زیادی از متغیر های مستقل و هم توزیع بدون توجه به نوع توزیع از توزیع نرمال پیروی می‌کند. این ثابت مـی‌کنـد کـه خطـا کـه خـود از مجموع تعداد زیادی متغیر مستقل و با ضریب توزیع یکسان تولید می‌شود از توزیع نرمال پیروی خواهد می‌کنـد. البتـه در واقعیـت، مؤلفـه هـای مختلفی که در نویز تأثیر گذارند همگی از یک توزیع پیروی نمی‌کنند، که در این شرایط این قضیه توجیهی برای استفاده از این توزیع نیست.

[1] likelihood

مینیمم کردن مجموع خطای مربعی روشی متداول در بسیاری از شبکه های عصبی، منحنی‌های تخمین و ... در تخمین توابع حقیقی مقدار است. فصل ۴ روش شیب نزول را که مینیمم کردن خطای مربعی در شبکه های عصبی را با آن انجام می‌دهیم مفصلاً توضیح داده است.

بد نیست که قبل از اتمام بحث رابطه‌ی بین محتمل‌ترین فرضیه و فرضیه ای که خطای مربعی را مینیمم می‌کند، بعضی محدودیت‌ها این شرایط مسئله را ذکر کنیم. بررسی بالا فقط خطا در تابع هدف نمونه های آموزشی در نظر گرفته شده است و از خطای خود ویژگی‌هایی که نمونه را توصیف می‌کنند صرف نظر شده بود. برای مثال، اگر مسئله یادگیری پیش بینی وزن افراد بر اساس سن و قدشان باشد، در شرایط ذکر شده فقط می‌توان خطا را برای وزن در نظر گرفت و مقادیر سن و قد دقیق فرض می‌شوند. بررسی زمانی پیچیده تر می‌شود که فرض‌های ساده کننده حذف شوند.

۶.۵ محتمل‌ترین فرضیه برای مسائل پیش بینی

در تعریف مسئله‌ی قسمت قبلی محتمل‌ترین فرضیه را فرضیه ای مشخص کردیم که مجموع خطای مربعی را بر روی نمونه های آموزشی مینیمم می‌کند. در این بخش معیاری مشابه برای تعریف مسئله‌ی دیگری که در شبکه های عصبی متداول است بیان می‌کنیم: یادگیری پیش بینی احتمالات.

حالتی را در نظر بگیرید که در آن می‌خواهیم تابعی غیر قطعی (احتمالی) $f: X \to \{0,1\}$ را یاد بگیریم که دو خروجی گسسته دارد. برای مثال، فضای نمونه ای X ممکن است توصیف بیماران با علائم بیماریشان باشد، و تابع هدف f(x) زمانی که بیمار زنده بماند ۱ و در غیر این صورت ۰ باشد. یا به طور مشابه X می‌تواند توصیف مراجعین دریافت وام با وضعیت حسابشان در گذشته باشد و f(x) زمانی که وام بعدی کامل پرداخت می‌شود ۱ و در غیر این صورت ۰ باشد. برای مثال، در مجموعه ای از بیماران که علائم مشترکی دارند ۹۲٪ درصد زنده می‌مانند و ۸٪ جان سالم به در نمی‌برند. این عدم قطعیت ممکن است ناشی از ناتوانی ما را در مشاهده‌ی تمامی علائم مهم بیمار باشد یا ممکن است ناشی از یک فرایند تصادفی در پیشرفت بیماری باشد. جدا از اینکه منشأ مشکل چیست، ما تابع هدف f(x) را داریم که به صورت احتمالی روی این ورودی عمل می‌کند.

با این تعریف مسئله ممکن است از یک شبکه‌ی عصبی (یا تخمین زننده‌ی توابع حقیقی مقدار دیگر) که خروجی‌اش احتمال f(x)=1 باشد استفاده کنیم. به عبارت دیگر، ما دنبال یادگیری تابع هدف $f': X \to [0,1]$ هستیم که در آن $f'(x)=P(f(x)=1)$. در مثال بالا، اگر آن علائم غیر قابل تمیز را داشته باشیم به احتمال ۹۲٪ بیمار زنده می‌ماند، پس $f'(x)=0.92$ که یعنی احتمال اینکه f(x) برابر با ۱ باشد ۹۲٪ است، و احتمال اینکه f(x) برابر با ۰ باشد، ۸٪ است.

چگونه می‌توان f' را با روشی مثل شبکه های عصبی یاد گرفت؟ یکی از راه‌های غیر هوشمندانه جمع کردن تعداد تکرار ۱ ها و ۰ های تابع برای هر نمونه‌ی ممکن x و آموزش شبکه‌ی عصبی با نسبت این تعداد تکرارهاست. همان طور که در ادامه نیز خواهیم دید، به جای این کار می‌توان از خود نمونه های آموزشی f برای آموزش شبکه‌ی عصبی استفاده کرد و محتمل‌ترین فرضیه برای f' را بدست آورد.

چه معیاری را بهینه می‌کنیم تا محتمل‌ترین فرضیه در این تعریف مسئله را بیابیم؟ برای جواب این سؤال ابتدا باید رابطه ای برای P(D|h) پیدا کنیم. بیایید فرض کنیم که نمونه های آموزش D به فرم $D = \{< x_1, d_1 > \cdots < x_m, d_m >\}$ هستند که در آن d_i ها مقدار مشاهده شده ی $f(x_i)$ است.

با توجه به آنچه درباره‌ی محتمل‌ترین فرضیه گفته شد، مینیمم خطای مربعی قسمت قبل، فرض کردیم که نمونه های $< x_1 \dots x_m >$ ثابتند. تا بتوان داده‌ها را فقط با مقدار هدفشان، d_i بررسی کرد. با وجود اینکه می‌توانستیم فرض دیگری در این تعریف مسئله‌ی جدید داشته باشیم، بیایید با همین فرض قبلی ادامه دهیم تا نشان دهیم این چنین فرض‌هایی در نتیجه‌ی حاصل اثری ندارند. بنابراین فرض می‌کنیم که x_i و d_i متغیر های تصادفی هستند و هر نمونه‌ی آموزشی مستقل ایجاد شده است پس می‌توانیم P(D|h) را به صورت زیر بنویسیم:

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) \tag{6.7}$$

باز هم فرض می‌کنیم که احتمال مواجهه با هر نمونه مثل x_i مستقل از h است. برای مثال، احتمال اینکه در مجموعه‌ی آموزشی بیمار x_i را داشته باشیم مستقل از فرضیه‌ی ما درباره‌ی احتمال زنده ماندن است (با این وجود البته احتمال زنده ماندن d_i خیلی به h مربوط نیست، ارتباط بین مجموعه‌ی آموزشی و فرضیه انکار ناشدنی است). زمانی که x از h مستقل باشد می‌توانیم رابطه‌ی بالا به رابطه‌ی زیر ساده کنیم، (با استفاده از قانون جدول ۶.۱)،

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) = \prod_{i=1}^{m} P(d_i|h, x_i)P(x_i) \tag{6.8}$$

حال احتمال $P(d_i|h, x_i)$ یا احتمال مشاهده‌ی $d_i = 1$ برای تک نمونه‌ی x_i با فرض اینکه فرضیه‌ی h درست است چیست؟ با توجه به اینکه h فرضیه‌ی ما از تابع هدفی است که احتمالات را محاسبه می‌کند، $P(d_i = 1|h, x_i) = h(x_i)$ و در کل،

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & if\ d_i = 1 \\ (1 - h(x_i)) & if\ d_i = 0 \end{cases} \tag{6.9}$$

برای جایگزینی این رابطه در رابطه‌ی ۶.۸ برای P(D|h) بیایید ابتدا این رابطه را به فرم ریاضی‌وار تری بنویسیم،

$$P(d_i|h, x_i) = h(x_i)^{d_i}\left(1 - h(x_i)\right)^{1-d_i} \tag{6.10}$$

به سادگی می‌توان نشان داد که دو رابطه‌ی ۶.۹ و ۶.۱۰ هم ارزند. توجه داشته باشید که زمانی که $d_i = 1$ عبارت دوم رابطه‌ی ۶.۱۰، $\left(1 - h(x_i)\right)^{1-d_i}$ مساوی یک می‌شود بنابراین خواهیم داشت که $P(d_i = 1|h, x_i) = h(x_i)^{d_i}$ که هم ارز حالت اول رابطه‌ی ۶.۹ است، به طور مشابه می‌توان نشان داد که برای $d_i = 0$ نیز دو رابطه با هم، هم ارزند.

می‌توان از رابطه‌ی ۶.۱۰ برای جایگزینی $P(d_i|h, x_i)$ در رابطه‌ی ۶.۸ استفاده کرد،

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i}\left(1 - h(x_i)\right)^{1-d_i} P(x_i) \tag{6.11}$$

حال می‌توانیم رابطه‌ی محتمل‌ترین فرضیه را بنویسیم،

$$h_{ML} = \arg\max_{h\in H} \prod_{i=1}^{m} h(x_i)^{d_i}\left(1-h(x_i)\right)^{1-d_i} P(x_i)$$

جمله‌ی آخری مستقل از h است و می‌توان آن‌را حذف کرد،

$$h_{ML} = \arg\max_{h\in H} \prod_{i=1}^{m} h(x_i)^{d_i}\left(1-h(x_i)\right)^{1-d_i} \qquad (6.12)$$

عبارت سمت راست رابطه‌ی ۶.۱۲ را می‌توان در تعمیم توزیع دو جمله ای در جدول ۵.۳ دید. عبارت رابطه‌ی 6.12 احتمال ظهور برآمد $< d_1 \dots d_m >$ را با فرض اینکه هر سکه‌ی x_i احتمال شیر آمدن $h(x_i)$ را داشته باشد را نشان می‌دهد. توجه داشته باشید که توزیع دو جمله ای که در جدول ۵.۳ آمد مشابه این رابطه است، اما فرض دیگری نیز دارد، احتمال شیر آمدن برای تمامی سکه‌ها را مساوی فرض می‌کند $\left(h(x_i) = h(x_j), \forall i,j\right)$. اما در هر دو حالت فرض می‌کنیم که برآمد پرتاب سکه‌ها ناسازگارند، فرضی که در تعریف مسئله فعلی ما نیز صدق می‌کند.

مشابه گذشته، کار با لگاریتم محتمل بودن راحت‌تر از خود محتمل بودن است پس داریم:

$$h_{ML} = \arg\max_{h\in H} \sum_{i=1}^{m} d_i ln h(x_i) + (1-d_i)\ln\left(1-h(x_i)\right) \qquad (6.13)$$

رابطه‌ی ۶.۱۳ کمیتی را نشان می‌دهد که برای پیدا کردن محتمل‌ترین فرضیه در تعریف مسئله‌ی فعلی ماکزیمم می‌کنیم. این نتیجه مشابه نتیجه‌ی قبلی ما در مینیمم کردن مجموع خطای مربعی محتمل‌ترین فرضیه در تعریف مسئله‌ی قبلی است. به شباهت بین رابطه‌ی 6.13 و فرم کلی تابع آنتروپی $-\sum_i p_i log p_i$ که در فصل ۳ آمد توجه کنید. بخاطر این شباهت، قرینه‌ی عبارت بالا گاهی آنتروپی دورگه[1] نامیده می‌شود.

۶.۵.۱ شیب نزول برای پیدا کردن محتمل‌ترین فرضیه در یک شبکه‌ی عصبی

در بالا نشان دادیم که با ماکزیمم کردن کمیت رابطه‌ی ۶.۱۳ محتمل‌ترین فرضیه بدست خواهد آمد. بیایید این کمیت را با اختصار G(h,D) نشان دهیم. در این بخش قانونی برای آموزش وزن‌ها[2] برای شبکه های عصبی بدست خواهیم آورد که G(h,D) را توسط روش شیب نزول ماکزیمم می‌کند.

همان طور که در فصل ۴ نیز بحث شد، گرادیان G(h,D) توسط بردار مشتق‌های جزئی G(h,D) نسبت به وزن‌های مختلف شبکه که فرضیه‌ی h را مشخص می‌کند ایجاد می‌شود (برای توضیح کامل درباره‌ی جزئیات جستجوی شیب نزول و واژگان بکار رفته به فصل ۴ مراجعه کنید). در این قسمت، مشتق جزئی G(h,D) نسبت به وزن w_{jk} که از واحد k ام به واحد j ام است به فرم زیر است:

---

[1] cross entropy
[2] weight training

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \frac{\partial G(h,D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$= \sum_{i=1}^{m} \frac{\partial (d_i lnh(x_i) + (1-d_i)\ln(1-h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{\partial h(x_i)(1-h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}} \tag{6.14}$$

برای ساده نگه داشتن محاسبات، فرض کنید که شبکه‌ی عصبی ما از یک لایه واحد سیگموید تشکیل شده و در این حالت داریم که

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x')x_{ijk} = h(x_i)(1-h(x_i))x_{ijk}$$

در این رابطه x_{ijk}، k امین ورودی به واحد j برای i امین نمونه‌ی آموزشی است، و σ'(x) مشتق تابع سیگموید است (به فصل ۴ رجوع کنید). بالاخره، این رابطه را در رابطه‌ی ۶.۱۴ جایگذاری می‌کنیم و رابطه ای برای مؤلفه های گرادیان بدست می‌آوریم،

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} (d_i - h(x_i))x_{ijk}$$

چون بیشتر به دنبال ماکزیمم P(D|h) هستیم تا مینیمم به جای شیب نزول از جستجوی شیب صعود[1] استفاده می‌کنیم. در هر حلقه جستجو بردار توسط قانون زیر به سمت گرادیان تصحیح می‌شود.

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

که داریم،

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))x_{ijk} \tag{6.15}$$

و در این رابطه نیز η مقدار کوچک و مثبت است که اندازه‌ی قدم‌ها در جستجوی شیب صعود را مشخص می‌کند.

جالب است که این قانون تغییر وزن‌ها را با قانون تغییر وزن الگوریتم Backpropagation که مجموع خطای مربعی بین پیش بینی و مقدار اصلی را مینیمم می‌کرد مقایسه کنیم. قانون تغییر وزن برای واحد های خروجی در Backpropagation با نشانه گذاری این فصل به شکل زیر است،

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

[1] gradient ascent

که در آن

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} h(x_i)(1 - h(x_i))\big(d_i - h(x_i)\big)x_{ijk}$$

توجه دارید که این رابطه جز در جمله‌ی $h(x_i)(1 - h(x_i))$ که از تابع سیگموید ناشی شده کاملاً شبیه رابطه‌ی ۶.۱۵ است.

خلاصه اینکه، این دو قانون تغییر وزن هر دو در تعریف مسئله‌ی خودشان به سمت محتمل‌ترین فرضیه همگرا می‌شوند. قانونی که مجموع خطا های مربعی را مینیمم می‌کند با فرض اینکه خطا های داده های آموزشی را می‌توان با توزیع نرمال مدل سازی کرد به دنبال محتمل‌ترین فرضیه می‌گردد. قانونی که آنتروپی دورگه را مینیمم می‌کند با فرض اینکه مقادیر منطقی مشاهده شده احتمالی (و نه قطعی) هستند به دنبال محتمل‌ترین فرضیه برای تابع پیش بینی احتمال بر حسب نمونه‌ها می‌گردد.

٦.٦ قانون کمترین طول توضیح[1]

با توجه به آنچه در فصل ۳ درباره‌ی تیغ Ocam گفته شد، یک بایاس استقرایی متداول، به فرم "توضیحی که کوتاه تر است را در مورد داده های مشاهده شده قبول کن" است. در آن فصل درباره‌ی ضرر های توضیحات بلند با توجه به تیغ Ocam استدلال کردیم. در اینجا با دیدی بیزی به این موضوع می‌پردازیم و قانونی مشابه به نام قانون کمترین طول توضیح (MDL) را بررسی خواهیم کرد.

انگیزه‌ی ایجاد قانون کمترین طول توضیح تفسیر تعریف h_{MAP} با مفاهیم اولیه‌ی تئوری اطلاعات است. دوباره تعریف نه چندان ناآشنای h_{MAP} را به خاطر بیاورید.

$$h_{MAP} = \arg\max_{h \in H} P(D|h)P(h)$$

این رابطه را می‌توان به صورت معادل با log_2 آن نیز نشان داد،

$$h_{MAP} = \arg\min_{h \in H} -\log_2 P(D|h) - \log_2 P(h) \qquad (6.16)$$

جالب است که رابطه‌ی ۶.۱۶ را می‌توان طوری تفسیر کرد که فرضیه های کوتاه‌تر ارجح‌ترند، با فرض اینکه یک طرح نمایش خاص برای کد کردن فرضیه‌ها و داده‌ها استفاده کنیم. برای توضیح این، بیایید ابتدا یک نتیجه‌ی اساسی تئوری اطلاعات را معرفی کنیم: مسئله‌ی طراحی کدی برای ارسال پیام‌های تصادفی، را که در آن احتمال ارسال پیام i مقدار p_i است را در نظر بگیرید. در اینجا علاقه‌ی ما به فشرده‌ترین کد ممکن است؛ به عبارت دیگر علاقه‌ی ما به کدی است که امید تعداد بیت‌هایی که باید ارسال شوند تا یک پیام تصادفی فرستاده شود مینیمم کند. واضح است که برای مینیمم کردن امید طول کد ارسالی باید کد های کوتاه تر را به پیام‌هایی اختصاص دهیم که احتمال بیشتری دارند. (Shannon and Weaver 1949) نشان دادند که کد بهینه (کدی که امید تعداد بیت‌های ارسالی را مینیمم می‌کند) به پیام i، $\log_2 p_i$ بیت برای کد کردن اختصاص می‌دهد. به این تعداد بیت که برای کد کردن پیام i توسط کد C لازم است طول توضیح پیام i بر اساس C نیز می‌گویند و با $L_C(i)$ آن‌را نشان می‌دهند.

[1] Minimum description length

بیایید حالا رابطه‌ی ۶.۱۶ را با توجه به نتیجه‌ی بالا از تئوری کد سازی بررسی کنیم.

- $-\log_2 P(h)$ اندازه‌ی توضیح h بر اساس کد بهینه‌ی تمامی فضای فرضیه ای H است. به عبارت دیگر، این مقدار اندازه‌ی توضیحات فرضیه‌ی h با استفاده از نمایش بهینه است. در نماد گذاری فعلی $L_{C_H}(h) = -\log_2 P(h)$، که در آن C_H کد بهینه برای کد کردن فضای فرضیه ای H است.
- $-\log_2 P(D|h)$ اندازه‌ی توضیح داده های آموزشی D با معلوم بودن h توسط کد بهینه است. در نماد گذاری فعلی $L_{C_{D|h}}(D|h) = -\log_2 P(D|h)$، که در آن $C_{D|h}$ کد بهینه برای توضیح داده های D با فرض اینکه فرستنده و گیرنده هر دو مطلع از h هستند است.
- بنابراین می‌توانیم رابطه‌ی ۶.۱۶ را برای تعریف h_{MAP} بازنویسی کنیم و بگوییم h_{MAP} فرضیه ای مثل h است که مجموع طول توضیحات فرضیه‌ها به علاوه‌ی طول توضیحات داده‌ها با معلوم بودن فرضیه را مینیمم می‌کند.

$$h_{MAP} = \arg\min_h L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

در این رابطه C_H و $C_{D|h}$ به ترتیب کد های بهینه برای H و D با معلوم بودن h هستند.

قانون کمترین طول توضیح (MDL) توصیه می‌کند که فرضیه‌هایی را انتخاب کنیم که مجموع این دو طول توضیح را حداقل کنند. البته برای بکار بردن این قانون در عمل باید کد سازی یا نمایش خاصی را که با عمل یادگیری متناسب است انتخاب کنیم. با فرض اینکه ما از کد های C_1 و C_2 برای نمایش فرضیه‌ها و داده‌ها با معلوم بودن فرضیه استفاده می‌کنیم، می‌توان MDL را به صورت زیر بیان کرد،

قانون کمترین طول توضیح: فرضیه‌ی h_{MDL} را انتخاب کن،

$$h_{MDL} = \arg\min_{h \in H} L_{C_1}(h) + L_{C_2}(D|h) \qquad (6.17)$$

بررسی بالا نشان می‌دهد که اگر ما C_1 را برای کد سازی بهینه‌ی فرضیه‌ها، C_H، و C_2 را برای کد سازی بهینه‌ی دادها، $C_{D|h}$، انتخاب کنیم داریم $h_{MDL} = H_{MAP}$.

به صورت مفهومی، می‌توان به قانون MDL به فرم ترجیح متد های کوتاه‌تر برای کد سازی دوباره‌ی داده های آموزشی نگاه کرد که در آن هر دو معیار اندازه‌ی فرضیه و هزینه‌ی اضافی کد سازی داده‌ها به شرط معلوم بودن فرضیه در نظر گرفته می‌شود.

بیایید مثالی را در نظر بگیریم. فرض کنید قصد داریم از قانون MDL برای مسئله‌ی یادگیری درخت‌های تصمیم از داده های آموزشی‌ای استفاده کنیم. برای نمایش فرضیه C_1 و داده های C_2 چه نمایشی را باید در نظر بگیریم؟ برای C_1 می‌توان به طور طبیعی یکی از کد سازی‌های واضح درخت تصمیم، که در آن طول توضیح با افزایش تعداد گره های درخت و تعداد یال‌ها افزایش می‌یابد را انتخاب کرد. اما چگونه باید با معلوم بودن یک درخت فرضیه‌ی خاص مجموعه‌ی داده های C_1 را کد کرد. برای ساده نگه داشتن موضوع، فرض کنید که سری نمونه های $< x_1 \dots x_n >$ برای فرستنده و گیرنده معلوم باشد، پس تنها چیز باقیمانده برای ارسال دسته بندی‌های $< f(x_1) \dots f(x_n) >$ است. (توجه دارید که هزینه‌ی ارسال خود نمونه‌ها از درستی فرضیه مستقل است، پس به هر حال تأثیری بر انتخاب h_{MDL} ندارد). حال اگر دسته بندی‌های $< f(x_1) \dots f(x_n) >$ همان پیش بینی‌های فرضیه باشد، دیگر نیازی به ارسال اطلاعات در مورد نمونه‌ها نیست (گیرنده می‌تواند این مقادیر را با فرضیه ای که دریافت کرده محاسبه کند). پس بنابراین طول توضیحات لازم

با داشتن فرضیه در این حالت صفر است. در چنین شرایطی اگر نمونه‌هایی توسط h اشتباه دسته بندی شده باشند، لازم است پیغامی مبنی بر دسته بندی اشتباه این نمونه‌ها (طول این پیغام حداکثر $\log_2 n$ بیت خواهد بود) را به همراه دسته بندی درست آن‌ها ارسال کنیم (این کار را می‌توان با پیغامی با حداکثر طول $\log_2 k$ انجام داد که در آن k تعداد دسته بندی‌های ممکن هر نمونه است). در چنین شرایطی فرضیه‌ی h_{MDL} تحت کد سازی C_1 و C_2 فرضیه ای است که کمترین مجموع طول توضیح را لازم داشته باشد.

بنابراین قانون MDL راهی برای ارزیابی پیچیدگی فرضیه‌ها با تعداد اشتباه های فرضیه ارائه می‌کند. ممکن است این معیار فرضیه ای کوتاه‌تر را که اشتباهات کمی دارد را نسبت به یک فرضیه بلند تر که اشتباهی ندارد ترجیح دهد. MDL از این نظر، متدی مناسب برای برخورد با مسئله‌ی overfit است.

(Quinlar and Rivest 1989) آزمایشاتی را با استفاده از قانون MDL برای تشخیص بهترین اندازه‌ی درخت تصمیم انجام داده‌اند. آن‌ها گزارش داده‌اند که متد مبتنی بر MDL درخت‌هایی را ایجاد می‌کند که دقتی قابل مقایسه با درخت‌های خروجی الگوریتم‌های فصل ۳ دارند. (Mehta et al. 1995) نیز روش دیگری مبتنی بر MDL برای هرس درخت تصمیم ارائه می‌کند و آزمایش‌هایی را تشریح کرده که در آن روش مبتنی بر MDL نتایج قابل مقایسه ای با روش‌های معمول را می‌دهد.

چه نتیجه گیری‌ای را باید از بررسی قانون کمترین طول توضیح بگیریم؟ آیا این اثباتی بر این که تمامی فرضیه های کوتاه‌تر ارجحند است؟ خیر. بلکه ما اثبات کردیم که اگر نمایش فرضیه طوری انتخاب شود که کد سازی فرضیه‌ی h، $-\log_2 P(h)$ باشد و اگر کد سازی استثنا به گونه ای باشد که طول کد D با شرط معلوم بود h، $-\log_2 P(h|D)$، آنگاه قانون MDL فرضیه ای MAP خروجی خواهد داد. با این وجود، برای نشان دادن برقراری چنین شرطی باید تمامی احتمالات اولیه‌ی P(h) و P(D|h) را داشته باشیم. هیچ دلیلی برای این وجود ندارد که باور داشته باشیم که MDL برای هر کد سازی دلخواه C_1 و C_2 بر قرار است. ممکن گاهی برای طراح انسانی مشخص کردن نمایشی خاص برای دانش در مورد احتمالات نسبی فرضیه‌ها راحت‌تر از نمایش کامل احتمال دقیق هر یک از فرضیه‌ها باشد. توصیفات به کار رفته در ادبیات کاربرد MDL در مسائل یادگیری کاربردی گاهی شامل معیارهایی می‌شود که فرم خاصی از کد سازی C_1 و C_2 را توجیه می‌کند.

۶.۷ دسته بندی کننده‌ی بهینه‌ی بیز[1]

تا اینجا به سؤال "محتمل‌ترین فرضیه با داشتن داده های آموزشی کدام است؟" پرداختیم، در واقع، این سؤال بیشتر شبیه این سؤال است که "محتمل‌ترین دسته بندی کننده نمونه های جدید با داشتن داده های آموزشی کدام است؟". با وجود اینکه ممکن است به نظر برسد که این سؤال دوم را می‌توان با اعمال فرضیه‌ی MAP به نمونه های جدید جواب داد، کاری بهتر ممکن است.

برای ایجاد شهود فضای فرضیه ای را در نظر بگیرید که سه فرضیه‌ی h_1، h_2 و h_3 را شامل می‌شود. فرض کنید که احتمال ثانویه‌ی این فرضیه با داده های آموزشی به ترتیب 4. و 3. و 3. است. بنابراین، h_1 فرضیه‌ی MAP است. حال فرض کنید که نمونه‌ی جدید x به ما داده می‌شود که توسط h_1 مثبت و توسط دو فرضیه‌ی h_2 و h_3 منفی دسته بندی می‌شود. با در نظر گرفتن تمامی فرضیه‌ها نمونه‌ی x به احتمال 4. مثبت است (احتمال مربوط به فرضیه‌ی h_1)، و به احتمال 6. منفی است. محتمل‌ترین دسته بندی (منفی) در این مثال با دسته بندی MAP متفاوت است.

[1] bayes optimal classifier

در کل محتمل‌ترین دسته بندی نمونه‌ی جدید از ترکیب پیش بینی‌های همه‌ی فرضیه‌ها بدست می‌آید، فقط هـر فرضـیه بـه انـدازه‌ی احتمـال ثانویه‌اش در این دسته بندی تأثیر گذار است. اگر دسته بندی ممکن نمونه‌ی جدید v_j عضو مجموعه‌ی V باشـد، $P(v_j|D)$، احتمـال اینکـه دسته بندی v_j برای نمونه‌ی جدید درست باشد به صورت زیر است،

$$P(v_j|D) = \sum_{h_j \in H} P(v_j|h_i)P(h_i|D)$$

دسته بندی ی بهینه‌ی نمونه‌ی جدید مقدار v_j است که با آن $P(v_j|D)$ ماکزیمم می شود،

دسته بندی ی بهینه‌ی بیز:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \qquad (6.18)$$

برای شهود در مثال بالا، مجموعه‌ی دسته بندی‌های نمونه‌ی جدید $V = \{\oplus, \ominus\}$ است و

$$P(h_1|D) = .4\,, P(\ominus\,|h_1) = 0, P(\oplus\,|h_1) = 1$$

$$P(h_2|D) = .3\,, P(\ominus\,|h_2) = 1, P(\oplus\,|h_2) = 0$$

$$P(h_3|D) = .3\,, P(\ominus\,|h_3) = 1, P(\oplus\,|h_3) = 0$$

بنابراین،

$$\sum_{h_i \in H} P(\oplus\,|h_i)P(h_i|D) = .4$$

$$\sum_{h_i \in H} P(\ominus\,|h_i)P(h_i|D) = .6$$

و

$$\arg\max_{v_j \in \{\oplus, \ominus\}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

هر سیستمی که نمونه های جدید را با رابطه‌ی ۶.۱۸ دسته بندی کند دسته بندی کننده‌ی بهینه‌ی بیز[1] یا یادگیر بهینه‌ی بیز[2] نامیـده مـی‌شـود. هیچ متد دسته بندی دیگری با همان فضای فرضیه ای و همان دانش اولیه نمی‌تواند به طور متوسط بازده بهتری داشته باشد. این متـد احتمـال اینکه نمونه‌ی جدید درست دسته بندی شود را با معلوم بودن داده های موجود و فضای فرضیه ای احتمالات اولیه‌ی فرضیه‌ها حداکثر می‌کند.

[1] Bayes optimal classifier
[2] Bayes optimal learner

برای مثال در یادگیری مفاهیم حقیقی مقدار با استفاده از فضای ویژه، همان طور که در قسمت قبلی هم گفته شد، دسته بندی بهینه‌ی بیز نمونه های جدید با دادن وزن (احتمال ثانویه‌ی فرضیه) و رای گیری بین اعضای فضای ویژه انجام می‌گرفت.

یکی از ویژگی‌های عجیب دسته بندی کننده‌ی بهینه‌ی بیز این است که پیش بینی‌هایی که انجام می‌دهد ممکن است فرضیه ای را تشکیل دهد که حتی در H موجود نیست. تصور کنید که از رابطه‌ی ۶.۱۸ برای دسته بندی تمامی نمونه های X استفاده کرده‌ایم. این دسته بندی نمونه‌ها که بدین صورت تعریف می‌شود الزاماً با فرضیه ای مثل h در H سازگار نیست. یکی از روش‌های نگاه به این وضعیت تصور دسته بندی کننده بهینه‌ی بیز به عنوان عاملی است که فضای فرضیه ای H′ را به طرز موثری، که با فضای فرضیه ای H (که قضیه بیز روی آن اعمال شده) فرق دارد، در نظر می‌گیرد. در کل، H′ به صورت موثر فرضیه‌هایی که مقایسه ای خطی بین ترکیبات پیش بینی‌های فرضیه های مختلف H می‌کنند را شامل می‌شود.

۶.۸ الگوریتم گیبس

با وجود اینکه دسته بندی کننده‌ی بهینه‌ی بیز بهترین عملکرد ممکن را با داشتن داده های آموزشی دارد، اما اعمال آن هزینه بر است. این هزینه در محاسبه‌ی احتمال ثانویه‌ی تمامی فرضیه های H و ترکیب پیش بینی‌هایشان برای هر نمونه‌ی جدید است.

یک روش جایگزین، ولی کمتر بهینه الگوریتم گیبس (رجوع کنید به Opper and Haussler 1991) است، که به صورت زیر تعریف می‌شود:

۱. فرضیه ای مثل h از H به طور تصادفی و با توزیع احتمالات ثانویه انتخاب کن.
۲. از h برای دسته بندی نمونه‌ی جدید بعدی استفاده کن.

زمانی که نمونه‌ی جدیدی برای دسته بندی ارائه می‌شود، الگوریتم گیبس به سادگی فرضیه ای به طور تصادفی و با توزیع احتمالات ثانویه انتخاب می‌کند و دسته بندی آن‌را به عنوان خروجی می‌دهد. جالب‌تر اینکه، می‌توان نشان داد که در شرایطی امید تعداد دسته بندی‌های غلط این الگوریتم حداکثر دو برابر امید خطای دسته بندی کننده‌ی بهینه‌ی بیز است (Haussler 1994). به عبارت دقیق‌تر، مقدار امید برای تمامی مفاهیم هدف تصادفی و توزیع احتمال اولیه‌ی یادگیر محاسبه شده. در چنین شرایطی، مقدار امید خطای الگوریتم گیبس دو برابر بد تر از مقدار امید خطای دسته بندی کننده‌ی بهینه‌ی بیز است.

این نتیجه معنای جالبی در مسائل یادگیری مفهوم که قبلاً در موردشان بحث کردیم دارد. در کل، این نتیجه نشان می‌دهد که اگر یادگیر احتمالات اولیه H را یکسان فرض کند، و مفاهیم هدف نیز در واقع با چنین احتمالی انتخاب شوند، آنگاه دسته بندی نمونه‌ی بعدی با فرضیه ای که به طور تصادفی از فضای ویژه انتخاب می‌شود (با توزیعی یکنواخت)، حداکثر دو برابر امید خطای دسته بندی کننده‌ی بهینه‌ی بیز، امید خطا خواهد داشت. دوباره، با نمونه ای از بررسی بیزی یک الگوریتم غیر بیزی طرف هستیم که این بررسی میزان کارایی آن الگوریتم را مشخص می‌کند.

۶.۹ دسته بندی کننده‌ی ساده‌ی بیز

یکی از متد های پر کاربرد یادگیری بیزی، یادگیر ساده‌ی بیز[1] است که معمولاً دسته بندی کننده‌ی ساده‌ی بیز[2] نیز نامیده می‌شود. در بعضی کاربردها کارایی این متد قابل مقایسه با شبکه های عصبی و یادگیری درختی است. در این بخش دسته بندی کننده‌ی ساده‌ی بیز را مورد بحث و بررسی قرار می‌دهیم و در بخش بعدی آن‌را در مسئله یادگیری‌ای واقعی دسته بندی متون زبان‌های طبیعی به کار می‌بریم.

دسته بندی کننده‌ی ساده‌ی بیز در کار های یادگیری‌ای به کار می‌رود که در آن x با عطفی از مقادیر ویژگی‌ها مشخص می‌شود و تابع هدف f(x) می‌تواند هر مقدار از مجموعه‌ی V باشد. مجموعه ای از نمونه های آموزشی تابع هدف و نمونه ای جدید که با ویژگی‌هایش توصیف شده به یادگیر داده می‌شود، $< a_1, a_2 \dots a_n >$ و از آن خواسته می‌شود که مقدار تابع هدف یا دسته بندی تابع هدف را برای این نمونه‌ی جدید پیش بینی کند.

روش بیزی برای دسته بندی نمونه‌ی جدید، دسته بندی آن بر اساس محتمل‌ترین مقدار تابع هدف، v_{MAP} است با داشتن نمونه های $< a_1, a_2 \dots a_n >$ است.

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j | a_1, a_2 \dots a_n)$$

با استفاده از قضیه‌ی بیز این رابطه را بازنویسی می‌کنیم،

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)}$$

$$= \arg\max_{v_j \in V} P(a_1, a_2 \dots a_n | v_j)\, P(v_j) \qquad (6.19)$$

حال می‌توانیم دو عبارت رابطه‌ی ۶.۱۹ را بر اساس داده های آموزشی تخمین بزنیم. تخمین مقادیر $P(v_j)$ با شمارش تعداد تکرار مقدار های ویژگی هدف در بین داده های آموزشی بسیار ساده است. با این وجود، تخمین عبارتی با فرم $P(a_1, a_2 \dots a_n | v_j)$ بدین صورت ممکن نیست، مگر اینکه مجموعه‌ی داده های آموزشی مان بسیار بزرگ باشد. مشکل اینجاست که تعداد این چنین عبارت‌هایی مساوی تعداد نمونه های ممکن ضربدر تعداد مقادیر ممکن تابع هدف است. بنابراین لازم است که هر نمونه ممکن در فضای نمونه ای چندین بار مشاهده شود تا تخمین احتمال قابل اطمینان باشد.

دسته بندی کننده‌ی ساده‌ی بیز بر اساس یک فرض ساده سازی است، مقدار ویژگی‌ها با معلوم بودن مقدار هدف مستقلند. به عبارت دیگر، فرض‌هایی که با داشتن مقدار هدف نمونه می‌توان زد، احتمال مشاهده‌ی عطف $a_1, a_2 \dots a_n$ فقط وابسته به احتمال تک تک این نمونه‌هاست: $P(a_1, a_2 \dots a_b | v_j) = \prod_i P(a_i | v_i)$. با جایگذاری این رابطه در رابطه‌ی ۶.۱۹ به دسته بندی کننده‌ی ساده‌ی بیز می‌رسیم.

دسته بندی کننده‌ی ساده بیز:

---

[1] Naïve bayes learner
[2] Naïve bayes classifier

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_i) \qquad (6.20)$$

در این رابطه v_{NB} نماد مقدار هدفی خروجی دسته بندی کننده‌ی ساده‌ی بیز است. توجه دارید که در یک دسته بندی کننده‌ی ساده‌ی بیز تعداد جملات متمایز $P(a_i|v_i)$ موجود، که باید بر اساس داده های آموزشی تخمین زده شود، ضرب تعداد مقادیر ویژگی‌ها و تعداد مقادیر هدف است، این عدد در نگاه اول، نسبت به تعداد جملات ممکن $P(a_1, a_2 \dots a_b|v_j)$ بسیار کوچکتر است.

به طور خلاصه، متد یادگیری ساده‌ی بیز مرحله ای دارد که در آن جملات مختلف $P(a_i|v_i)$ و $P(v_j)$ بر اساس تعداد تکرارشان در میان نمونه های آموزشی تخمین زده می‌شوند. مجموعه‌ی این تخمین‌ها تعیین کننده‌ی فرضیه‌ی تخمینی خواهد بود. این فرضیه، برای دسته بندی نمونه های جدید رابطه‌ی ۶.۲۰ را بکار خواهد بست. هرگاه که فرض استقلال شرطی ارضا می‌شود، و دسته بندی ساده‌ی بیز v_{NB} همان دسته بندی MAP خواهد بود.

یکی از تفاوت‌های جالب دسته بندی کننده‌ی ساده‌ی بیز و دیگر متد های یادگیری بحث شده، این است که این روش جستجویی صریح در میان فرضیه های ممکن انجام نمی‌دهد (در چنین شرایطی، فضای فرضیه‌ها همان فضای مقادیر ممکن قابل نسبت به متغیر $P(v_j)$ و $P(a_i|v_j)$ است). در مقابل، فرضیه‌ها بدون جستجو و فقط با شمارش تعداد تکرار ترکیب‌های مختلف داده در میان نمونه های آموزشی ایجاد می‌شوند.

۶.۹.۱ مثالی توضیحی

بیایید دسته بندی کننده‌ی ساده‌ی بیز را به مسئله یادگیری مفهومی که در فصل یادگیری درختی مطرح شد بکار بریم: دسته بندی روزها بر اساس اینکه کسی تنیس بازی خواهد کرد یا خیر. جدول ۳.۲ مجموعه ای از ۱۴ نمونه‌ی آموزشی را برای مفهوم PlayTennis نشان می‌دهد، در اینجا روزها با ویژگی‌های Outlook، Tempereture، Humidity، و Wind توصیف می‌شوند. در اینجا از دسته بندی کننده‌ی ساده‌ی بیز و داده های آموزشی این جدول برای دسته بندی نمونه‌ی جدید زیر استفاده می‌کنیم:

<Outlook=sunny,Temperature=cool,Humidity=high,Wind=strong>

هدف در اینجا پیش بینی مقدار هدف (Yes یا No) مفهوم هدف PlayTennis برای نمونه‌ی جدید است. با مقدار گذاری رابطه‌ی ۶.۲۰ برای این کار مقدار v_{NB} به صورت زیر محاسبه می شود.

$$v_{NB} = \arg\max_{v_j \in \{yes,no\}} P(v_j) \prod_i P(a_i|v_j)$$

$$= \arg\max_{v_j \in \{yes,no\}} \begin{matrix} P(v_j)\, P(\text{Outlook} = \text{sunny}|v_j) P(\text{Temperature} = \text{cool}|v_j) \\ P(\text{Humidity} = \text{High}|v_j) P(\text{Wind} = \text{strong}|v_j) \end{matrix} \qquad (6.21)$$

توجه دارید که در عبارت آخری a_i با استفاده از مقادیر ویژگی‌های نمونه‌ی جدید نوشته شده است. برای محاسبه‌ی v_{NB} به ۱۰ احتمال نیاز داریم که از روی داده های آموزشی تخمین زده می‌شوند. ابتدا، احتمال مقادیر مختلف هدف، که می‌توان آن‌را به سادگی با شمارش تکرار مقادیر از نمونه های آموزشی استخراج کرد.

$$P(PlayTennis = yes) = \frac{9}{14} = .64$$

$$P(PlayTennis = no) = \frac{5}{14} = .36$$

به طور مشابه می‌توان احتمالات شرطی را تخمین زد. برای مثال، برای Wind = strong داریم،

$$P(Wind = strong|PlayTennis = yes) = \frac{3}{9} = .33$$

$$P(Wind = strong|PlayTennis = no) = \frac{3}{5} = .60$$

با استفاده از تخمین‌های احتمالات مذکور و تخمین مشابه دیگر ویژگی‌ها، v_{NB} را بر اساس رابطه‌ی ۶.۲۱ به صورت زیر محاسبه می‌کنیم،

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = \ .0053$$

$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = \ .0053$$

دسته بندی کننده‌ی ساده‌ی بیز احتمالات تخمینی بر اساس داده های آموزشی موجود مقدار PlayTennis = no را به این نمونه‌ی جدید اختصاص می‌دهد. علاوه بر این، با نرمالیزه کردن کمیت‌های بالا (طوری که جمعشان یک شود) می‌توان احتمال شرطی اینکه مقدار تابع هدف no باشد را حساب کرد. برای مثال فعلی، این احتمال مقدار $\frac{0206}{.0206+.0053} = .795$ است.

۶.۹.۱.۱ تخمین احتمالات

تا به حال، احتمالات را با نسبت تعداد مشاهده‌ی اتفاق به کل حالات را تخمین زدیم. برای مثال، در مثال بالا مقدار P(Wind=strong|PlayTennis=no) را با نسبت $\frac{n_c}{n}$ تخمین زدیم، در این نسبت n=5 تعداد نمونه های آموزشی PlayTennis=no و $n_c = 3$ تعداد نمونه‌هایی بود که در آن Wind=strong بود.

با وجود اینکه در بسیاری از موارد این نسبت تخمین خوبی از احتمال به ما می‌دهد، اما زمانی که n_c بسیار کوچک است تخمین ضعیف خواهد بود. برای درک این مشکل، فرض کنید که در حقیقت مقدار احتمال P(Wind=strong|PlayTennis=no) برابر با 08. باشد و در مجموعه‌ی نمونه های ما فقط ۵ نمونه مقدار PlayTennis=no را داشته باشند. با این فرض‌ها، n_c به احتمال زیادی صفر خواهد بود. این حقیقت دو مشکل ایجاد می‌کند. ابتدا اینکه $\frac{n_c}{n}$ تخمینی بایاس دار و دست کم گیرنده از مقدار احتمال خواهد بود. دوم اینکه زمانی که تخمین این احتمال صفر است باعث می‌شود که تمامی نمونه‌هایی که در آن‌ها Wind=strong است جزو دسته‌ی دیگر اطلاق شوند.

برای پرهیز از این مشکل می‌توان از روش بیزی برای تخمین احتمالات استفاده کرد، برای این کار تخمین m [1] را به فرم زیر تعریف می‌کنیم.

تخمین m احتمالات:

[1] m-estimate

$$\frac{n_c + mp}{n + m} \tag{6.22}$$

در این رابطه n_c و n همان مقادیر رابطه‌ی قبلی‌اند و p احتمال اولیه‌ی مقدار تخمینی است. m ثابتی که اندازه‌ی نمونه‌ی معادل[1] نامیده می‌شود. این ثابت مشخص می‌کند که مقدار احتمال به چه میزان به نمونه های آموزشی وابسته باشد. یکی از روش‌های متداول انتخاب p، بدون داشتن هیچ اطلاعات قبلی‌ای، یکنواخت گرفتن تمامی احتمالات اولیه است؛ بدین معنا که اگر ویژگی‌ای k مقدار ممکن دارد خواهیم داشت که $p = \frac{1}{k}$. برای مثال، در تخمین P(Wind=strong|PlayTennis=no) می‌دانیم که ویژگی Wind دو مقدار ممکن دارد، پس با احتمال اولیه‌ی یکنواخت خواهیم داشت که p=.5. توجه دارید که اگر m را صفر انتخاب کنیم، تخمین m معادل همان کسر ساده‌ی $\frac{n_c}{n}$ می‌شود. اگر مقادیر m و n هر دو غیر صفر باشند، حاصل تخمین m میانگین دو مقدار با وزن m خواهد بود. m ثابت اندازه‌ی نمونه‌ی معادل نامیده می‌شود از این رو که رابطه‌ی ۶.۲۲ را می‌توان به صورت ترکیب n مشاهده‌ی واقعی و m مشاهده‌ی مجازی (با احتمال p) در نظر گرفت.

۶.۱۰ یک مثال: یادگیری دسته بندی متون

برای تصور اهمیت کاربردی متد های یادگیری بیز، مسئله‌ی یادگیری‌ای را در نظر بگیرید که در آن نمونه‌ها متنند. برای مثال، شاید بخواهیم مفهوم هدف "مقالات خبری الکترونیکی جالب برای من" یا "صفحاتی از Web که یادگیری ماشین در آن‌ها بحث شده" را یاد بگیریم. در هر دو حالت، اگر یک کامپیوتر بتواند چنین کاری را انجام دهد می‌تواند به جای تعداد بسیاری زیادی از متون وب فقط مربوط‌ترین نتیجه جستجو روی وب را به کاربر ارائه کند.

در اینجا الگوریتمی کلی بر اساس دسته بندی کننده‌ی ساده‌ی بیز برای یادگیری دسته بندی متون ارائه می‌کنیم. جالب است که روش‌های احتمالی مثل آنچه پیش‌تر توضیح دادیم یکی از مؤثرترین الگوریتم‌های شناخته شده برای دسته بندی متون هستند. مثال‌هایی از چنین سیستم‌هایی در (Lewis 1991)، (Lang 1995) و (Joachims 1996) توصیف شده‌اند.

الگوریتم دسته بندی کننده‌ی ساده بیز که توضیح خواهیم داد با تعریف مسئله ای کلی تطابق دارد. فضای نمونه ای X را که شامل تمامی مستندات متنی (تمامی رشته کلمات و علامات با طول دلخواه) است در نظر بگیرید. به ما نمونه های آموزشی تابع هدف مجهول f(x) داده شده است، این تابع مجهول ممکن است هر یک از اعضای V باشد. هدف ما یادگیری از این نمونه های آموزشی برای پیش بینی مقدار هدف متنی جدید است. برای تصور، تابع هدف دسته بندی متون به دو دسته‌ی جذاب و غیر جذاب برای فرد بخصوص است، برای این تابع هدف مقادیر like (جذاب) و dislike (غیر جذاب) برای دسته بندی این دو مجموعه تعریف می‌شود.

دو مشکل اصلی برای کاربرد دسته بندی کننده‌ی ساده‌ی بیز در مسائل دسته بندی متن وجود دارد. اول اینکه با چه روشی یک متن دلخواه را با مقدار ویژگی‌هایی نمایش داد و دوم اینکه احتمالات لازم برای دسته بندی کننده‌ی ساده‌ی بیز را با چه روشی تخمین زد.

روش ما در نمایش متن دلخواه به طرز مشکل سازی ساده است: با داشتن یک متن، مثل همین پاراگراف، باید یک ویژگی برای هر مکان کلمه در متن تعریف کنیم و مقدار ویژگی‌ها را هم کلمات آن مکان‌ها در نظر بگیریم. بنابراین این پاراگراف ۹۷ مقدار ویژگی خواهد داشت که متناسب

[1] equivalent sample size

با ۹۷ کلمه‌ی این پاراگراف است. مقدار ویژگی اول کلمه‌ی "روش" و مقدار ویژگی دوم کلمه‌ی "ما" خواهد بود و ... توجه دارید که با این روش متون بلند تعداد بیشتری ویژگی خواهند داشت. همان طور که بعداً نیز خواهیم دید، این تفاوت هیچ مشکلی ایجاد نمی‌کند.

با این نمایش برای متون، حال می‌توانیم دسته بندی کننده‌ی ساده‌ی بیز را به مسئله اعمال کنیم. بیایید به خاطر حفظ سادگی، فرض کنیم که ۷۰۰ متن آموزشی که فردی dislike دسته بندی کرده به همراه ۳۰۰ متن دیگر که like دسته بندی شده در اختیار است. حال متن جدیدی در اختیار یادگیر قرار گرفته و از وی دسته بندی این متن سؤال می‌شود. دوباره به خاطر سادگی، بیایید فرض کنیم که متن جدید پاراگراف قبلی باشد. در چنین شرایطی، اگر رابطه‌ی ۶.۲۰ را برای دسته بندی مقدار دهی کنیم خواهیم داشت که،

$$v_{NB} = \arg \max_{v_j \in \{like, dislike\}} P(v_j) \prod_{i=1}^{97} P(a_i | v_j)$$

$$\arg \max_{v_j \in \{like, dislike\}} P(v_j) P\left(a_1 = \text{"روش"} \middle| v_j\right) P\left(a_2 = \text{"ما"} \middle| v_j\right)$$

$$\dots P\left(a_{97} = \text{"نمیکند"} \middle| v_j\right)$$

به طور خلاصه، دسته بندی ساده‌ی بیز v_{NB} دسته بندی‌ای است که احتمال مشاهده‌ی کلماتی را که واقعاً در متن بوده‌اند را با توجه به فرض مستقل بودن ساده‌ی بیز ماکزیمم می‌کند. فرض مستقل بودن $P(a_1, \dots a_{97} | v_j) = \prod_1^{97} P(a_i | v_j)$ در این تعریف مسئله فرض می‌کند که احتمال هر کلمه با داشتن دسته بندی متن v_j، برای هر مکان در متن مستقل از دیگر کلمات دیگر مکان‌هاست. توجه می‌کنید که این فرض به وضوح غلت است. برای مثال، در متون ممکن است احتمال آمدن کلمه‌ی "ماشین" بعد از کلمه‌ی "یادگیری" بسیار بیشتر از دیگر کلمات باشد. با وجود این نقص مشهود فرض مستقل بودن، انتخاب دیگری جز این نداریم، زیرا که بدون این شرط تعداد جملات احتمالی‌ای که باید محاسبه شوند به شدت زیاد می‌شوند. خوشبختانه در یادگیر ساده‌ی بیز در بسیاری از موارد در مسائل دسته بندی متون بر خلاف غلت بودن فرض استقلال نتایج خوبی بدست می‌آید. (Domingos and Pazzani 1996) بررسی جالبی از این پدیده‌ی تصادفی ارائه می‌کند.

برای محاسبه‌ی v_{NB} با استفاده از رابطه‌ی بالا، نیاز داریم که احتمال جمله های $P(v_j)$ و $P(a_i = w_k | v_j)$ را محاسبه کنیم (در این مسئله w_k، kامین واژه‌ی واژگان زبان فارسی است). احتمالات جمله‌ی اول را می‌توان به سادگی و با یک نسبت ساده از نمونه های آموزشی محاسبه کرد، (در مثال فعلی P(like)=.3 و P(dislike)=.7). مثل همیشه تخمین دسته بندی احتمالات شرطی (مثل $P(a_1 = \text{"روش"} | dislike)$) مشکل ساز تر خواهد بود زیرا که باید چنین جمله‌ی احتمالی را برای هر ترکیب از متن ممکن کلمات فارسی و تابع هدف محاسبه کنیم. متأسفانه با 50,000 کلمه در واژگان زبان و ۲ حالت مقادیر هدف و ۹۷ کلمه‌ی نمونه‌ی فعلی به تقریباً محاسبه‌ی 2·97·50,000≈10,000,000 حالت نیاز خواهیم داشت.

خوشبختانه، می‌توانیم فرض استدلالی دیگری نیز که تعداد احتمالات را کم بکند به فرض‌ها پیشین اضافه کنیم. در کل، می‌توانیم احتمال بر خورد با کلمه‌ی خاص w_k (مثل "شکلات") را مستقل از مکان حضورش (مثل a_{23} یا a_{95}) در نظر بگیریم. به عبارت رسمی‌تر، ویژگی‌ها از هم مستقلند و توزیع یکسان نیز دارند، با معلوم بودن دسته بندی هدف؛ برای تمامی m،k،j،i داریم $P(a_i = w_k | v_j) = P(a_m = w_k | v_j)$. بنابراین تخمین می‌زنیم که کل مجموعه احتمالات $P(a_1 = w_k | v_j)$ ، $P(a_2 = w_k | v_j)$ ... برابر با یک مقدار مستقل مثبت $P(w_k | v_j)$ باشد، یعنی این مقدار احتمال به مکان کلمه بستگی ندارد. تأثیر این فرض این است که حال فقط نیاز به محاسبه‌ی

2·50,000 جمله‌ی مستقل به فرم $P(w_k|v_j)$ داریم. این مقدار هنوز زیاد است اما دیگر در حد کنترل است. توجه دارید که در شرایطی که داده های آموزشی محدود باشند، مزیت اولیه‌ی این فرض افزایش تعداد نمونه های موجود برای تخمین هر یک از احتمالات و متعاقباً دقت دسته بندی است.

برای کامل کردن طراحی الگوریتم یادگیریمان، هنوز باید متدی برای تخمین جملات احتمالات پیدا کنیم. از تخمین m، کـه در رابطـه‌ی ۶.۲۲ آمد، و احتمالات اولیه‌ی یکنواخت و اندازه‌ی واژگان موجود برای $P(w_k, v_j)$ داریم،

$$\frac{n_k + 1}{n + |Vocabulary|}$$

در این رابطه n کل تعداد کلمات ممکن در نمونه های آموزشی است با مقدار تابع هدف v_j است، n_k تعـداد تکـرار کلمـه‌ی w_k در میـان n کلمه‌ی ممکن است و $|Vocabulary|$ نیز تعداد خالص کل کلمات (و دیگر نشانه‌های) موجود در نمونه های آموزشی است.

به طور خلاصه اینکه الگوریتم نهایی از دسته بندی کننده‌ی ساده‌ی بیز به همراه فرض استقلال کلمات از مکانشان استفاده مـی‌کنـد. الگـوریتم نهایی در جدول ۶.۲ آورده شده است. توجه می‌کنید که این الگوریتم به نسـبت سـاده اسـت. در طـول یـادگیری، زیـر روال Learn-Naive-bayes-text تمامی متون آموزشی را برای استخراج تمام کلمات و نشانه های موجود در متون بررسی می‌کند و تعداد تکرارشان را در دسـته بندی‌های مختلف تابع می‌شمرد تا تخمین‌های لازم را بدست آورد. سپس، برای یک مـتن جدیـد (کـه لازم اسـت دسـته بنـدی شـود) فرآینـد Classify-naive-bayes-text با توجه به رابطه‌ی ۶.۲۰ از این تخمین احتمالات برای محاسبه‌ی v_{NB} استفاده می‌کند. توجه داریـد کـه کلمه که در متن جدید ظاهر شده که در متون قبلی نبوده‌اند توسط Classify-naive-bayes-text نادیده گرفته می‌شود. کد و مجموعه‌ی داده های آموزشی در آدرس http://www.cs.cmu.edu/~tom/book.html موجود است.

۶.۱۰.۱ نتیجه های تجربی

الگوریتم جدول ۶.۲ به چه میزان کارایی دارد؟ در یک آزمایش (Joachims 1996)، الگوریتم بسیار مشابهی برای دسته بندی مقالات خبری یوزنت[1] بکار رفت. دسته بندی مقاله در این مثال اسم گروه خبری مقاله در یوزنت بود. الگوریتمی که هر مقاله را پـس از دسـته بنـدی در جـای اصلی خود قرار می‌دهد. در این آزمایش ۲۰ گروه خبری الکترونیکی در نظر گرفته شد (که در جدول ۶.۳ نیز آمده‌اند)، سپس 1,000 مقاله از هر گروه خبری جمع شد تا تعداد نمونه‌ها به 20,000 برسد. الگوریتم ساده‌ی بیز دو سوم از این 20,000 متن به عنـوان نمونـه هـای آموزشـی آموزش داده شد و سپس کارایی الگوریتم برای یک سوم باقیمانده ارزیابی شد. از ۲۰ گروه خبری ممکن، حداکثر مقدار دسته بندی درست اتفاقی ۵٪ خواهد بود، اما دقت دسته بندی الگوریتم ۸۹٪ اندازه گیری شد. الگوریتم به کار رفته در این آزمایش فقط یک تفاوت کوچـک بـا الگـوریتم جدول ۶.۲ داشت، یک زیر مجموعه از کلمات متون به عنوان واژگان[2] در نظر گرفته شده بود. به عبـارت دقیـق‌تـر، ۱۰۰ کلمـه‌ی پرکـاربرد تـر واژگان در آن در نظر گرفته نشده بود (کلماتی مثل "این")، و همچنین تمامی کلماتی که کمتر از ۳ بار ظاهر شده بودنـد نادیـده گرفتـه شـدند. واژگان بدست آمده بدین ترتیب تقریباً 38,500 کلمه داشت.

[1] use net
[2] Vocabulary

نتایج چشم گیر دیگری نیز توسط دیگر روش‌های یادگیری آماری متون بدست آمده است. برای مثال، (Lang 1995) نسخه ای دیگر از الگوریتم ساده‌ی بیز را توصیف کرده و آن‌را در یادگیری مفهوم هدف "مقالات یوزنتی که من به آن‌ها علاقه دارم" به کار می‌برد. وی سیستم NewsWeeder را معرفی می‌کند، برنامه ای که به کاربران اجازه می‌دهد تا متون را بعد از خواند ارزیابی[1] کنند. سیستم NewsWeeder از این ارزیابی‌ها به عنوان نمونه های آموزشی برای پیش بینی اینکه مقاله ای برای کاربر جالب است یا خیر استفاده می‌کند، پس برنامه می‌تواند مقالاتی که پیش بینی می‌کند کاربر به خواندن آن‌ها علاقه دارد را به وی پیشنهاد کند. (Lang 1995) آزمایشی را گزارش می‌کند که در آن NewsWeeder از اطلاعات یاد گرفته خود بر اساس علاقه‌ی کاربر، مقاله ای که بالاترین مقدار پیش بینی ارزیابی را دارد به کاربر ارائه می‌دهد. با ذخیره‌ی ۱۰٪ اول این مقالات اتوماتیک ارزیابی شده، برنامه مجموعه ای از مقالات خواهد داشت که نسبت به مجموعه‌ی کل مقالات سه تا چهار برابر برای کاربر جالب‌ترند. برای مثال، برای یک کاربر نسبت مقالاتی که "جالب[2]" دسته بندی می‌کند در کل ۱۶٪ است اما در میان این مقالات ۵۹٪ توصیه‌ی NewsWeeder بوده.

تعداد زیاد روش‌های غیر بیزی آماری برای یادگیری متون متداولند، بسیاری از این روش‌ها بر اساس معیارهای مشابه استخراج اطلاعات هستند. (Rocchio 1971; Salton). الگوریتم‌های یادگیری متون دیگر در (Hearst and Hirsh 1996) آورده شده است.

٦.١١ شبکه های باور بیزی

همان طور که در دو قسمت قبلی نیز گفته شد، دسته بندی کننده‌ی ساده‌ی بیز از فرض اینکه احتمالات شرطی $a_1 \dots a_2$ با داشتن مقدار تابع هدف v مستقلند استفاده‌ی شدیدی می‌کند. این فرض به طور قابل توجهی میزان پیچیدگی یادگیری تابع هدف را کاهش می‌دهد. با این فرض، دسته بندی کننده‌ی ساده‌ی بیز دسته بندی بهینه‌ی بیز را خروجی می‌دهد. با این وجود، در بسیاری از موارد این شرط مستقل بودن به شدت محدود کننده است.

شبکه های باور بیزی[3] توزیع احتمالات حاکم بر مجموعه‌ی متغیرهایی که با دسته ای از فرض استقلال احتمالات شرطی مشخص می‌شوند را توصیف می‌کنند. بر خلاف دسته بندی کننده‌ی ساده‌ی بیز که فرض می‌کرد تمامی متغیرهای به طور شرطی با معلوم بودن فرضیه‌ی h مستقلند، شبکه های باور بیزی فرض‌های استقلال احتمالات را در زیر مجموعه های متغیرها درست می‌دانند. بنابراین، شبکه های باور بیزی، روشی میانی که شرطی آزادتر از فرض مستقل بودن تمامی متغیر های دسته بندی کننده‌ی ساده‌ی بیز و محدود کننده تر از پرهیز از هر گونه شرط استقلال است، ارائه می‌کنند. شبکه های باور بیز یکی از موضوعات مورد توجه تحقیقات فعلی هستند، و دامنه‌ی وسیعی از الگوریتم‌ها برای یادگیری و استنتاج از آن‌ها ارائه شده است. در این بخش مفاهیم کلیدی و نحوه‌ی نمایش شبکه های باور بیزی را معرفی خواهیم کرد. اطلاعات دقیق‌تر در این زمینه (Pearl 1988) و (Russell and Norving 1995) و (Heckerman 1995) و (Jensen 1996) آمده است.

در کل، یک شبکه‌ی باور بیز توزیع‌های احتمال دسته ای از متغیرها را توصیف می‌کند. مجموعه‌ی دلخواهی از متغیر های تصادفی $Y_1 \dots Y_n$ را در نظر بگیرید که هر Y_i می‌تواند هر یک از مقادیر مجموعه‌ی $V(Y_i)$ را داشته باشد. فضای توأم[4] را مجموعه‌ی متغیر های Y که از ضرب

[1] rate
[2] interesting
[3] bayesian belief networks
[4] joint space

خارجی $V(Y_1) \times V(Y_2) \dots V(Y_n)$ بدست می‌آید تعریف می‌کنیم. به عبارت دیگر، هر عضو فضای توأم متناسب با یکی از مقادیر ممکن متغیر های $< Y_1 \dots Y_n >$ است. توزیع احتمال این فضای توأم، توزیع احتمال توأم[1] نامیده می‌شود. توزیع احتمال توأم، احتمال مشاهده‌ی هر یک از نمونه های $< Y_1 \dots Y_n >$ را مشخص می‌کند. یک شبکه‌ی باور بیز توزیع احتمال توأم یک مجموعه از متغیرها را توصیف می‌کند.

۶.۱۱.۱ شرط استقلال

بیایید بحثمان درباره‌ی شبکه های باور بیزی را با تعریف دقیق مفهوم استقلال آغاز کنیم. فرض کنید X، Y و Z سه متغیر تصادفی گسسته مقدار باشند، زمانی می‌گوییم که X از Y به شرط Z مستقل است که توزیع احتمال حاکم بر X با فرض داشتن مقدار Z مستقل از مقدار Y باشد؛ به عبارت دیگر،

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

در این رابطه $x_i \in V(X)$، $y_i \in V(Y)$ و $z_i \in V(Z)$ است. معمولاً عبارت بالا را به طور خلاصه به فرم P(X|Y,Z)=P(X|Z) می‌نویسیم. تعریف استقلال شرطی را می‌توان برای مجموعه ای از متغیرها تعمیم داد. می‌گوییم که مجموعه متغیر های $X_1 \dots X_l$ مستقل از مجموعه متغیر های $Y_1 \dots Y_m$ به شرط متغیرهای $Z_1 \dots Z_l$ هستند اگر

$$P(X_1 \dots X_l | Y_1 \dots Y_m, Z_1 \dots Z_l) = P(X_1 \dots X_l | Z_1 \dots Z_l)$$

به رابطه‌ی این تعریف و تعریفمان از استقلال شرطی در دسته بندی کننده‌ی ساده‌ی بیز توجه کنید. دسته بندی کننده‌ی ساده بیز به طور شرطی مستقل بودن ویژگی A_1 از ویژگی A_2 را تعریف می‌کند. این تعریف به دسته بندی کننده‌ی ساده‌ی بیز اجازه می‌دهد که مقدار $P(A_1, A_2 | V)$ را که در رابطه‌ی ۶.۲۰ آمده با استفاده از رابطه‌ی زیر محاسبه کند،

$$P(A_1, A_2 | V) = P(A_1 | A_2, V) P(A_2 | V) \qquad (6.23)$$

$$= P(A_1 | V) P(A_2 | V) \qquad (6.24)$$

رابطه‌ی ۶.۲۳ فقط فرم کلی حاصل از قانون احتمال جدول ۶.۱ است. رابطه‌ی ۲.۲۴ نیز از آن نتیجه شده است، زیرا که اگر A_1 با معلوم بودن V از A_2 مستقل باشد، پس طبق تعریف مستقل شرطی خواهیم داشت که، $P(A_1 | A_2, V) = P(A_1 | V)$.

۶.۱۱.۲ نمایش

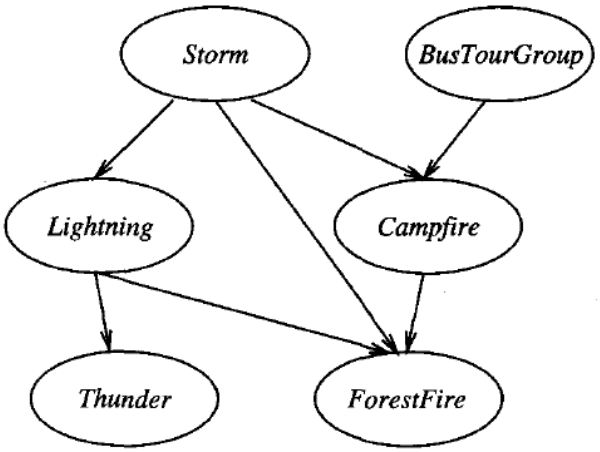
یک شبکه‌ی باور بیزی (که معمولاً به اختصار شبکه بیزی نامیده می‌شود) با توزیع احتمالات توأم مجموعه ای از متغیرها نمایش داده می‌شود. برای مثال، شبکه بیزی شکل ۶.۳ توزیع احتمال توأم متغیر های منطقی Storm، Lightning، Thunder، ForestFire، Campfire و BusTourGroup را نشان می‌دهد. در کل، یک شبکه‌ی بیزی توزیع احتمال توأم را با استفاده از مشخص کردن مجموعه ای از فرض‌های استقلال شرطی (که با یک گراف بدون دور نمایش داده می‌شود) و مجموعه های از احتمالات شرطی هر کدام مشخص می‌کند. هر متغیر فضای توأم با یک گره در شبکه بیزی نشان داده می‌شود. برای هر متغیر دو نوع اطلاعات ذکر می‌شود، اول اینکه با فرض داشتن والدین

[1] Joint probability distribution

(در گراف) متغیر از متغیر های غیر زیرینش مستقل شرطی است. زمانی می‌گوییم که X زیرینی[1] برای Y است که مسیری مستقیم از Y به X باشد. دوم اینکه جدولی از احتمالات شرطی برای هر متغیر داده می‌شود که توزیع احتمال را برای مقدار متغیر های بالایی[2] مشخص می‌کند. احتمال توأم هر یک از مقادیر $< y_1, \dots, y_n >$ که مقداری از $< Y_1 \dots Y_n >$ است را می‌توان با استفاده از رابطه‌ی زیر محاسبه کرد،

$$P(y_1, \dots, y_n) = \prod_{i=1}^{n} P(y_i | Parnets(Y_i))$$

در این رابطه $Parents(Y_i)$ نماد مجموعه ای از بالایی‌های مستقیم Y_i در شبکه است. توجه داشته باشید که $P(y_i|Parents(Y_i))$ دقیقاً مقدار های ذخیره شده در جدول احتمالات شرطی مربوط به گره Y_i است.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

شکل ۶.۳ یک شبکه‌ی باور بیزی.

شبکه‌ی سمت چپ مجموعه ای از فرض‌های استقلال شروط را نشان می‌دهد. در کل، هر گره مستقل شرطی است از شروط غیر زیرینش[3] با معلوم بودن شروط والدش مستقل است. برای هر گره جدول مقادیر شرطی‌ای وجود دارد که توزیع احتمال شرطی متغیرها را با معلوم بودن شروط والدینش در گراف مشخص می‌کند. جدول احتمال شرطی مربوط به گره Campfire که به طور خلاصه با C نمایش داده شده در سمت راست شکل آورده شده است، گره های Storm و BusTourGroup نیز به ترتیب به طور خلاصه با S و B نمایش داده شده‌اند.

برای تصور، شبکه‌ی بیزی شکل ۶.۳ توزیع احتمال توأم را برای متغیر های منطقی Storm، Lightning، Thunder، ForestFire، Campfire و BusTourGroup نشان می‌دهد. گره‌ها و یال‌های[4] شبکه نشان می‌دهد که Campfire با معلوم بودن والدینش، Storm و BusTourGroup، از Lightning و Thunder مستقل شرطی است. این بدین معناست که زمانی که مقدار Storm و BusTourGroup مشخص است متغیر های Lightning و Thunder هیچ اطلاعات اضافه ای در مورد متغیر Campfire به ما نخواهند داد. برای مثال، سه داده‌ی اول سمت چپ جدول نشان می‌دهند که،

$$P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4$$

---

[1] descendant
[2] predecessors
[3] nondescendants
[4] arc

توجه دارید که این جدول فقط مقادیر احتمال شرطی Campfire را با معلوم بودن مقادیر متغیر های Storm و BusTourGroup می‌دهد. مجموعه‌ی موضعی جدول احتمالات شرطی برای تمامی متغیرها و مجموعه ای از فرض‌های استقلال شرطی که شبکه می‌گذارد، با هم توزیع احتمال شبکه روی کل فضای توأم را مشخص می‌کنند.

یکی از ویژگی‌های جذاب شبکه های باور بیزی این است که اجازه‌ی نمایش ساده‌ی اطلاعات علی[1]، مثل این حقیقت که رعد و برق (lightning) باعث طوفان (Thunder) می‌شود، را به ما می‌دهد. در واژگان استقلال شرطی، این حقیقت را در شبکه با اینکه احتمال Thunder با معلوم بودن مقدار Lightning از بقیه‌ی متغیرها مستقل است نشان می‌دهیم. توجه داشته باشید که این فرض استقلال شرطی با یال‌های شبکه‌ی بیزی شکل ۶.۳ نشان داده شده است.

٦.١١.٣ استنتاج[2]

ممکن است بخواهیم از شبکه های بیزی برای استنتاج مقدار چند متغیر (مثل ForestFire) با داشتن چند متغیر دیگر استفاده کنیم. البته، با دانستن اینکه کار ما با متغیر های تصادفی است، در کل نیز نسبت دادن یک مقدار به متغیر هدف صحیح نخواهد بود. در اینجا ما بیشتر به استنتاج توزیع احتمال متغیر هدف علاقه داریم، توزیع احتمالی که مشخص می‌کند مقدار هدف با معلوم بودن مقادیر مفروض با چه احتمالی کدام مقدارش را می‌تواند داشته باشد. اگر مقادیر تمامی متغیر های دیگر شبکه معلوم باشند مرحله‌ی استنتاج خیلی ساده خواهد شد. در حالت کلی‌تر ممکن است بخواهیم توزیع احتمال یک متغیر را با داشتن فقط زیر مجموعه ای از تمامی متغیرها (مثل ForestFire) استنتاج کنیم (ممکن است دو مقدار Thunder و BusTourGroup تنها مقادیر مشاهده شده‌ی ما باشند). در کل، یک شبکه‌ی بیزی را می‌توان برای محاسبه‌ی توزیع احتمال هر زیرمجموعه ای از متغیر های شبکه با استفاده از معلوم بودن مقادیر هر زیر مجموعه‌ی دیگری از متغیر های شبکه استفاده کرد.

استنتاج دقیق احتمالات در حالت کلی برای هر شبکه‌ی بیز دلخواه NP-hard است (Cooper 1990). متد های عددی‌ای نیز برای استنتاج احتمالات در شبکه های بیزی، شامل متدهای استنتاج دقیق[3] و متد های تخمین استنتاج که دقت را فدای بازده می‌کنند ارائه شده‌اند. برای مثال، متد های Monte Carlo راه حل‌های تخمینی را با استفاده از نمونه برداری تصادفی از توزیع احتمال متغیر های مورد نظر را پیشنهاد می‌کنند (Pradham and Dagum 1996). در تئوری، حتی تخمین استنتاجی احتمالات شبکه‌ی بیزی را می‌توان NP-hard دانست (Dagum and Luby 1993). خوشبختانه در عمل، متد های تخمینی در بسیاری از موارد مفید از آب در آمده‌اند. بحث متد های استنتاج شبکه های بیزی در (Russell and Norvig 1995) و (Jensen 1996) آمده است.

٦.١١.٤ یادگیری شبکه های باور بیزی

آیا می‌توانیم الگوریتمی موثر برای یادگیری شبکه های باور بیزی از داده های آموزشی پیدا کنیم؟ این سؤال، زمینه‌ی مورد توجه اکثر تحقیقات فعلی است. تعریف مسئله های مختلفی را می‌توان برای این سؤال در نظر گرفت. ابتدا اینکه ساختار شبکه ممکن است دقیق مشخص باشد، یا ممکن است ساختار شبکه با توجه به داده های آموزشی انتخاب شود. دوم اینکه تمامی متغیر های شبکه ممکن است در هر نمونه‌ی آموزشی مشهود و معلوم باشد یا بلعکس بعضی ممکن است بعضی متغیرها غیر قابل مشاهده باشند.

---

[1] causal knowledge
[2] inference
[3] exact inference

در حالتی که ساختار شبکه دقیق مشخص است، و فقط بعضی از مقادیر متغیر های آن قابل مشاهده‌اند، مسئله‌ی یادگیری بسیار سخت‌تر خواهد بود. این مسئله به نحوی مشابه یادگیری وزن‌های واحد های پنهان شبکه های عصبی است، جایی که ورودی و خروجی شبکه مشخص است اما اطلاعاتی در مورد لایه‌ی پنهان شبکه در نمونه های آموزشی نیست. در واقع، (Russell 1995) فرایندی مشابه با شیب صعود ارائه داده است که احتمالات شرطی جدول را یاد می‌گیرد. این فرایند شیب صعود در فضایی از فرضیه‌ها که متناسب با مجموعه ای از تمامی حالت‌های ممکن احتمالات شرطی است برای یافتن مقادیر جدول احتمالات شرطی جستجو می‌کند. تابع ای که در طول شیب صعود ماکزیمم می‌شود احتمال P(D|h) داده های آموزشی D است با معلوم بودن فرضیه‌ی h است. طبق تعریف، این جستجو معادل جستجو برای محتمل‌ترین فرضیه برای مقادیر جدول است.

۶.۱۱.۵ آموزش شیب صعود برای شبکه های بیزی

قانون شیب صعودی که (Russell 1995) معرفی کرد، با حرکت به سمت افزایش ln P(D|h) مقدار P(D|h) را با توجه به مقادیر جدول احتمالات شرطی شبکه‌ی بیز ماکزیمم می‌کند. فرض کنید w_{ijk} نماد تک داده‌ی ای از جدول‌های احتمالات شرطی شبکه باشد. در کل فرض کنید که w_{ijk} نماد احتمال شرطی اینکه متغیر شبکه‌ی Y_i مقدار y_{ij} را داشته باشد با معلوم بودن اینکه متغیر U_i مقدار u_{ij} را داشته باشد. برای مثال، اگر w_{ijk} داده‌ی گوشه‌ی بالا و سمت راست جدول احتمالات شرطی ۶.۳ باشد و Y_i نیز متغیر Campfire باشد و U_i نیز والدین آن یعنی <Storm,BusTourGroup> و $y_{ij} = True$ و $u_{ik} =< False, False >$ است. گرادیان ln P(D|h) که با $\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$ برای هر w_{ijk} نشان داده می‌شود را همان طور که بعداً نیز نشان خواهیم داد می‌توان به صورت زیر محاسبه کرد،

$$\frac{\partial lnP(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \qquad (6.25)$$

برای مثال، برای محاسبه‌ی هر یک از مقادیر مشتق ln P(D|h) نسبت به داده‌ی گوشه‌ی بالا و راست جدول ۶.۳ باید مقدار P(Campfire=True,Storm=False,BusTourGroup=False|d) را برای هر یک از نمونه های آموزشی d در D محاسبه کنیم. زمانی که این متغیر های برای نمونه ای مثل d مجهول است، لازم است که این احتمال را با استنتاج از متغیرهای دیگر آموزشی موجود d محاسبه کنیم. در واقع، این کمیت‌ها به راحتی از محاسبات استنتاجی انجام شده در اکثر شبکه های بیزی استخراجی می‌شود، بنابراین یادگیری را می‌توان با هزینه ای کمی بیشتر، که از شبکه‌ی بیزی برای استنتاج و مدارک جدید متعاقباً بدست می‌آید.

در زیر از رابطه‌ی ۶.۲۵ که (Russell 1995) معرفی کرده بدست می‌آوریم. ادامه‌ی این قسمت را می‌توانید بدون از دست دادن پیوستگی قسمت‌ها در اولین خواند کتاب نخوانید. برای ساده سازی نماد، در این مشتق گیری از نماد $P_h(d)$ برای نمایش P(D|h) استفاده خواهیم کرد. می‌خواهیم گرادیان این تابع را بیابیم پس باید رابطه‌ی $\frac{\partial lnP_h(D)}{\partial w_{ijk}}$ را به ازای تمامی مقادیر i و j و k محاسبه کنیم. با فرض اینکه نمونه های آموزشی d در مجموعه‌ی داده های D مستقل باشند، می‌توان نوشت،

$$\frac{\partial lnP(D|h)}{\partial w_{ijk}} = \frac{\partial}{\partial w_{ijk}} \ln \prod_{d \in D} P_h(d)$$

$$= \sum_{d \in D} \frac{\partial \ln P_h(d)}{\partial w_{ijk}}$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial P_h(d)}{\partial w_{ijk}}$$

مرحله‌ی آخر از رابطه‌ی $\frac{\partial \ln f(x)}{\partial x} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}$ نتیجه گیری شده است. حال می‌توان مقادیر متغیر هـای Y_i و $U_i = Parents(Y_i)$ را با استفاده از جمع روی مقادیر $y_{ij'}$ و $u_{ik'}$ معرفی کرد.

$$\frac{\partial lnP(D|h)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}, u_{ik'})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}|u_{ik'}) P_h(u_{ik'})$$

مرحله‌ی آخر از قانون احتمال جدول ۶.۱ نتیجه گیری شده است. حال جمع سمت راستی رابطه‌ی بالا را در نظـر بگیریـد، بـا توجـه بـه تعریـف $w_{ijk} \equiv P_h(y_{ij}|u_{ik})$ خواهیم داشت که تمامی جملات جز جمله‌ی j'=j و i'=i صفر خواهند بود پس داریم،

$$\frac{\partial lnP(D|h)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})$$

با استفاده از قضیه‌ی بیز برای مقدار $P_h(d|y_{ij}, u_{ik})$ داریم،

$$\frac{\partial lnP(D|h)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}} \qquad (6.26)$$

بدین صورت مشتق رابطه‌ی ۶.۲۵ محاسبه می‌شود. قبل از رفتن به سراغ قانون فرایند شیب صعود باید در نظر گرفت که باید پس از تغییر مقادیر w_{ijk} آن‌ها همچنان در بازه‌ی [0,1] باقی بمانند تا احتمالات معتبری باشند. از طرف دیگر باید مقدار $\sum_j w_{ijk}$ برای تمامی مقادیر i و k، ۱ باقی بماند. این شروط را می‌توان با تغییر دو مرحله ای وزن‌ها اعمال کرد، ابتدا هر w_{ijk} را با توجه به شیب صعود تغییر می دهیم،

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

در این رابطه η ثابت کوچکی به نام ضریب یادگیری است. در مرحله‌ی دوم وزن‌ها را نرمالیزه می‌کنیم تا در شروط بالا صدق کنند. همان طور که Russell نیز توضیح داده است این فرایند به محتمل‌ترین فرضیه‌ی نسبی برای احتمالات شرطی در شبکه بیز میل خواهد کرد.

مثل دیگر روش‌های بر پایه‌ی شیب صعود، این الگوریتم نیز فقط تضمین می‌کند که راه حل بهینه‌ی موضعی پیدا کند. جایگزین دیگر موجود برای شیب صعود الگوریتم EM است که در قسمت ۶.۱۲ توضیح داده می‌شود، این الگوریتم نیز راه حلی بهینه موضعی پیدا خواهد کرد.

٦.١١.٦ یادگیری ساختار شبکه‌ی بیزی

یادگیری شبکه های بیزی هنگامی که ساختار شبکه به دقت معلوم نیست نیز پیچیده است. (Cooper and Herskovits 1992) روشی Bayesian scoring metric برای انتخاب میان شبکه های مختلف ارائه می‌کنند. آن‌ها همچنین جستجویی ابتکاری به نام الگوریتم K2 برای یادگیری ساختار شبکه در شرایطی که داده‌ها به طور کامل قابل مشاهده‌اند ارائه می‌کنند. مشابه اکثر الگوریتم‌های یادگیری ساختار شبکه‌ی بیز، K2 نیز از جستجویی حریصانه که پیچیدگی فرضیه را فدای دقت روی داده های آموزشی می‌کند استفاده می‌کند. در آزمایشی به K2 مجموعه ای از 3,000 نمونه‌ی آموزشی تصادفی از شبکه بیزی‌ای معلومی با ۳۷ گره و ۴۶ یال داده شد. این شبکه‌ی خاص مشکلات بیهوشی را در یک اتاق جراحی بیمارستان توصیف می‌کرد. علاوه بر این داده‌ها، به برنامه ترتیبی اولیه ای از ۳۷ متغیری که سازگار با قسمتی از ترتیب وابستگی متغیرها در شبکه‌ی واقعی بود نیز داده شد. این برنامه در تشخیص شبکه‌ی بیزی درست تقریباً موفق شد، این شبکه یالی اضافه و یالی دیگر کمتر از شبکه‌ی اصلی داشت.

روش‌های مبتنی بر قیود[1] نیز در یادگیری ساختار شبکه های بیزی نیز پیشنهاد شده است (Sprites et al. 1993). این روش‌ها روابط استقلال و وابستگی را از داده‌ها استنتاج کرده و از آن‌ها برای ساخت شبکه های بیزی استفاده می‌کنند. بررسی مربوطه‌ی روش‌های فعلی یادگیری ساختار شبکه های بیزی در (Heckerman 1995) و (Buntine 1994) آورده شده است.

٦.١٢ الگوریتم EM

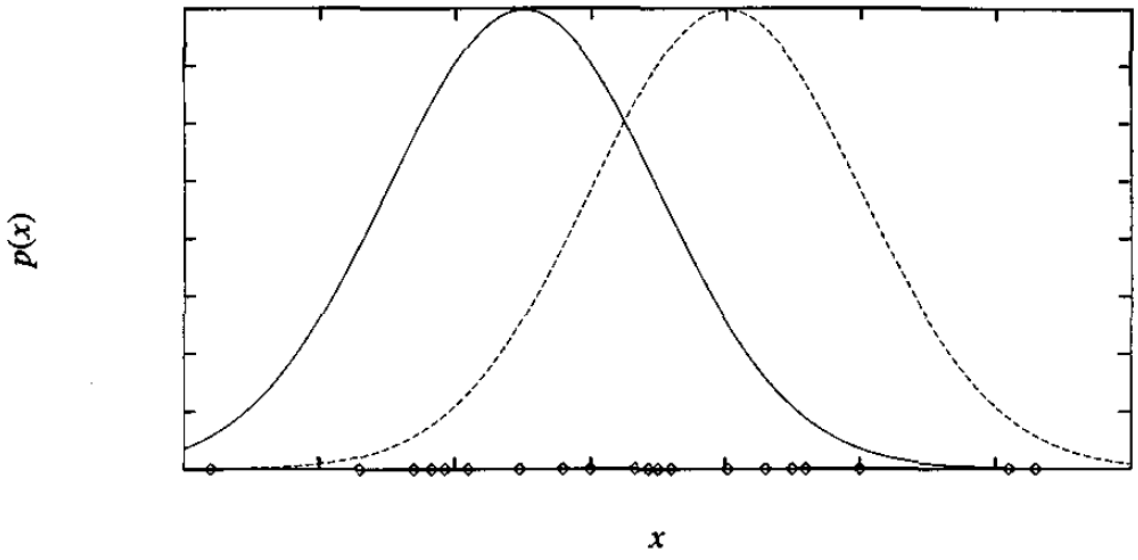
در بسیاری از تعریف مسئله های کاربردی، فقط یک زیر مجموعه از ویژگی‌های نمونه‌ها قابل مشاهده است. برای مثال، در یادگیری یا استفاده‌ی شبکه‌ی باور بیزی‌ای که در جدول ۶.۳ آورده شد، ممکن است فقط داده های نظیر یک زیر مجموعه از متغیر های شبکه مثل زیر مجموعه‌ی Storm, Lightning, Thunder, ForestFire, Campfire, BusTourGroup را داشته باشیم. روش‌های بسیاری برای کنترل این مشکل پیشنهاد شده است، همان طور که در فصل ۳ نیز دیدید، اگر بعضی متغیرها در بعضی موارد غیر قابل مشاهده و در

[1] constraint-based

بعضی موارد قابل مشاهده باشند، می‌توان از نمونه های آموزشی‌ای که این مقدار را دارند برای پیش بینی این ویژگی در دیگر نمونه‌ها استفاده کرد. در این بخش به الگوریتم EM که در یادگیری با وجود ویژگی‌های مجهول کاربرد زیاد دارد می‌پردازیم. از الگوریتم EM می‌توان حتی برای متغیرهایی که هیچ وقت به طور مستقیم قابل مشاهده نیستند نیز استفاده کرد، اما لازم است که فرم کلی توزیع احتمال حاکم بر این متغیرها معلوم باشد. الگوریتم EM برای آموزش شبکه های باور بیزی (برای اطلاعات بیشتر به Heckerman 1995 رجوع کنید) و شبکه های توابع شعاعی[1] که در قسمت ۸.۴ توضیح داده شد به کار می‌روند. الگوریتم EM پایه‌ی بسیاری از الگوریتم‌های خوشه یابی[2] (Cheeseman 1988) و همچنین پایه‌ی الگوریتم‌های پرکاربرد Baum-Welch forward-backward برای یادگیری مدل‌های نیمه مشهود مارکوو[3] است (Rabiner 1989).

۶.۱۲.۱ تخمین میانگین k توزیع نرمال

راحت‌ترین راه معرفی الگوریتم EM از طریق یک مثال است. مسئله ای را در نظر بگیرید که در آن داده های آموزشی D مجموعه ای از نمونه‌هایی است که از طریق توزیعی که ترکیب k توزیع نرمال[4] است بدست آمده‌اند. این تعریف مسئله برای k=2 در شکل ۶.۴ آمده است، در این شکل نمونه‌ها نقاط روی محور x هستند. هر نمونه از فرایندی دو مرحله ای بدست می‌آید. ابتدا به تصادف یکی از k توزیع نرمال انتخاب می‌شود. سپس بر اساس آن توزیع نرمال نمونه‌ی x_i ایجاد می‌گردد. این فرایند برای ایجاد مجموعه ای از نمونه های آموزشی همان طور که در شکل نشان داده شده است تکرار خواهد شد. برای ساده سازی بحث، حالتی را بررسی می‌کنیم که احتمال تمامی توزیع‌های نرمال در مرحله‌ی اول یکسان است و تمامی توزیع‌های نرمال واریانس مشترک σ^2 دارند. هدف یادگیری پیدا کردن فرضیه ای به شکل $h =< \mu_1, \dots, \mu_k >$ است که میانگین‌های k توضیح احتمال را توصیف کند. در کار یادگیری سعی می‌کنیم تا محتمل‌ترین فرضیه را پیدا کنیم؛ فرضیه ای که p(D|h) را ماکزیمم کند.



[1] Radial basis function network
[2] clustering
[3] Partially Observable Markov Models
[4] Mixed Gaussian distribution

شکل ۶.۴ نمونه های حاصل از ترکیب دو توزیع نرمال با واریانس σ یکسان.
نمونه‌ها با نقاطی روی محور x نشان داده شده‌اند. اگر میانگین توزیع‌های نرمال نامعلوم باشد، الگوریتم EM را می‌توان برای جستجوی محتمل‌ترین مقدار تخمین آن‌ها به کاربرد.

توجه دارید که محاسبه‌ی محتمل‌ترین فرضیه برای میانگین یک توزیع نرمال با داشتن نمونه های $x_1, x_2, \dots, x_m$ فقط حالت خاصی از مسئله ای است که در قسمت ۶.۴ بحث شد، در رابطه‌ی ۶.۶ نشان دادیم که محتمل‌ترین فرضیه، فرضیه ای است که مجموع خطاهای مربعی را برای تمامی m نمونه مینیمم می‌کند. اگر رابطه‌ی ۶.۶ را با توجه به نماد گذاری جدید بازنویسی کنیم، خواهیم داشت،

$$\mu_{ML} = \arg\max_{\mu} \sum_{i=1}^{m} (x_i - \mu)^2 \tag{6.27}$$

در چنین شرایطی مجموع خطا های مربعی با تساوی زیر مینیمم خواهد شد،

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{6.28}$$

با وجود این وجود مسئله‌ی ما درباره‌ی ترکیبی از توابع نرمال است و تشخیص اینکه نمونه‌ها از کدام توزیع بدست آمده‌اند نیز ممکن نیست. بنابراین، با صورت مثالی کلی از مسئله‌هایی که متغیر های پنهان دارند مواجهیم. در مثال شکل ۶.۴، می‌توان توضیح کامل مربوطه‌ی هر نمونه را به شکل سه تایی مرتب $< x_i, z_{i1}, z_{i2} >$ در نظر گرفت، در این سه تایی مرتب x_i مقدار مشاهده شده‌ی iامین نمونه است و دو مقدار z_{i1}, z_{i2} مشخص می‌کند که کدام یک از دو توزیع نرمال برای تولید این نمونه به کار رفته‌اند. در کل، z_{ij} زمانی یک است که نمونه از توزیع نرمال j ام بدست آمده است و در غیر این صورت صفر خواهد بود. در اینجا متغیر x_i قابل مشاهده و متغیر های z_{i1}, z_{i1} متغیر های پنهان هستند. اگر مقادیر متغیر های z_{i1}, z_{i1} قابل مشاهده بودند می‌شد از رابطه‌ی ۶.۲۷ برای پیدا کرد μ_1 و μ_2 استفاده کرد، حال چون این متغیرها قابل مشاهده نیستند از الگوریتم EM استفاده خواهیم کرد.

در این مثال، پیدا کردن k میانگین، الگوریتم EM به تخمین مقادیر z_{ij} با معلوم بودن فرضیه‌ی فعلی $< \mu_1 \dots \mu_k >$ می‌پردازد و سپس مقادیر محتمل‌ترین فرضیه‌ها را با توجه به این مقادیر تصادفی برای متغیر های پنهان دوباره محاسبه می‌کند. ابتدا این مثال را در الگوریتم EM توصیف حل می‌کنیم، الگوریتم EM را در حالت کلی بیان خواهیم کرد.

برای شکل ۶.۴، الگوریتم EM ابتدا مقدار اولیه‌ی فرضیه را به $h =< \mu_1, \mu_2 >$ که در آن μ_1 و μ_2 دو مقدار دلخواه هستند مقدار دهی اولیه می‌کند. سپس فرضیه‌ی h را با تکرار حلقه‌ی دو مرحله ای زیر ارزیابی می‌کند، این حلقه تا زمانی که فرایند به مقدار ثابتی از h همگرا شود حلقه تکرار خواهد شد.

مرحله‌ی ۱: مقدار امید $E[z_{ij}]$ را برای هر متغیر پنهان z_{ij} با فرض درستی فرضیه‌ی $h =< \mu_1, \mu_2 >$ محاسبه کن.

مرحله‌ی ۲: محتمل‌ترین فرضیه‌ی جدید $h' =< \mu_1', \mu_2' >$ را با فرض اینکه تمامی مقادیر z_{ij} مقدار امید $E[z_{ij}]$ که در مرحله‌ی ۱ محاسبه شد است را محاسبه کن. سپس فرضیه‌ی $h =< \mu_1, \mu_2 >$ را با فرضیه‌ی $h' =< \mu_1', \mu_2' >$ جایگزین کن.

بیایید نحوه‌ی پیاده سازی هر یک از مراحل را در عمل بررسی کنیم. مرحله‌ی اول باید مقدار امید هر یک از z_{ij} را محاسبه کند. این مقدار $E[z_{ij}]$ فقط احتمال نمونه‌ی x_i است که از طریق j امین توزیع نرمال ایجاد شده است.

$$E[z_{ij}] = \frac{p(x = x_i|\mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i|\mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}}$$

مرحله‌ی اول با بکار گیری مقادیر فعلی $< \mu_1, \mu_2 >$ و مقدار فعلی x_i نتیجه گیری شده است.

در مرحله‌ی دوم مقداری که برای $E[z_{ij}]$ در مرحله‌ی اول محاسبه شد استفاده می‌شود تا محتمل‌ترین فرضیه $h' =< \mu_1', \mu_2' >$ محاسبه شود. همان طور که بعداً نیز بحث خواهیم کرد، محتمل‌ترین فرضیه در این حالت با رابطه‌ی زیر محاسبه می‌شود،

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}]x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

توجه دارید که این رابطه تشابه بسیاری به رابطه‌ی ۶.۲۸ برای تخمین مقدار μ برای یک توزیع نرمال دارد. در رابطه‌ی جدید فقط میانگین وزن دار μ_j هاست، وزن هر μ_j مقدار $E[z_{ij}]$ که از j امین توزیع نرمال بدست آمده است.

الگوریتم بالا برای تخمین میانگین‌های ترکیب k توزیع نرمال با روش الگوریتم EM است: فرضیه‌ی فعلی برای تخمین متغیر های نا مشهود استفاده می‌شود، سپس مقدار امید این متغیرها برای محاسبه‌ی فرضیه‌ی بهتری به کار می‌رود. می‌توان اثبات کرد که با هر دور اجرای حلقه الگوریتم EM میزان محتمل بودن P(D|h) را بیشتر می‌کند، مگر اینکه آن یک ماکزیمم نسبی باشد. پس الگوریتم EM در انتها به یک ماکزیمم موضعی برای محتمل بودن $< \mu_1, \mu_2 >$ میل خواهد کرد.

۶.۱۲.۲ حالت کلی الگوریتم EM

در بالا الگوریتم EM را برای مسئله‌ی تخمین میانگین‌های ترکیب توزیع احتمال‌های نرمال بیان کردیم. در حالت کلی‌تر، الگوریتم EM را می‌توان در بسیاری از تعریف مسئله‌ها که در آن‌ها بحث از تخمین مجموعه ای از پارامتر های θ که توضیح احتمال حاکم را توصیف می‌کنند با استفاده از قسمتی از داده‌ها که قابل شهود است به کاربرد. در مثال دو میانگین بالا پارامتر های مورد علاقه $\theta =< \mu_1, \mu_2 >$ است، داده های کامل سه تایی مرتب‌های $< x_i, z_{i1}, z_{i2} >$ هستند که فقط x_i قابل مشاهده است. در کل اگر $X = \{x_1, \dots, x_m\}$ داده های مشاهده شده در مجموعه ای از m نمونه‌ی مستقل و $Z = \{z_1, \dots, z_m\}$ نیز داده های غیر قابل مشاهده باشد در این نمونه های آموزشی Y=X∪Z کل داده‌ها خواهد بود. توجه دارید که با Z می‌توان به دید متغیر تصادفی‌ای که توزیع احتمالش به پارامترهای نامعلوم θ و داده های مشاهده شده‌ی X وابسته است نگاه کرد. به طور مشابه، Y نیز متغیری تصادفی است، زیرا که توسط متغیر تصادفی Z تعریف می‌شود. در ادامه‌ی این بخش فرم کلی الگوریتم EM را توضیح خواهیم داد. از h برای نمایش مقادیر مفروض فعلی از پارامتر های θ و از h' برای فرضیه‌ی بازبینی شده‌ی هر حلقه الگوریتم EM استفاده خواهیم کرد.

الگوریتم EM فضای فرضیه‌ی محتمل‌ترین فرضیه‌ها h' را برای پیدا کردن h' ی که مقدار $E[\ln P(Y|h')|h]$ را ماکزیمم کند جستجو می‌کند. این مقدار امید برای تمامی توزیع احتمالات Y که توسط پارامتر های نامعلوم مشخص می‌گردد محاسبه می‌شود. بیایید مفهوم دقیق این مقدار امید را با هم بررسی کنیم. ابتدا اینکه $P(Y|h')$ محتمل بودن داده های کامل Y را با شرط معلوم بودن h' نشان می‌دهد. پس پیدا کردن فرضیه‌ی h' به قسمی که تابعی از این معیار را ماکزیمم کند منطقی خواهد بود. دوم اینکه، ماکزیمم کردن لگاریتم این کمیت، $ln\, P(Y|h')$ نیز $P(Y|h')$ را ماکزیمم می‌کند (همان طور که پیش‌تر نیز گفته بودیم). سوم اینکه ما مقدار امید $E[ln\, P(Y|h')]$ را برای اینکه داده های کامل Y خود متغیری تصادفی است معرفی می‌کنیم. با دانستن اینکه داده های کامل Y ترکیبی از داده های مشاهده شده‌ی X و داده های مشاهده نشده ی Z است، باید میانگین را برای مقادیر ممکن Z های مشاهده نشده با وزن متناسب با احتمالشان محاسبه کنیم. به عبارت دیگر، مقدار امید $E[ln\, P(Y|h')]$ بر روی توزیع احتمالات تصادفی Y محاسبه می‌شود. توزیع Y توسط مقادیر کاملاً معلوم X و توزیع احتمال حاکم بر Z تعیین می‌شود.

توزیع احتمال حاکم بر Y چیست؟ در کل این توزیع را نمی‌دانیم، زیرا که این توزیع با پارامتر های θ که می‌خواهیم تخمین بزنیم تعیین می‌شود. بنابراین، الگوریتم EM از فرضیه‌ی فعلی h به جای پارامتر های واقعی θ برای تخمین توزیع احتمال حاکم بر Y استفاده می‌کند. بیایید تابعی به فرم $Q(h'|h)$ تعریف کنیم که مقدار $E[\ln P(Y|h')|h]$ را به عنوان تابعی از h' با فرض θ=h و داشتن قسمت قابل مشاهده‌ی X از کل داده های Y بیان کند.

$$Q(h'|h) = E[\ln P(Y|h')|h, X]$$

این تابع Q را به فرم $Q(h'|h)$ **می** نویسیم تا نشان دهد که این تابع با این فرض تعریف شده که فرضیه‌ی فعلی h با θ مساوی است. در فرم کلی، الگوریتم EM تا رسیدن به همگرایی دو مرحله‌ی زیر را تکرار می‌کند:

مرحله‌ی ۱: مرحله‌ی تخمین (E): مقدار Q(h′|h) را با استفاده از فرضیه‌ی فعلی h و داده های مشاهده شده‌ی X برای تخمین توزیع احتمال روی Y محاسبه کن.

$$Q(h'|h) = E[\ln p(Y|h')\,|h, X]$$

مرحله‌ی ۲: مرحله‌ی ماکزیمم سازی (M): فرضیه‌ی h را با فرضیه‌ی h′ که مقدار Q را ماکزیمم می‌کند جایگزین کن.

$$h \leftarrow \arg\max_{h'} Q(h'|h)$$

اگر تابع Q پیوسته باشد، الگوریتم EM به نقطه تعادل محتمل‌ترین فرضیه‌ی P(Y|h′) میل خواهد کرد. اگر این تابع محتمل بودن فقط یک ماکزیمم داشته باشد، EM نیز به همان تخمین همان ماکزیمم مطلق برای h′ میل خواهد کرد. در غیر این صورت، این الگوریتم تضمین می‌کند تا به ماکزیممی موضعی میل کند. در چنین شرایطی، EM محدودیت‌های الگوریتم‌های دیگری که از جستجوی شیب نزول استفاده می‌کنند را خواهد داشت، در فصل ۴ توضیح کاملی در مورد این مشکلات و راه حل‌های آن‌ها آورده شده است.

۶.۱۲.۳ اشتقاق الگوریتم k میانگین

برای تصور بهتر کلی الگوریتم EM، بیایید مشتق الگوریتم آورده شده در قسمت ۶.۱۲.۱ برای تخمین میانگین‌های k توزیع نرمال را بررسی کنیم. همان طور که در بالا نیز توضیح داده شد، مسئله‌ی تخمین k میانگین، مسئله‌ی تخمین پارامتر های $\theta =< \mu_1 \dots \mu_k >$ است که

میانگین k توزیع نرمال تعریف می‌شوند. داده های مشاهده شده‌ی $X = \{< x_i >\}$ به ما داده شده‌اند. در این مسئله متغیر های پنهان $Z = \{z_{i1}, \dots, z_{ik}\}$ هستند که مشخص می‌کنند که نمونه با استفاده از کدام توزیع ایجاد شده است.

برای به کار بردن EM ابتدا باید مشتق Q(h|h′) را برای این تخمین k میانگین پیدا کرد. بیایید ابتدا مشتق رابطه‌ی p(Y|h′) را محاسبه کنیم. توجه دارید که احتمال $p(y_i|h')$ برای یک تک نمونه‌ی $y_i =< x_i, z_{i1}, \dots, z_{ik} >$ از داده های کامل را می‌توان به صورت زیر دقیق محاسبه کرد.

$$p(y_i|h') = p(x_i, z_{i1}, \dots, z_{ik}|h') = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}\left(x_i-\mu'_j\right)^2}$$

توجه داشته باشید که از تمامی z_{ij} ها فقط یکی ۱ است و بقیه ۰ هستند. بنابراین این رابطه توزیع احتمال x_i را بر اساس توزیع نرمال انتخابی نشان می‌دهد. با داشتن احتمال تک نمونه، $p(y_i|h')$ ، لگاریتم احتمال P(Y|h′) برای تمامی m نمونه در داده‌ها به صورت زیر خواهد بود،

$$\ln P(Y|h') = ln \prod_{i=1}^{m} p(y_i|h')$$

$$= \sum_{i=1}^{m} \ln p(y_i|h')$$

$$= \sum_{i=1}^{m} \left(ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} z_{ij}\left(x_i - \mu'_j\right)^2 \right)$$

حال می‌توان بالاخره مقدار امید ln P(Y|h′) را بر اساس توزیع احتمال حاکم بر Y یا به طور مشابه توزیع احتمال حاکم بر متغیر های غیر قابل مشاهده‌ی Y (z_{ij} ها) محاسبه کرد. توجه دارید که عبارت بالا برای ln P(Y|h′) تابعی خطی از z_{ij} هاست. در کل برای هر تابع f(z) که تابعی خطی از z است رابطه‌ی زیر درست است،

$$E[f(z)] = f(E[z])$$

با استفاده از حقیقت بالا درباره‌ی توابع خطی می‌توان نوشت،

$$E[\ln P(Y|h')] = E\left[\sum_{i=1}^{m} \left(ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} z_{ij}\left(x_i - \mu'_j\right)^2 \right)\right]$$

$$= \sum_{i=1}^{m} \left(ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[z_{ij}]\left(x_i - \mu'_j\right)^2 \right)$$

برای خلاصه سازی تابع Q(h′|h) در مسئله‌ی k میانگین به صورت زیر است،

$$Q(h'|h) = \sum_{i=1}^{m}\left(ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}]\left(x_i - \mu'_j\right)^2 \right)$$

در این رابطه $h' =< \mu'_1, \dots, \mu'_k >$ و $E[z_{ij}]$ نیز بر اساس فرضیه‌ی فعلی h و داده های مشاهده شده‌ی X محاسبه می‌شود. همان طور که قبلاً نیز نشان دادیم،

$$E[z_{ij}] = \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}} \qquad (6.29)$$

بنابراین مرحله اول (تخمین) الگوریتم EM تابع Q را بر اساس تخمین $E[z_{ij}]$ تعریف می‌کند. مرحله‌ی دوم (ماکزیمم سازی) نیز مقادیر $\mu'_1, \dots, \mu'_k$ را پیدا خواهد کرد که این تابع Q را ماکزیمم کند. در مثال فعلی داریم که،

$$\arg\max_{h'} Q(h'|h) = \arg\max_{h'} \sum_{i=1}^{m}\left(ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}]\left(x_i - \mu'_j\right)^2 \right)$$

$$= \arg\min_{h'} \sum_{i=1}^{m}\sum_{j=1}^{k} E[z_{ij}]\left(x_i - \mu'_j\right)^2 \qquad (6.30)$$

محتمل‌ترین فرضیه اینجا مجموعی وزن دار از خطا های مربعی را مینیمم می‌کند، در این خطا مربع اختلاف x_i ها با μ'_j با وزن $E[z_{ij}]$ مینیمم می‌شود. کمیت رابطه‌ی ۶.۳۰ با قرار دادن مقادیر μ'_j به صورت زیر مینیمم می‌شود،

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}]x_i}{\sum_{i=1}^{m} E[z_{ij}]} \qquad (6.31)$$

توجه دارید که روابط ۶.۲۹ و ۶.۳۱ دو مرحله‌ی الگوریتم k میانگین قسمت ۶.۱۲.۱ را توصیف می‌کنند.

۶.۱۳ خلاصه و منابع برای مطالعه‌ی بیشتر

این فصل شامل موارد زیر می‌شود:

- متدهای بیزی پایه ای برای متدهای یادگیری احتمالی‌ای با دانش قبلی یا فرض آن درباره‌ی احتمالات ثانویه فرضیه‌ها و احتمال مشاهده‌ی نمونه‌ها است. متد های بیزی نسبت دادن احتمال ثانویه به هر فرضیه‌ی ممکن را بر اساس احتمالات اولیه‌ی مفروض را ممکن می‌سازند.
- از متد های بیزی می‌توان برای تعیین محتمل‌ترین فرضیه با فرض داشتن داده‌ها استفاده کرد، فرضیه‌ی MAP. این فرضیه‌ی از این جهت بهینه است که از تمامی فرضیه های دیگر محتمل‌تر است.

- دسته بندی کننده‌ی بهینه‌ی بیز پیش بینی تمامی فرضیه های ممکن را با احتمالات ثانویه‌شان ترکیب کرده محتمل‌ترین دسته بندی هر نمونه را به ما می‌دهد.
- دسته بندی کننده‌ی ساده‌ی بیز متد یادگیری بیزی است که در بسیاری از موارد کاربردی مفید شناخته شده است. به این الگوریتم به این دلیل "ساده" می‌گویند که شامل این فرض ساده کننده می‌باشد که ویژگی‌های نمونه‌ها با فرض داشتن دسته بندی نمونه مستقلند. با این فرض، دسته بندی کننده‌ی ساده‌ی بیز یک فرضیه MAP را خروجی خواهد داد. حتی زمانی که این فرض هم درست نیست هم، مثل حالتی که از این دسته بندی کننده برای دسته بندی متون استفاده کردیم، گاهی دسته بندی کننده‌ی ساده بیزی موثر است. شبکه های بیزی نمایش بهتری نسبت به مجموعه فرض‌ها استقلال در بین زیر مجموعه ای از ویژگی‌ها دارند.
- چهار چوب استدلال بیزی می‌تواند پایه مناسبی برای بررسی متدهای یادگیری بخصوص که مستقیماً از قضیه‌ی بیز استفاده نمی‌کنند باشد. برای مثال در شرایط خاص می‌توان نشان داد که زمانی که تابع هدف حقیقی مقداری را با مینیمم کردن مجموع خطاهای مربعی یاد می‌گیریم، محتمل‌ترین فرضیه را یاد می‌گیریم.
- قانون حداقل طول توضیح توصیه می‌کند که فرضیه‌هایی را انتخاب کنیم که کمترین طول توضیح برای فرضیه به اضافه‌ی طول توضیحات همراه فرضیه را داشته باشد. قضیه‌ی بیز و نتایج پایه ای تئوری اطلاعات را می‌توان برای ایجاد دلیلی برای این قانون به کار برد.
- در بسیاری از کارهای یادگیری عملی، بعضی از ویژگی‌های نمونه های ممکن است قابل مشاهده نباشد. الگوریتم EM روش کلی‌ای برای یادگیری در حضور متغیر های غیر مشهود ارائه می‌کند. این الگوریتم کار خود را با مجموعه ای از فرضیه های دلخواه آغاز می‌کند. سپس مقدار امید متغیر نامشهود را محاسبه کرده (با این فرض که فرضیه فعلی درست است). و سپس مقدار محتمل‌ترین فرضیه را محاسبه می‌کند (با فرض اینکه متغیر های پنهان همان مقادیر امید محاسبه شده‌ی این مرحله هستند). تکرار این فرایند به یک ماکزیمم نسبی در احتمال درستی فرضیه میل می‌کند و مقادیر متغیر های پنهان را نیز تقریب می‌زند.

کتب آموزشی ساده‌ی بسیاری درباره‌ی احتمالات و آمار مثل Casella and Berger (1990) نوشته شده است. کتب مرجع سریع بسیاری نیز مثل Maisel (1971) و Speigel (1991) نوشته شده، این کتب نماد گذاری آمار و احتمال متناسب با یادگیری ماشین را نیز ارائه می‌کنند.

بسیاری از نمادگذاری‌های ابتدایی دسته بندی کننده های بیزی و دسته بندی کننده های مینیم خطای مربعی در Duda and Hart (1973) بررسی شده است. Domingos and Pazzani (1996) شرایط اینکه دسته بندی کننده‌ی ساده‌ی بیز، دسته بندی بهینه را خروجی می‌دهد تحلیل می‌کند، این بررسی در حالتی انجام شده که شرط استقلال دسته بندی کننده‌ی ساده‌ی بیز ممکن است درست نباشد (نکته در این است که شروطی وجود دارد که دسته بندی درست باشد اما احتمالات ثانویه درست نباشند.)

Cestnik (1990) بحث درباره‌ی استفاده از تخمین m برای دسته بندی احتمالات را مطرح می‌کند.

نتایج تجربی که از مقایسه‌ی روش‌های مختلف بیزی و درخت تصمیم و دیگر الگوریتم‌های یادگیری انجام شده در Michie et al. (1994) آورده شده است. Chauvin and Rumelhart (1995) بررسی بیزی شبکه های عصبی را که بر اساس الگوریتم backpropagation است را مطرح می‌کنند.

بحث بر روی قانون کمترین طول توضیح را می‌توانید در Rissanen (1983,1989) بیابید. Quinlan and Rivest (1989) نیز استفاده از این قانون را در اجتناب از overfit در درخت‌های تصمیم را بررسی می‌کنند.

تمرینات

۶.۱ دوباره مثال عملی قانون بیز در قسمت ۶.۲.۱ را در نظر بگیرید. فرض کنید که دکتر تصمیم می‌گیرد که دستور دهد که آزمایش دومی انجام شود، و فرض کنید که نتیجه‌ی آزمایش دوم نیز مثبت است. احتمال ثانویه‌ی ¬cancer و cancer را محاسبه کنید. فرض کنید که دو تست مستقلند.

۶.۲ در مثال قسمت ۶.۲.۱ احتمال ثانویه‌ی cancer را با نرمالیزه کردن P(+|cancer).P(cancer) و P(+|¬cancer).P(¬cancer) به صورتی که مجموعشان یک شود محاسبه کردیم. از قضیه‌ی بیز قضیه مجموع احتمالات (با توجه به جدول ۶.۱) برای اثبات این متد استفاده کنید. (ثابت کنید که نرمالیزه کرده به این صورت مقدار درستی برای P(cancer|+) ارائه می‌کند).

۶.۳ الگوریتم یادگیری مفهوم FindG را در نظر بگیرید، که کلی‌ترین فرضیه‌ی ممکن ساخته شده از فرضیه‌ها را ارائه می‌کند (کلی‌ترین اعضای فضای فرضیه ای متناسب با نمونه های آموزشی).

(a) توزیعی برای P(h) و P(D|h) ارائه کنید (فرض کنید FindG تضمین می‌کند که همیشه فرضیه ای MAP خروجی دهد)

(b) توزیعی برای P(h) و P(D|h) ارائه کنید (فرض کنید FindG تضمین نمی‌کند که همیشه فرضیه ای MAP خروجی دهد)

(c) توزیعی برای P(h) و P(D|h) ارائه کنید (فرض کنید FindG تضمین نمی‌کند که همیشه فرضیه ای ML خروجی دهد)

۶.۴ در بررسی یادگیری مفهوم در بخش ۶.۳ فرض کردیم که ترتیب نمونه های $< x_1, \dots, x_m >$ همیشه ثابت است. بنابراین برای استخراج عبارتی P(D|h) فقط کافی است که احتمال مشاهده‌ی سری‌ای از مقادیر هدف $< d_1, \dots, d_m >$ را برای این سری ثابت نمونه‌ها بررسی کنیم. حالت کلی‌تری را که در ان نمونه‌ها ثابت نیستند را در نظر بگیرید، اما فرض کنید که تمامی نمونه‌ها با توزیع مشخصی روی X انتخاب می‌شوند. داده های D را باید اکنون به فرم زوج‌های مرتب $\{< x_i, d_i >\}$ نشان داده و در P(D|h) باید احتمال حضور x_i ها را علاوه بر d_i دخیل کرد. نشان دهید که رابطه‌ی ۶.۵ در این حالت کلی‌تر نیز درست است. (راهنمایی: بررسی رابطه‌ی ۶.۵ را نیز در نظر بگیرید)

۶.۵ قانون کمترین طول توضیح را در نظر بگیرید که به فضای فرضیه ای H ی که شامل عطف n متغیر منطقی است اعمال می‌شود. واضح است که هر فرضیه به سادگی با ویژگی‌های موجود در فرضیه توصیف می‌شود، اگر تعداد بیت‌های لازم برای هر یک از متغیرها $\log_2 n$ است. فرض کنید که کد سازی هر نمونه با داشتن فرضیه نیاز به صفر بیت دارد اگر نمونه سازگار با فرضیه باشد و نیاز به $\log_2 m$ بیت دارد اگر نمونه با فرضیه سازگار نباشد، m تعداد نمونه‌هایی است که اشتباه دسته بندی می‌شوند. (برای تعیین اینکه کدام یک از m نمونه‌ی اشتباه دسته بندی شده، دسته بندی درست را می‌توان به نقیض آنچه فرضیه دسته بندی می‌کند دانست)

(a) رابطه‌ی لازم برای کمیتی که باید بنا بر قانون کمترین طول مینیمم شود را بیابید.

(b) آیا ممکن است که دسته ای از داده های آموزشی موجود باشد که فرضیه ای سازگار با آن‌ها وجود داشته باشد اما MDL فرضیه ای با سازگاری کمتر را بر گزیند؟ اگر چنین است ان مجموعه را بیابید. اگر خیر، توضیح دهید چرا.

(c) توزیع احتمال P(h) و P(D|h) را برای اینکه الگوریتم MDL فوق فرضیه ای MAP را خروجی دهد بیابید.

۶.۶ شبکه‌ی باور بیزی‌ای را که فرض استقلال دسته بندی کننده‌ی ساده‌ی بیز را برای مفهوم PlayTennis در مسئله‌ی قسمت ۶.۹.۱ بکشید. جدول احتمال شرطی مربوطه‌ی گره باد را نیز رسم کنید.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

احتمال اولیه	prior probability
احتمال ثانویه	posterior probability
آنتروپی دورگه	cross entropy
اندازه‌ی نمونه‌ی معادل	equivalent sample size
بدون شعور	brute-force
برآمد	Outcome
تخمین m	m estimate
تعریف مسئله	problem setting
جرم احتمال	probability mass
چگالی احتمال	probability density
دسته بندی کننده‌ی بهینه‌ی بیز	bayes optimal classifier
دسته بندی کننده‌ی ساده‌ی بیز	naive Bayes classifier
زیرین	Descendant
شبکه های باور بیزی	bayesian belief networks
فرضیه با حداکثر احتمال	Maximum A Posteriori
فضای توأم	joint space
قانون کمترین طول توضیح	Minimum description length
الگوریتم گیبس	Gibbs algorithm
محتمل‌ترین	maximum likelihood
معیار	Criterion
یادگیر سازگار	consistent learner
یال	Arc

Telegram: @Farabar_BI
www.farabar.net

فصل هفتم : یادگیری محاسباتی

این فصل به صورت تئوری ویژگی های چندین نوع مسئله ی یادگیری ماشین و قابلیت های چندین نوع از الگوریتم های یادگیری ماشین را بیان می کند. این تئوری به دنبال جواب سوالاتی چون "تحت چه شرایطی یادگیری موفق ممکن یا غیرممکن است؟" و "تحت چه شرایطی یک الگوریتم یادگیری خاص موفقیت یادگیری را تضمین می کند؟" است. دو چهارچوب[1] برای بررسی یادگیری الگورتیم های یادگیری در نظر گرفته می شود. چارچوب اول چهارچوب تقریبا درست[2] یا (PAC) است، که در آن چارچوب کلاس فرضیه هایی را که می توان یا نمی توان با تعداد چند جمله ای ای از نمونه های آموزشی یادگرفت را بررسی و معیاری طبیعی برای پیچیدگی فضای فرضیه ای که تعداد نمونه های آموزشی برای یادگیری استقرایی را محدود می کند تعریف خواهیم کرد. در چارچوب کران خطا[3] تعداد خطاهای آموزشی ای را که یادگیر قبل از تعیین فرضیه ی درست انجام می دهد را بررسی خواهیم کرد.

۷.۱ مقدمه

در مطالعه ی یادگیری ماشین این سوال طبیعی است که بپرسیم چه قوانین کلی ای بر یادگیر های ماشین (یا غیر ماشین) حاکم است. آیا می توان کلاس های مسائل یادگیری را که ذاتا سخت یا آسانند را مستقل از الگوریتم یادگیری تعیین کرد؟ آیا می توان تعداد نمونه های لازم برای اینکه یادگیری حتما موفق باشد را تعیین کرد؟ اگر یادگیر بتواند بجای آموزش با دسته ی معینی از نمونه ها آزمایش انجام دهد (در مقابل اینکه نمونه ها به صورت تصادفی به یادگیر داده شوند) این تعداد چگونه تغییر خواهد کرد؟ یا آیا می توان تعداد خطای های یادگیر قبل از یادگیری تابع هدف را مشخص کرد؟ آیا می توان پیچیدگی محاسباتی ذاتی کلاسهای مسائل مختلف را مشخص کرد؟

[1] framework
[2] probably approximately correct
[3] mistake bound framework

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

اگر چه جواب جامع همه ی این سوالات هنوز معلوم نیست، اما قسمت از تئوری هوش محاسباتی برای پاسخ به این سوالات به وجود آمده است. این فصل نتایج کلیدی این تئوری و جواب به این سوالات در بعضی مسائل خاص را در بر می گیرد. در اینجا بحث را به مسئله ی یادگیری استقرایی تابع هدفی نامعلوم از نمونه های آموزشی این تابع هدف و فضای فرضیه ای معلوم محدود می کنیم. با این تعریف مسئله، پاسخ به سوالاتی مثل تعداد نمونه های لازم برای یادگیری موفق و تعداد اشتباهات قبل از یادگیری کامل مطرح می شود. همانطور که بعدا نیز خواهیم دید تعیین مرز های این کمیت ها به ویژگی های مسئله ی یادگیری از جمله موارد زیر وابسته است:

- اندازه یا پیچیدگی فضای فرضیه ای در نظر گرفته شده
- دقت لازم برای یادگیری
- احتمال اینکه یادگیر فرضیه ای موفق را خروجی دهد
- روند ارائه ی نمونه ها

در اکثر موارد، ما بر روی الگوریتم یادگیری خاصی تمرکز نمی کنیم و ترجیح داده می شود بیشتر بر روی کلاسهای الگوریتم های یادگیری با خواص یکسان (فضای فرضیه ای مشابه، نحوه ی نمایش نمونه های آموزشی مشابه و ...) بحث شود. هدف از این فصل پاسخ به سوالاتی نظیر سوالات زیر است:

- پیچیدگی نمونه ای[1]. تعداد نمونه های آموزشی لازم برای اینکه یادگیر (با احتمال بالایی) به فرضیه ای موفق میل کند؟
- پیچیدگی محاسباتی[2]. چه میزان محاسبه انجام می شود تا یادگیر با احتمال خوبی به فرضیه ای موفق میل کند؟
- مرز خطا[3]. تعداد نمونه ها آموزشی ای که یادگیر قبل از همگرا شدن به فرضیه موفق غلت دسته بندی می کند؟

توجه داشته باشید که در بسیاری از حالات چنین سوالاتی مطرح اند. برای مثال، روش های گوناگونی برای تعریف "موفق" وجود دارد. ممکن است یادگیری فرضیه ای را موفق تعریف کنیم که فرضیه ی خروجیش دقیقا مشابه مفهوم هدف باشد. یا در مقابل ممکن است یادگیری را موفق بدانیم که فرضیه اش در اکثر مواقع مشابه مفهوم هدف باشد، یا به طور معمول چنین فرضیه ای را خروجی می دهد. یا به طور مشابه، روند ارائه ی نمونه ها ممکن است متفاوت باشد، ممکن است این نمونه ها توسط یک معلم به یادگیر داده شود یا یادگیر اجازه ی انجام آزمایش داشته باشد یا اینکه نمونه ها توسط یک فرایند تصادفی خارج از کنترل یادگیر انتخاب شوند. همانطور که انتظار می رود، جواب این سوالات به تعریف مسئله و مدل یادگیری وابسته است.

ادامه ی این فصل به صورت زیر ساختار بندی شده است. قسمت ۷.۲ حالت یادگیری احتمالی تقریبا درست (PAC) را معرفی می کند. در ادامه، قسمت ۷.۳ پیچیدگی نمونه ای و پیچیدگی محاسباتی چندین مسئله ی یادگیری را در این حالت بررسی می کند. قسمت ۷.۴ معیار مهمی از پیچیدگی فضا[4] به نام بعد VC و تاثیر آن در بررسی PACمان در مسائلی که فضای فرضیه محدود است را بررسی خواهیم کرد. قسمت ۷.۵ مدل مرز خطا را معرفی کرده و مرزی برای تعداد خطاهای الگوریتمهای مختلف یادگیری فصول قبلی پیدا می کند. در انتها نیز، الگوریتم Weighted-Majority را معرفی می کنیم، این الگوریتم روشی برای تلفیق پیشبینی های الگوریتمهای مختلف رقیب است، مرز خطای تئوری این الگوریتم را نیز بررسی خواهیم کرد.

---

[1] Sample complexity
[2] Computational complexity
[3] Mistake bound
[4] space complexity

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۷.۲ احتمال یادگیری یک فرضیه ی تقریبا درست

در این بخش حالت خاصی را برای مسائل یادگیری در نظر می گیریم، این حالت مدل یادگیری تقریبا درست (PAC) نامیده می شود. بیایید کار را با این سوال که چه تعداد نمونه ی آموزشی و چه میزان محاسبه لازم است تا کلاس های مختلف یادگیری را با این مدل یاد بگیریم شروع کنیم. برای سادگی کار، بحث را به یادگیری مفاهیم منطقی از داده های آموزشی بدون خطا محدود می کنیم. با این وجود بسیاری از نتایج حاصل را می توان به حالت کلی یادگیری توابع حقیقی مقادیر تابع هدف تعمیم داد (برای مثال به (Natarajan 1991) مراجعه کنید) و بسیاری دیگر از نتایج را می توان به یادگیری از انواع خاصی از داده های خطا دار تعمیم داد (برای مثال به (Laird 1988) و (Kearns and Vazirani 1994) مراجعه کنید).

۷.۲.۱ تعریف مسئله

مشابه فصول گذشته، X مجموعه ی تمامی نمونه های ممکن بر روی تابع هدف مفروض است. برای مثال، X ممکن است مجموعه ی تمامی افراد باشد که با ویژگی های age (young or old) و height (short or tall) باشد. C مجموعه ی مفاهیم هدفی است که ممکن است یادگیر برای یادگیری آنها به کار برده شود. هر مفهوم هدف c در C متناسب با زیر مجموعه ای از X است یا به طور مشابه متناسب با تابع $c: X \rightarrow \{0,1\}$ است. برای مثال، یک تابع هدف c در C ممکن است مفهوم "افراد اسکی باز" باشد. اگر x نمونه ی مثبتی از c باشد، داریم که $c(x)=1$؛ و اگر x نمونه ی منفی ای باشد داریم $c(x)=0$.

در این حالت فرض می کنیم که نمونه ها به صورت تصادفی و با توزیع احتمال $\mathcal{D}$ انتخاب می شوند. برای مثال، $\mathcal{D}$ ممکن است توزیع احتمال نمونه ها افرادی باشد که از یک باشگاه ورزشی در سوئد بیرون می آیند (توزیع احتمالی بر روی تمامی افراد). در کل $\mathcal{D}$ ممکن است هر توزیع احتمالی باشد و در حالت کلی این توزیع احتمال برای یادگیر ناشناخته است. تمامی اطلاعات موجود در مورد $\mathcal{D}$ این است که توزیع احتمالی ثابت است؛ بدین معنا که این توزیع احتمال با زمان تغییر نمی کند. نمونه های آموزشی با این توزیع احتمال انتخاب شده و به همراه مقدار تابع هدف شان c(x) به یادگیر داده می شوند.

یادگیر L مجموعه ای از فرضیه های ممکن مثل H را در یادگیری مفهوم هدف در نظر می گیرد. برای مثال، H ممکن است مجموعه ی تمامی فرضیه های قابل بیان به صورت عطف ویژگی های age و height باشد. بعد از مشاهده ی سری ای از نمونه های آموزشی برای تابع هدف c، L باید فرضیه ای مثل h از H که تخمین آن از c است به عنوان فرضیه ی تخمینی خروجی دهد. موفقیت L را کارایی این فرضیه h بر روی نمونه های جدیدی که به صورت تصادفی از X و با توزیع $\mathcal{D}$ انتخاب می شوند می سنجیم. توزیع احتمال $\mathcal{D}$ همان توزیع احتمالی است که نمونه های آموزشی با انتخاب شده اند.

در چنین حالتی، علاقه ی ما به بررسی کارایی یادگیر های مختلف L با فضای فرضیه های مختلف H در یادگیری مجموعه توابع هدف مختلف درون C است. زیرا که می خواهیم یادگیر L به اندازه ی کلی جامع باشد تا بتواند هر تابع هدف درون C را مستقل از اینکه توزیع $\mathcal{D}$ چیست یاد بگیرد. در بعضی مواقع نیز علاقه داریم که در بدترین حالت توابع هدف درون C را برای تمامی توزیع های $\mathcal{D}$ را بررسی کنیم.

www.farabar.net

Telegram: @Farabar_BI

۷.۲.۲ خطای یک فرضیه

چون علاقه ی ما به نزدیکی فرضیه خروجی یادگیر h به تابع هدف حقیقی c است، بیایید کار را با تعریف خطای واقعی[1] یک فرضیه ی h بر روی c و توزیع احتمال $\mathcal{D}$ شروع کنیم. به صورت غیر رسمی خطای واقعی h، خطای h در دسته بندی نمونه های جدید با توزیع $\mathcal{D}$ است. در واقع این تعریف خطا همان تعریف خطا ی فصل ۵ است. برای راحتی تعریف را برای c که مفهومی منطقی است بازنویسی می کنیم.

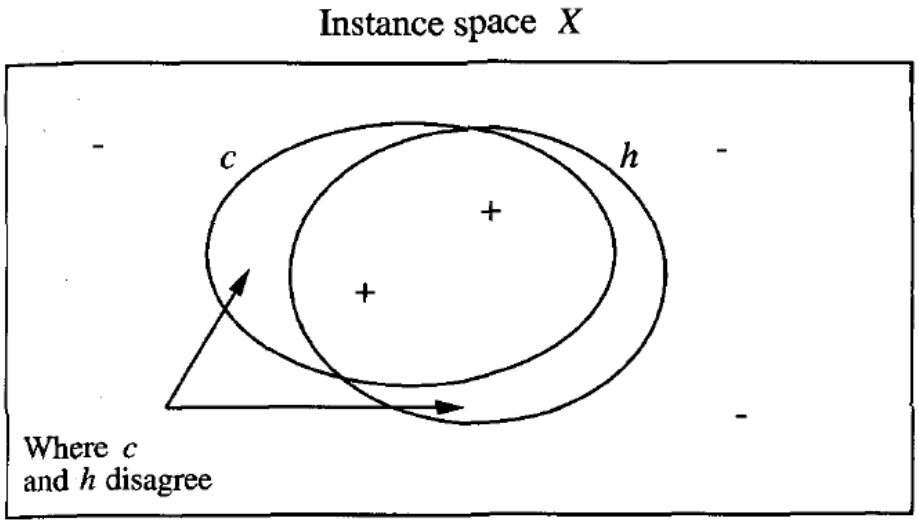
تعریف: خطای واقعی $(error_D(h))$ ی فرضیه ی h برای تابع هدف c و توزیع احتمال نمونه ای $\mathcal{D}$ احتمال این است که نمونه ی انتخابی بر اساس توزیع $\mathcal{D}$ اشتباه دسته بندی شود.

$$error_D(h) \equiv \Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$$

در اینجا نماد $\Pr_{x \in D}$ نشان دهنده ی احتمال عبارت با فرض اینکه x از توزیع $\mathcal{D}$ پیروی می کند است.

شکل ۷.۱ این تعریف را به فرم گرافیکی نشان می دهد. مفاهیم c و h با مجموعه ی نمونه های X نمایش داده شده اند، نمونه های آموزشی در این مثال با علامت های + و − نشان داده شده اند. خطای h برای c احتمال دسته بندی غلت نمونه تصادفی در این صفحه یا قرار گرفتن در اختلاف این دو مجموعه (قرار گرفتن در ناحیه هلالی) است. توجه دارید که خطا را طوری تعریف کرده ایم که خطای تمامی نمونه های ممکن را اندازه بگیرد و فقط محدود به نمونه های آموزشی نباشد بنابراین انتظار داریم که زمانی که از فرضیه بدست آمده بر روی نمونه های تصادفی جدید استفاده می کنیم چنین خطایی داشته باشند.

توجه دارید که این خطا به شدت به توزیع احتمال نا معلوم $\mathcal{D}$ وابسته است. برای مثال اگر $\mathcal{D}$ توزیعی یکنواخت باشد که به تمامی نمونه های X احتمالی یکسان نسبت می دهد خطای فرضیه ی آمده در شکل ۷.۱ نسبت نمونه های درون ناحیه هلالی به تمامی نمونه ها خواهد بود. با این وجود اگر $\mathcal{D}$ احتمال بیشتری به نمونه های ناحیه هلالی نسبت دهد این خطا بیشتر خواهد شد. و در بد ترین حالت $\mathcal{D}$ احتمال صفر به نمونه های خارج ناحیه هلالی نسبت می دهد و خطا ۱ خواهد بود با وجود اینکه h و c واقعا اشتراک دارند.



[1] True Error

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

شکل ۷.۱ خطای فرضیه h برای مفهوم هدف c.
خطای h برای مفهوم هدف c احتمال این است که نمونه ای تصادفی در درون ناحیه ای قرار بگیرد که h و c در آن ناحیه دسته بندی های مشابهی ندارند. نقاط + و – نشان دهنده ی نمونه های آموزشی مثبت و منفی اند. توجه داشته باشید که با وجود اینکه در تمامی ۵ نمونه ی مشاهده شده دسته بندی h و c یکی است، h خطای غیر صفری برای مفهوم هدف c دارد.

بلاخره، توجه داشته باشید که این خطای h برای c به طور مستقیم برای یادگیر غیر قابل مشاهده است. L فقط کارایی h بر روی نمونه های آموزشی را در دسترس دارد و باید انتخاب خود در مورد فرضیه را بر اساس همین معیار انجام دهد. ما از عبارت خطای آموزشی[1] (در مقابل خطای واقعی) برای نمایش نسبت نمونه های آموزشی با دسته بندی اشتباه توسط h به کل نمونه های آموزشی استفاده می کنیم. قسمت بزرگی از بررسی ما از پیچیدگی یادگیری بر محور این سوال متمرکز می شود که "چگونه احتمال دارد که خطای آموزشی مشاهده شده معیاری غلت انداز از $error_{\mathcal{D}}(h)$ باشد؟" است.

به رابطه ی بین این سوال و سوال مطرح شده در فصل ۵ دقت کنید. با توجه به آنچه در فصل ۵ گفته شد، خطای نمونه ای h را برای مجموعه ی S از نمونه ها نسبت دسته بندی اشتباه اعضای S توسط h تعریف شد. خطای آموزشی تعریف شده در بالا خطای نمونه ای S است با این فرض که S مجموعه ی نمونه های آموزشی باشد. در فصل ۵ احتمال اینکه خطای نمونه ای تخمینی غلت انداز از خطای واقعی باشد را با این فرض که داده های نمونه ی S مستقل از h باشند بررسی کردیم. اما اینجا حتی این فرض هم درست نیست و فرضیه ی h کاملا وابسته به مجموعه S است! بنابراین، در این فصل ما این حالت خاص مهم را بررسی خواهیم کرد.

۷.۲.۳ قابلیت یادگیری PAC

هدف ما تعیین ویژگی های توابع هدفی است که می توان آنها را از تعداد معقولی نمونه آموزشی تصادفی با پیچیدگی محاسباتی معقولی یادگرفت.

چه عبارت هایی را می توان درباره ی قابلیت یادگیری یک تابع بیان کرد درست فرض کرد؟ ممکن است سعی کنیم تعداد نمونه های آموزشی لازم برای یادگیری فرضیه ای با $error_{\mathcal{D}}(h) = 0$ را تعیین کنیم. متاسفانه در این تعریف مسئله به دو دلیل این کاری بیهوده است. ابتدا اینکه برای اینکه به چنین خطایی برسیم باید تمامی نمونه های X را به عنوان نمونه ی آموزش به یادگیر ارائه کنیم (که این فرضی غیر واقعی است)، و ممکن است چندین فرضیه با مجموعه نمونه های آموزشی سازگار باشند و یادگیر در انتخاب فرضیه ی تخمینی برای مفهوم هدف سر در گم خواهد ماند. دوم اینکه با معلوم بودن نمونه های آموزشی تصادفی، همیشه احتمالی غیر صفر وجود دارد که نمونه های آموزشی معیاری غلت انداز باشد. (برای مثال، با وجود اینکه اغلب قد افراد خارج شده از یک مجموعه ی ورزشی در سوئد متفاوت است اما احتمال غیر صفری وجود دارد که در یک روز تمامی نمونه های مشاهده شده قد ۲ متر داشته باشند).

برای غلبه بر این دو مشکل، شرایط خواستاری مسئله را از دو نظر کاهش می دهیم. ابتدا بجای اینکه شرط کنیم خطای فرضیه صفر شود شرط می کنیم که خطا از مقدار دلخواه کوچک ε کوچکتر باشد. دوم اینکه بجای اینکه شرط کنیم یادگیر روی هر نمونه ی آموزشی ممکن موفق باشد شرط می کنیم که احتمال عدم موفقیت کمتر از حد دلخواه کوچک خاصی، δ، کمتر باشد. به طور خلاصه شرط می کنیم که یادگیر به صورت احتمالی فرضیه ای تقریبا درست[2] را یاد بگیرد، بنابراین به این فرضیه، فرضیه ی احتمالی تقریبا درست یا PAC می گویند.

---

[1] training error
[2] approximately correct

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

مجموعه ی C را به عنوان مجموعه ی مفاهیم هدف ممکن و یادگیر L با فضای فرضیه ای H را در نظر بگیرید. به طور غیر رسمی، زمانی می گوییم که مجموعه ی C توسط L با استفاده از H قابل یادگیری PAC [1] است که، برای هر تابع هدف c در C، L با احتمال $(1-\delta)$ با داشتن تعداد قابل قبولی نمونه ی آموزشی و انجام مقدار قابل قبولی محاسبه فرضیه ای مثل h خروجی دهد که داشته باشیم، $error_{\mathcal{D}}(h) < \varepsilon$. به عبارت دقیقتر،

تعریف: مجموعه ی مفاهیم هدف C را که بر روی نمونه های X با اندازه ی n تعریف شده و یادگیر L با فضای فرضیه ای H را در نظر بگیرید. C توسط L با استفاده از H زمانی قابل یادگیری PAC است که برای تمامی $c \in C$ و توزیعهای $\mathcal{D}$ بر روی X و $0 < \varepsilon < 1/2$ و δ هایی که $0 < \delta < 1/2$، یادگیر L با احتمال حداقل $(1-\delta)$ فرضیه $h \in H$ خروجی دهد که داشته باشیم که $error_{\mathcal{D}}(h) \leq \varepsilon$، در زمان حداکثر چند جمله ای از $1/\delta$، $1/\varepsilon$، n و size(c) خروجی باید محاسبه شود.

تعریف ما دو شرط بر روی L می گذارد. ابتدا اینکه L باید با احتمال دلخواه بالایی $(1-\delta)$ فرضیه ای با مقدار خطای به اندازه ی دلخواه کوچکی ε خروجی دهد. دوم اینکه باید کارایی خوبی داشته باشد، در زمانی حداکثر چندجمله ای از $1/\delta$، $1/\varepsilon$، n و size(c) فرضیه ی خروجی را مشخص کند. در اینجا، n تعداد نمونه های X است. برای مثال، اگر نمونه های X، عطف k ویژگی منطقی باشند، خواهیم داشت که n=k. پارامتر دوم فضا، size(c) است که اندازه ی کدهای c های درون C را نشان می دهد با فرض اینکه برای C نمایش خاصی را تعیین کرده باشیم. برای مثال اگر مفاهیم C با عطف حداکثر k ویژگی منطقی باشند، که هر کدام با لیستی از ویژگی ها مشخص شود، size(c) تعداد ویژگی های واقعی به کار رفته در توضیح c خواهد بود.

ممکن است ابتدا به نظر برسد که تعریف ما از یادگیری PAC فقط اهمیت منابع محاسباتی لازم برای یادگیری در نظر گرفته شده است در حالی که در عمل تعداد نمونه های لازم برای یادگیری بیشتر برای ما اهمیت دارد. با این وجود، این دو بسیار به یکدیگر نزدیکند: اگر L نیاز به پردازش حداقلی برای هر نمونه ی آموزشی داشته باشد، برای اینکه C را قابل یادگیری PAC توسط L بدانیم، L حتما به تعداد چند جمله ای ای نمونه ی آموزشی نیاز خواهد داشت. در اصل روشی متداول برای نشان دادن اینکه C ای خاص قابل یادگیری PAC است این است که ابتدا نشان دهیم هر مفهوم هدف از تعداد چند جمله ای ای از نمونه های آموزشی قابل یادگیری است و سپس نشان دهیم زمان محاسبات نیز از چند جمله ای کمتر است.

قبل از رفتن به قسمت بعد، باید به فرض محدود کننده ای در تعریفمان از قابل یادگیری PAC اشاره کنیم. این تعریف مطلقا فرض می کند که فضای فرضیه ای یادگیر H شامل فرضیه هایی است که خطای به اندازه ی دلخواه کوچک برای تمامی مفاهیم درون C دارند. این فرض از این حقیقت ناشی می شود که در تعریف بالا یادگیر زمانی موفق است که بتوان ε را به اندازه ی دلخواه به صفر نزدیک کرد. البته در حالتی که C دقیق معلوم نیست (برای مثال، C در برنامه ای که باید تصاویر چهره را تشخیص دهد چیست؟) تظمین این شرط سخت خواهد بود، مگر اینکه H مجموعه ی توانی X در نظر گرفته شود. همانطور که در فصل ۲ نیز گفته شد، چنین H بدون بایاسی دقت کافی تعمیمی با تعداد قابل قبولی از نمونه های آموزشی پیدا نمی کند. با این وجود، نتایج حاصل از مدل یادگیری PAC دید مفیدی درباره ی پیچیدگی نسبی مسائل یادگیری مختلف و ضریب بهبود تعمیم با افزایش نمونه های آموزشی به ما می دهد. علاوه بر این، در بخش ۷.۳.۱ این فرض محدود کننده را برای در نظر گرفتن یادگیر بدون پیشفرض حذف می کنیم.

[1] PAC learnable

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۷.۳ پیچیدگی نمونه ای برای فضای فرضیه ای محدود

همانطور که در بالا نشان دادیم، قابلیت یادگیری PAC [1] به شدت به تعداد نمونه های آموزشی لازم برای یادگیر وابسته است. افزایش تعداد نمونه های آموزشی لازم برای یادگیری متناسب با اندازه ی مسئله، که پیچیدگی نمونه ای مسئله ی یادگیری نامیده می شود، از مهمترین معیار های مسائل یادگیری است. علت این اهمیت از این رو است که در مسائل عملی محدودیت موفقیت در یادگیری بیشتر به خاطر محدودیت یادگیر در تعداد نمونه های آموزشی است.

در اینجا ما مرزی کلی برای پیچیدگی نمونه ای برای کلاس بزرگی از یادگیر ها، یادگیر های سازگار[2] ارائه می کنیم. یادگیر سازگار یادگیری است که زمانی که ممکن باشد فرضیه ای را که با نمونه های آموزشی سازگار باشد خروجی می دهد. انتظار سازگار بودن یادگیر ها انتظاری دور از ذهن نیست، زیرا که معمولا ما فرضیه ای را که با نمونه های آموزشی سازگار است را به فرضیه های دیگر ترجیح می دهیم. توجه دارید که اکثر الگوریتم های مطرح شده در فصول قبلی از جمله تمامی الگوریتم های فصل ۲ سازگار هستند.

آیا می توانیم مرزی برای تعداد نمونه های آموزشی لازم برای هر یادگیر سازگار که مستقل از الگوریتم است پیدا کنیم؟ جواب این سوال بلی است. برای ایجاد چنین مرزی بد نیست که بعضی تعاریف فصل ۲ را درباره ی فضای ویژه بازگو کنیم. در آنجا فضای ویژه ی $VS_{H,D}$ را مجموعه ی تمامی فرضیه های h ∈ H را که نمونه های آموزشی D را درست دسته بندی می کنند تعریف کردیم.

$$VS_{H,D} = \{\mathrm{h} \in H | (\forall < x, c(x) > \in D)(\mathrm{h(x)} = \mathrm{c(x)})\}$$

اهمیت فضای ویژه در اینجا این است که هر یادگیر سازگار مستقل از اینکه X یا H یا D چه باشند فرضیه ای از فضای ویژه را خروجی می دهد. دلیل این نتیجه در تعریف فضای ویژه مشهود است، زیرا که فضای ویژه تمامی فرضیه های سازگار در H با نمونه های آموزشی را در بر می گیرد. بنابراین برای محدود کردن تعداد نمونه های آموزش لازم برای هر یادگیر سازگار کافی است تعداد نمونه های آموزشی لازم را برای تظمین اینکه فضای ویژه فرضیه ای غیر قابل قبول را در بر نگیرد معلوم کنیم. تعریف زیر، که به نام Haussler (1988) نام گذاری شده، این شرط را به صورت دقیق مشخص می کند.

تعریف: فضای فرضیه ای H، مفهوم هدف c، توزیع نمونه ای $\mathcal{D}$ و مجموعه ی نمونه های آموزشی D که برای آموزشی c است را در نظر بگیرید. فضای ویژه ی $VS_{H,D}$ زمانی ε-exhausted برای c و D است که برای هر فرضیه ی h در $VS_{H,D}$ خطایی کمتر از ε برای c و D داشته باشیم.

$$\left(\forall h \in VS_{H,D}\right) error_{\mathcal{D}}(h) < \varepsilon$$

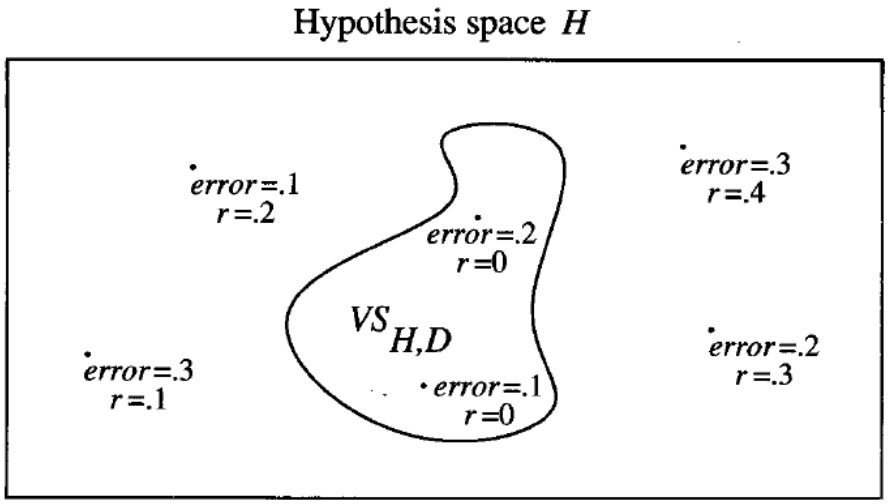
تعریف بالا در شکل ۷.۲ نمایش داده شده است. فضای ویژه زمانی ε-exhausted است که تمامی فرضیه های سازگار با نمونه های آموزشی مشاهده شده (برای مثال، آنهایی که خطای نمونه ای صفر دارند) خطایی کمتر از ε داشته باشند. البته از دید یادگیر فقط فرضیه هایی که به طور کامل با نمونه های آموزشی سازگار اند قابل تشخیص است، همگی آنها خطای آموزشی صفر خواهند داشت. فقط شاهدی که از ماهیت مفهوم هدف اگاه است می تواند با قطعیت فضای ویژه ی ε-exhausted را مشخص کند. جالب است که بررسی ای احتمالی به ما اجازه می دهد

[1] PAC-learnability
[2] consistent learners

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

که احتمال اینکه فضای ویژه بعد از تعدادی نمونه ی آموزشی ε-exhausted باشد را بدون اینکه اطلاعاتی در مورد ماهیت مفهوم هدف یا توزیع نمونه های آموزشی داشته باشیم محدود کنیم. Haussler (1988) چنین مرزی را با قضیه ی زیر ایجاد می کند.



شکل ۷.۲ exausted کردن فضای ویژه.

فضای ویژه ی $VS_{H,D}$ زیر مجموعه ای از فرضیه های $h \in H$ است که خطای آموزشی صفر دارند (در شکل با r=0 نشان داده شده است). البته خطای واقعی $error_{\mathcal{D}}(h)$ است (که در شکل با error نمایش داده شده) که حتی ممکن است برای فرضیه های فضای ویژه غیر صفر باشد. فضای ویژه زمانی ε-exausted است که تمامی فرضیه های باقیمانده ی درون $VS_{H,D}$ داشته باشیم $error_{\mathcal{D}}(h) < \varepsilon$.

قضیه ی ۷.۱ فضای ویژه ی ε-exhausted. اگر فضای فرضیه ای H محدود باشد و D نیز سرای ای از m≥1 نمونه ی تصادفی مستقل از مفهوم هدف c باشد، برای هر 0≤ε≤1 احتمال اینکه فضای ویژه ی $VS_{H,D}$ برای c ε-exhausted نباشد کمتر یا مساوی مقدار زیر است:

$$|H|e^{-\varepsilon m}$$

اثبات. فرض کنید $h_1, h_2, \dots h_k$ تمامی فرضیه های درون H باشند که خطای واقعی بیشتر از ε برای c دارند. اگر و فقط اگر حداقل یکی از این k فرضیه با تمامی نمونه های آموزشی سازگار باشد فضای ویژه ε-exhausted نخواهد بود. احتمال اینکه فرضیه ای که خطای واقعی بیشتر از ε دارد با نمونه ای که به صورت اتفاقی انتخاب می شود سازگار باشد حداکثر (1-ε) است. بنابراین احتمال اینکه این فرضیه با m نمونه ی مستقل سازگار باشد $(1-\varepsilon)^m$ خواهد بود. حال اگر k فرضیه خطایی بیشتر از ε داشته باشند، احتمال اینکه حداقل یکی از این فرضیه ها با تمامی m نمونه ی آموزشی سازگار باشد حداکثر

$$k(1-\varepsilon)^m$$

است و از آنجایی که $k \le |H|$، پس این مقدار حداکثر $|H|(1-\varepsilon)^m$ خواهد بود. بلاخره، از رابطه ی کلی $0 \le \varepsilon \le 1$ داریم که $(1-\varepsilon) \le e^{-\varepsilon}$. بنابراین،

$$k(1-\varepsilon)^m \le |H|(1-\varepsilon)^m \le |H|e^{-\varepsilon m}$$

که قضیه به اثبات می رسد.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

این قضیه کران بالایی بر حسب تعداد نمونه های آموزشی m و حداکثر خطای مجاز ε واندازه ی H برای احتمال اینکه فضای ویژه ε-exhausted نباشد ارائه می کند. اما از نظر دیگر، این مرز احتمال اینکه m نمونه ی آموزشی در حذف تمامی فرضیه های "بد" (فرضیه هایی که خطای واقعی بیشتر از ε دارند) در یادگیر سازگار با فضای فرضیه ای H موفق نشوند را نشان می دهد.

بیایید از این نتیجه برای تعیین تعداد نمونه های آموزشی لازم برای کاهش احتمال شکست به زیر حد دلخواه δ استفاده کنیم.

$$|H|e^{-\varepsilon m} \leq \delta \qquad (7.1)$$

با بازنویسی رابطه برای m داریم که

$$m \geq \frac{1}{\varepsilon}\left(\ln|H| + \ln\left(\frac{1}{\delta}\right)\right) \qquad (7.2)$$

به طور خلاصه نامساوی رابطه ی ۷.۲ مرزی کلی برای تعداد نمونه های آموزشی لازم برای اینکه تمامی یادگیر های سازگار با موفقیت هر مفهوم هدف درون H را برای مقادیر دلخواه δ و ε یادبگیرند را مشخص می کند. این تعداد نمونه ی آموزشی برای تظمین اینکه هر فرضیه ی سازگار تقریبا (با احتمال (1-δ) درست) درست (حداکثر با خطای ε) باشد را تعیین می کند. توجه دارید که m به صورت خطی با 1/ε و لگاریتمی 1/δ متناسب است. همچنین با نسبت لگاریتمی با اندازه ی فضای فرضیه ای H نیز متناسب است.

توجه دارید که مرز بالا می تواند ذاتا اغراق آمیز باشد. برای مثال، با وجود اینکه احتمال اینکه فضای ویژه باید در بازه ی [0,1] قرار بگیرد، اما این مرز با افزایش |H| به صورت خطی افزایش پیدا می کند. برای فضای های فرضیه ای به اندازه ی کافی بزرگ، این مرز می تواند به راحتی بزرگتر از یک شود. نتیجه اینکه مرز نامساوی ۷.۲ می تواند ذاتا برای تعداد نمونه های آموزشی اغراق آمیز باشد. ضعف این مرز معمولا در جمله ی |H| است که از اثبات هنگام جمع احتمالات یک فرضیه ی غیر قابل قبول در میان تمامی فرضیه ها ایجاد می شود. در واقع، در بسیاری مواقع مرزی کوچکتر فضاهای فرضیه ای بینهایت بزرگ را محدود می کند. این مرز موضوع قسمت ۷.۴ خواهد بود.

۷.۳.۱ یادگیری agnostic و فرضیه های غیر سازگار

رابطه ی ۷.۲ از این جهت اهمیت دارد که تعداد نمونه های آموزشی لازم برای تظمین اینکه (با احتمال (1-δ)) هر فرضیه ی H که خطای آموزشی صفر دارد خطای واقعی حداکثر ε داشته باشد را تعیین می کند. متاسفانه اگر H شامل تابع هدف c نباشد، همیشه نمی توان فرضیه ای پیدا کرد که خطای آموزشی صفر داشته باشد. در چنین حالتی، از یادگیر می خواهیم که فرضیه ای را خروجی دهد که کمترین خطای ممکن را بر روی نمونه های آموزشی داشته باشد. یادگیری که هیچ پیشفرضی در مورد تابع هدف نمی کند و فقط فرضیه ای از H را که کمترین خطای آموزشی دارد را خروجی می دهد، یادگیری agnostic نامیده می شود، زیرا که هیچ فرض قبلی ای برای اینکه آیا C⊆H درست است یا خیر نمی کند.

با وجود اینکه رابطه ی ۷.۲ بر این فرض که یادگیر فرضیه ای با خطای صفر را خروجی می دهد پایه گذاری شده است، اما مرزی مشابه را می توان برای حالت کلی تری که یادگیر فرضیه ای با خطای آموزشی غیر صفر را خروجی می دهد می توان بدست اورد. به عبارت دقیقتر فرض کنید که D مجموعه ی خاصی از نمونه ای آموزشی موجود است (البته با $\mathcal{D}$ که توزیع نمونه ای است متفاوت است) و $error_D(h)$ خطای نمونه ای فرضیه ی h بر روی D است. در کل، $error_D(h)$ نسبت اشتباه های h بر روی نمونه های آموزشی D تعریف شده است. توجه دارید که در حالت کلی $error_D(h)$ با $error_{\mathcal{D}}(h)$ که خطای واقعی بر روی توزیع نمونه ای است متفاوت است. حال فرض کنید

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

h_{best} نماد فرضیه ای از H باشد که کمترین خطای نمونه ای را بر روی نمونه های آموزشی دارد. چه تعداد نمونه ی آموزش لازم است تا تضمین شود که (با احتمال قوی) خطای واقعی $error_{\mathcal{D}}(h_{best})$ کمتر یا مساوی $error_D(h_{best})$ باشد؟ توجه دارید که رابطه ی این سوال با سوال مطرح شده در قسمت قبلی این است که سوال قسمت قبلی حالت خاصی از این سوال بود (حالتی که $error_D(h_{best}) = 0$).

این سوال را می توان با استفاده از تشابه با اثبات قضیه ی ۷.۱ جواب داد (به رابطه ی ۷.۳ مراجعه کنید). یادآوری مرز های Hoeffding [1] (که گاهی مرز های اضافی Hoeffding [2] نامیده می شود.) در اینجا مفید است. مرز های Hoeffding مشتق بین احتمال واقعی یک اتفاق و میزان مشاهده ی آن اتفاق در m آزمایش مستقل را بررسی می کند. به عبارت دقیقتر، این مرز ها به m آزمایش مستقل برنولی[3] اعمال می شوند (برای مثال در m پرتاب سکه ای با احتمال شیر آمدنی خاص). این کاملا مشابه تعریف مسئله ی ما در تخمین خطای فرضیه در فصل ۵ است: احتمال اینکه شیر بیاید مشابه احتمال این است که فرضیه یک نمونه ی تصادفی را اشتباه دسته بندی کند. m پرتاب مستقل سکه مشابه انتخاب m نمونه ی مستقل از توزیع نمونه ای است. نسبت تعداد شیر ها به کل m پرتاب مشابه نسبت دسته بندی های اشتباه به کل m نمونه ی تصادفی است.

مرز های Hoeffding می گوید که اگر خطای نمونه ای $error_D(h)$ بر روی مجموعه ی D شامل m نمونه ی تصادفی باشد، خواهیم داشت که:

$$\Pr[error_{\mathcal{D}}(h) > error_D(h) + \varepsilon] \leq e^{-2m\varepsilon^2}$$

این رابطه مرزی برای احتمال اینکه یک فرضیه ی دلخواه خطای نمونه ای بسیار گمراه کننده داشته باشد را به ما می دهد. برای اینکه مطمئن باشیم که بهترین فرضیه پیدا شده توسط L حداکثر خطایی با این مرز دارد، باید احتمال اینکه هر فرضیه از |H| فرضیه های موجود خطای بزرگی داشته باشند را در نظر گرفت.

$$\Pr[(\exists h \in H)(error_{\mathcal{D}}(h) > error_D(h) + \varepsilon)] \leq |H|e^{-2m\varepsilon^2}$$

اگر این احتمال را δ بنامیم و به دنبال تعداد نمونه های لازم برای کمتر بودن δ از مقدار خاصی بگردیم به این رابطه می رسیم که:

$$m \geq \frac{1}{2\varepsilon^2}\left(\ln|H| + \ln\left(\frac{1}{\delta}\right)\right) \qquad (7.3)$$

این رابطه تعمیم رابطه ی ۷.۲ برای حالتی است که یادگیر هنوز بهترین فرضیه h∈H را انتخاب می کند و خطای نمونه ای بهترین فرضیه نیز می تواند غیر صفر باشد. توجه دارید که m با H و 1/δ رابطه ای لگاریتمی دارد و همانطور که در مشاهده می شود که به حالت خاص تر ۷.۲ می رسیم. با این وجود در این حالت کلی تر m به جای رابطه ی خطی متناسب با مجذور 1/ε است.

[1] Hoeffding bounds
[2] Hoeffding additive bounds
[3] Bernoulli trial

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۷.۳.۲ عطف عبارات منطقی PAC-Learnable است

حال که مرزی برای تعیین تعداد نمونه های آموزشی کافی برای اینکه بتوان با احتمال خوبی تابع هدف را یادگرفت بدست آورده ایم، از این مرز می توانیم برای تعیین پیچیدگی نمونه ای و Pac-learnable بودن دسته ی خاصی از مفاهیم هدف استفاده کرد.

مجموعه ی مفاهیم هدف C را که با عطف عبارات منطقی بیان می شود را در نظر بگیرید. یک عبارت منطقی می تواند هر متغیر منطقی (مثل Old) یا نقیضش (مثل ¬Old) باشد. بنابراین عطف عبارات منطقی مثل توابع هدفی چون "Old ∧ ¬Tall" را نیز شامل می شود. آیا C قابل یادگیری PAC است؟ می توان نشان داد که پاسخ چنین سوالی آری است. کافی است ابتدا نشان دهیم هر یادگیر سازگار فقط تعداد چند جمله ای ای نمونه ی آموزشی برای یادگیری هر c در C لازم دارد و الگوریتمی ارائه کنیم که زمانی چند جمله ای برای هر نمونه لازم داشته باشد تا مفهوم هدف را یاد بگیرد.

یادگیر L را یک یادگیر سازگاری است در نظر بگیرید که از فضای فرضیه ای H که مشابه C است استفاده می کند. از رابطه ی ۷.۲ می توان برای محاسبه ی تعداد m نمونه ی آموزشی تصادفی کافی تا یادگیر L با احتمال (1-δ) فرضیه ای خروجی با ماکزیمم خطای ε بدهد استفاد کرد. برای این کار، لازم است که |H| را که اندازه ی فضای فرضیه ای مربوطه است را تعیین کرد.

حال فضای فرضیه ای H را که بر روی عطف n عبارات منطقی تعریف می شود را در نظر بگیرید. اندازه ی |H| در این فضای فرضیه ای 3^n است. توجه داشته باشید که هر عبارت ممکن است در فرضیه سه حالات داشته باشد: فرضیه آنرا شامل می شود، فرضیه نقیض آنرا شامل می شود، فرضیه در مورد آن نظری نداده است. پس اگر n متغیر داشته باشیم می توانیم 3^n فرضیه روی آنها تعریف کنیم.

با اضافه کردن $|H| = 3^n$ در رابطه ی ۷.۲ مرز پیچیدگی نمونه ای یادگیری عطف n عبارت منطقی به صورت زیر بدست می آید.

$$m \geq \frac{1}{\varepsilon}\left(nln3 + \ln\left(\frac{1}{\delta}\right)\right) \tag{7.4}$$

برای مثال اگر یک یادگیر سازگار، یادگیری بخواهد تابع هدفی توصیفی با ۱۰ عبارت و با احتمال درستی بیش از ۹۵ درصد فرضیه ای با خطای کمتر از 1. را یاد بگیرد، برای m که تعداد نمونه های آموزشی تصادفی لازم برای این کار خواهد بود خواهیم داشت که،

$$.m = \frac{1}{.1}\left(10ln3 + \ln\left(\frac{1}{.05}\right)\right) = 140$$

توجه دارید که m رابطه ی خطی و مستقیم با n (تعداد عبارات فضای فرضیه ای)، 1/ε و رابطه ای لگاریمتی با 1/δ دارد. اما رابطه ی m با میزان محاسبات کلی چقدر است؟ البته محاسبات به نوع الگوریتم یادگیری وابسته است. با این وجود، تا زمانی که الگوریتم ما محاسباتی کمتر از چند جمله ای برای هر نمونه داشته باشد و کل محاسبات نیز کمتر از چند جمله ای کل نمونه های آموزشی باشد، مسلما محاسبات کل نیز کمتر از چند جمله ای تعداد نمونه ها خواهد بود.

در یادگیری عطف عبارات منطقی، یکی از الگوریتم هایی که شرایط لازم را دارد در فصل ۲ مورد بحث قرار گرفت. این الگورتیم Find-S است، که خاص ترین فرضیه ی سازگار با نمونه های آموزشی را محاسبه می کند. برای هر نمونه ی مثبت آموزشی جدید، الگوریتم اشتراک بین عباراتی فرضیه ی فعلی و نمونه ی آموزشی جدید را در زمانی با رابطه ی خطی با n محاسبه می کند. بنابراین، الگوریتم Find-S کلاس مفاهیم عطفی n عبارت منطقی و نقیضشان را به فرم PAC یاد می گیرد.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

قضیه ی ۷.۲ قابلیت یادگیری PAC عطف عبارات منطقی. کلاس C که مجموعه ی عطف عبارات منطقی است توسط الگوریتم Find-S با استفاده از H=CFind-S قابل یادگیری PAC است.

اثبات. رابطه ی ۷.۴ نشان می دهد پیچیدگی نمونه ای این کلاس مفاهیم نسبت به $1/\delta$ و $1/\varepsilon$ چندجمله ای و از size(c) مستقل است. برای پردازش مرحله به مرحله ی هر نمونه ی آموزشی، الگوریتم Find-S نیاز به تلاشی متناسب خطی با n و مستقل از $1/\delta$ و $1/\varepsilon$ و size(c) خواهد داشت. بنابراین این کلاس مفاهیم توسط الگوریتم Find-S، قابل یادگیری PAC است.

۷.۳.۳ قابلیت یادگیری PAC دیگر کلاسهای مفهوم

همانطور که در بالا دیدیم، رابطه ی ۷.۲ پایه ای کلی برای محدود کردن پیچیدگی یادگیری توابع مفهوم کلاس معلوم C ارائه می کند. در بالا این رابطه را برای کلاس عطف عبارات منطقی به کار بردیم. به طور مشابه می توان نشان داد که بسیاری از کلاسهای مفهوم پیچیدگی نمونه ای چند جمله ای دارند. (تمرین ۷.۲)

۷.۳.۳.۱ یادگیر های بدون بایاس

همه ی کلاسهای مفهوم مرز پیچیدگی نمونه ای با رابطه ی ۷.۲ محدودی ندارند. برای مثال کلاس مفاهیم بایاس نشده ی C را که تمامی مفاهیم قابل تعلیم بر روی X را در بر می گیرد را در نظر بگیرید. مجموعه ی C تمامی مفاهیم هدف قابل تعریف همان مجموعه ی توانی X، مجموعه ی تمامی زیرمجموعه های X، خواهد بود که $|C| = 2^{|X|}$. فرض کنید که نمونه های درون X با n متغیر منطقی تعریف شوند، بنابراین خواهیم داشت که، $|X| = 2^n$ بنابرای خواهیم داشت که $|C| = 2^{2^n}$. البته برای یادگیری چنین کلاس بایاس نشده ی یادگیری، خود یادگیر نیز باید از فضای فرضیه ای بدون بایاس استفاده کند H=C. با جایگزاری $|H| = 2^{2^n}$ در رابطه ی ۷.۲ پیچیدگی نمونه ای برای یادگیر مفاهیم بدون بایاس روی X مشخص می شود.

$$m \geq \frac{1}{\varepsilon}\left(2^n ln2 + \ln\left(\frac{1}{\delta}\right)\right) \tag{7.5}$$

بنابراین، این کلاس بدون بایاس از مفاهیم هدف بنابر رابطه ی ۷.۲ پیچیدگی نمونه ای نمایی (exponential) در مدل PAC دارد. با وجود اینکه رابطه ی ۷.۲ پیچیدگی نمونه ای با اغراق برای کلاس مفاهیم بدون بایاس را توانی از n می داند اما در حقیقت اثبات می شود که این مرز اغراق آمیز نیست.

۷.۳.۳.۲ مفاهیم k-term DNF و k-CNF

همچنین می توان کلاس های مفاهیمی را پیدا کرد که پیچیدگی نمونه ای چند جمله ای دارند اما با این وجود نمی توان آنها را در زمانی چند جمله ای یادگرفت. یکی از مثالهای جالب چنین کلاسهایی، کلاس مفاهیم فرم نرمال فصلی k جمله ای (k-term DNF) [1] است. عبارات k-term DNF به فرم $T_1 \vee T_2 \vee \ldots \vee T_k$ هستند که در آن T_i عطفی از n ویژگی منطقی و نقیض هایشان است. با این فرض که H=C می تون به راحتی نشان داد که |H| حداکثر 3^{nk} خواهد بود (زیرا که k جمله داریم که هرکدام 3^n حالت دارند). توجه دارید که 3^{nk}

[1] k-term disjunctive normal form

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

تخمین بالایی از H است زیرا که در شرایطی که $T_i = T_j$ و T_i کلی تر از T_j است را دوبار مـی شـماریم. بـا ایـن وجـود مـی تـوان از مـرز حداکثری |H| برای پیدا کردن مرز حداکثری پیچیدگی نمونه ای استفاده کرد، با جایگذاری در رابطه ی ۷.۲ خواهیم داشت،

$$m \geq \frac{1}{\varepsilon}\left(nk\ln 3 + \ln\left(\frac{1}{\delta}\right)\right) \qquad (7.6)$$

این رابطه نشان می دهد که پیچیدگی نمونه ای k-term DNF چند جمله ای ای از 1/ε، 1/δ، n و k است. با ایـن وجـود کـه پیچیـدگی نمونه ای از درجه چند جمله ای است، پیچیدگی محاسباتی از درجه چند جمله ای نیست، زیرا می توان نشان داد که این مسئله یادگیری معـادل دیگر مسائل یادگیری است که در زمان چند جمله ای قابل حل نیسـتند (مگـر اینکـه RP=NP). بنـابراین، بـا وجـود اینکـه k-term DNF پیچیدگی چند جمله ای دارد، اما پیچیدگی محاسباتی آن برای یادگیری که در آن H=C از درجه ی چند جمله ای نخواهد بود.

حقیقت جالب در مورد k-term DNF این است که با وجود اینکه این کلاس قابل یادگیری PAC نیست، امـا بـا ایـن حـال کـلاس مفـاهیم بزرگتری وجود دارد که قابل یادگیری PAC است! این از این جهت ممکن است که کلاسهای مفاهیم بزرگتر پیچیدگی محاسباتی چند جمله ای از نمونه ها دارند و پیچیدگی نمونه ای چند جمله ای دارد. این کلاس بزرگتر کلاس نمایش های k-CNF است: عطف عبارات با تعـداد دلخـواه به فرم $T_1 \wedge T_2 \wedge \ldots \wedge T_k$ که در آن T_i فصلی از حداکثر k ویژگی منطقی است. نشان دادن این حکم کـه k-DNF زیـر مجموعـه ی k-CNF است بسیار ساده است زیرا که می توان هر عبارت k-DNF را به سادگی با یک عبارت k-CNF بازنویسی کرد (برعکس این قضـیه بـر قرار نیست). با این وجود که k-CNF نسبت به k-DNF شامل تر است، هم پیچیدگی نمونه ای چند جمله ای و هم پیچیـدگی محاسـباتی چند جمله ای دارد. بنابراین، کلاس مفاهیم k-DNF با الگوریتم یادگیری ای که از H=k-CNF استفاده می کند قابـل یـادگیری PAC اسـت. برای بحث دقیقتر به (Kearns and Vazirani 1994) مراجعه کنید.

۷.٤ پیچیدگی نمونه ای برای فضاهای فرضیه ای بیکران

در بخش بالا نشان دادیم که پیچیدگی نمونه ای برای یادگیری PAC متناسب با لگاریتم انـدازه ی فضـای فرضـیه ای اسـت. بـا وجـود اینکـه رابطه ی ۷.۲ رابطه ی بسیار مفیدی است اما دو اشکال در بیان پیچیدگی نمونه ای بر اسـاس |H| وجـود دارد. ابتـدا اینکـه ممکـن اسـت بـه مرز های ضعیفی ختم گردد (مقدار δ می تواند برای مقادیر بزرگ |H| به شکل قابـل تـوجهی بزرگتـر از یـک باشـد). دوم اینکـه در فضـای فرضیه ای بیکران به طور کلی نمی توان از رابطه ی ۷.۲ استفاده کرد.

در اینجا معیار دیگری از پیچیدگی H به نام بعد H Vapnik-Chervonenkis (به اختصار بعد H VC یا VC(H)) را معرفی خواهیم کـرد. همانطور که در ادامه نیز خواهیم دید، می توان مرز پیچیدگی را با معیار VC(H) به جای |H| بیان کرد. در بسیاری از موارد، مرز پیچیدگی بـر پایه ی VC(H) قوی تر از مرز رابطه ی ۷.۲ خواهد بود. به علاوه این مرز امکان بررسی پیچیدگی نمونـه ای بسـیاری از فضـاهای فرضـیه ای بیکران را نیز فراهم می کند.

www.farabar.net

۷.٤.۱ خرد کردن مجموعه ای از نمونه ها

بعد VC پیچیدگی فضای فرضیه ای H را نه بر اساس |H| و بلکه بر اساس تعداد نمونه های متمایز X که می توانند به کلی با H مشخص شوند بیان می کند.

برای دقیقتر کردن این نمادگذاری، بیایید ابتدا نمادگذاری خرد کردن مجموعه ای از نمونه ها[1] را مشخص کنیم. زیر مجموعه ای از نمونه ها مانند $S \subseteq X$ را در نظر بگیرید. برای مثال، شکل ۷.۳ زیر مجموعه ای از X شامل سه نمونه را نشان می دهد. هر فرضیه ی h در H S را به دو مجموعه تقسیم می کند؛ این دو مجموعه، مجموعه های $\{x \in S|h(x) = 0\}$ و $\{x \in S|h(x) = 0\}$ هستند. با معلوم بودن مجموعه ی S می توان $2^{|S|}$ تقسیم دوتایی مختلفی که اعضای H ممکن بعضی از آنها را نتوانند ایجاد کنند نوشت. زمانی می گوییم H، S را خرد می کند که برای هر یک از تقسیم های دوتایی S را بتوان با فرضیه ای از H نمایش داد.

تعریف. مجموعه ی نمونه های S با فضای فرضیه ای H خرد می شود اگر و فقط اگر برای هر تقسیم دوتایی S فرضیه ای سازگار وجود داشته باشد.

شکل ۷.۳ مجموعه ای از سه نمونه را نشان می دهد که توسط فضای فرضیه ای خرد می شود. توجه دارید که برای هر یک از 2^3 تقسیم دوتایی این سه نمونه فرضیه ای وجود دارد.

توجه دارید که اگر مجموعه ای از نمونه ها توسط یک فضای فرضیه ای خرد نشود، بدین معناست که فرضیه ای (تقسیم دوتایی ای) بر روی نمونه ها وجود دارد که نمی توان آنرا با فضای فرضیه ای نشان داد. قدرت خرد کردن یک فضای فرضیه ای برای مجموعه ای از نمونه ها معیاری از قدرت نمایش این فضای فرضیه ای برای نمایش مفاهیم تعریف شده بر روی این مجموعه از نمونه هاست.

۷.٤.۲ بعد Vapnik-Chervonenkis

قدرت خرد کردن مجموعه ای از نمونه ها رابطه ی نزدیکی با بایاس استقرایی یک فضای فرضیه ای دارد. با توجه به آنچه در فصل ۲ گفته شد، فضای فرضیه ای بدون بایاس فضای فرضیه ای است که می تواند تمامی مفاهیم (تقسیم های دوتایی) قابل تعریف روی فضای نمونه ی را نمایش بدهد. اما اگر H نتواند X را خرد کند، اما در مقابل بتواند زیر مجموعه ی بزرگی از X را خرد کند چه؟ به نظر می رسد اینکه هر قدر زیرمجموعه ی خردشده ی X بزرگتر باشد، H نیز شامل تر خواهد بود. بعد VC ی H به طور دقیقتر معیار زیر است.

تعریف. بعد Vapnik-Chervonenkis، VC(H)، برای فضای فرضیه ای H که بر روی فضای نمونه ای X تعریف شده اندازه ی بزرگترین زیرمجموعه ی کراندار X است که با H خرد می شود. اگر H بتواند هر زیرمجموعه ی دلخواه X را خرد کند خواهیم داشت، VC(H)≡∞.

توجه دارید که برای تمامی H های کراندار داریم $VC(H) \leq \log_2 |H|$. برای درک این رابطه فرض کنید داریم $VC(H) = d$. پس H به 2^d فرضیه ی محض برای خرد کردن d نمونه احتیاج خواهد داشت، پس $2^d \leq |H|$ و $d = VC(H) \leq \log_2 |H|$.

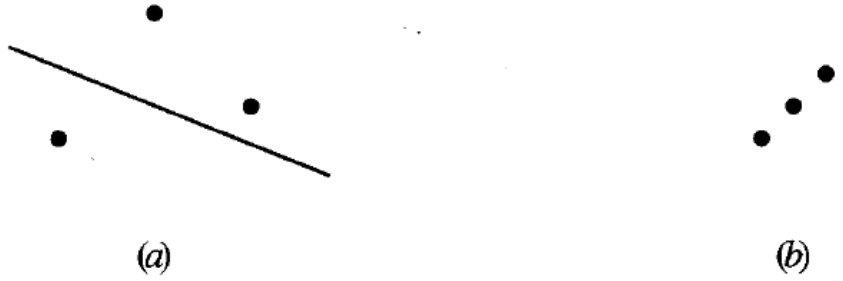
[1] shattering a set of instances

Telegram: @Farabar_BI
www.farabar.net

۷.٤.۲.۱ چندین مثال

برای پیدا کردن درکی از VC(H)، چند نمونه فضای فرضیه ای را در نظر بگیرید. برای شروع، فرض کنید که X مجموعه ی اعداد حقیقی $X=\Re$است (که قد افراد را توصیف می کند) و H مجموعه ی بازه های اعداد حقیقی است. به عبارت دیگر H مجموعه ی فرضیه هایی است که به فرم a<x<b بیان می شوند و a و b نیز اعداد ثابت حقیقی اند. VC(H) در اینجا چیست؟ برای جواب به این سوال، باید بزرگترین زیرمجموعه ی X را پیدا کنیم که با H خرد شود. زیر مجموعه ی دو عضوی خاصی از اعداد حقیقی مثل S={3.1,5.7} را در نظر بگیرید. آیا می توان S را با H خرد کرد؟ بله، برای مثال چهار فرضیه ی (1<x<2)، (1<x<4)، (4<x<7) و (1<x<7) این کار را انجام می دهند. این چهار مجموعه با هم هر یک از تقسیمهای دو عضوی S را، مثل دربر داشتن هیچکدام، یکی و هر دو نمونه، را نمایش می دهند. از آنجایی که مجموعه ای دو عضوی پیدا کردیم که با H خرد شود پس VC(H) حداقل دو خواهد بود. اما آیا مجموعه ای با اندازه ی سه وجود دارد که H آنرا خرد کند؟ مجموعه ی $S = \{x_0, x_1, x_2\}$ با سه نمونه ی دلخواه را در نظر بگیرید. بدون از دست دادن کلیت فرض کنیم که $x_0 < x_1 < x_2$. واضح است که این زیرمجموعه را نمی توان خرد کرد، زیرا که تقسیم دو عضوی ای که x_0 و x_2 را در برگرفته و x_1 را در بر نگیرد را نمی توان با یک فرضیه نشان داد. بنابراین، هیچ زیرمجموعه ی سه عضوی را نمی توان خرد کرد و VC(H)=2. توجه دارید که در اینجا H بیکران و VC(H) کراندار است.

فرض کنید که X مجموعه نقاط روی صفحه ی x-y باشد (شکل ۷.۴). فرض کنیم H تمام سطوح تصمیم گیری خطی بر روی این صفحه است. به عبارت دیگر، H فضای فرضیه ای متناسب با یک واحد پرسپترون با دو ورودی است (برای بحث کلی تر به فصل ۴ مراجعه کنید). بعد VC ی H چیست؟ درک اینکه هر دو نقطه در فضا را می توان با فضای فرضیه ای H خرد کرد بسیار آسان است، زیرا که چهار خط پیدا می شود که هیچکدام، یکی و یا هردو نمونه را در بر بگیرند؟ اما در مورد مجموعه های سه نقطه ای چه؟ تا زمانی که نقاط بر روی یک خط نیستند، 2^3 خط پیدا خواهیم کرد که مجموعه را خرد کند. البته مجموعه ی سه نقطه ی روی یک خط را نمی توان خرد کرد (به همان دلیل که نمی توانستیم در مثال قبل مجموعه ی سه نقطه ای از روی خط حقیقی را خرد کنیم). در چنین شرایطی VC(H) چند است، دو یا سه؟ حداقل سه است. تعریف بعد VC می گوید که اگر بتوانیم مجموعه ای d عضوی را پیدا کنیم که بتوان آنرا با H خرد کرد داریم که $VC(H) \geq d$. برای نشان دادن اینکه $VC(H) < d$ باید نشان دهیم که مجموعه ی d عضوی را نمی توان با H خرد کرد. در این مثال، هیچ مجموعه ی چهار عضوی را نمی توان خرد کرد پس VC(H)=3. به طور کلی می توان نشان داد که بعد VC ی سطوح تصمیم خطی در فضای r بعدی (بعد VC پرسپترونی با r ورودی) r+1 است.



شکل ۷.۴ بعد VC ی سطوح تصمیم خطی در صفحه ی x-y ۳ است.
(a) مجموعه ای از سه نقطه که با سطوح تصمیم خطی خرد می شود. (b) مجموعه ای از سه نقطه که نمی توان آنرا با سطوح تصمیم خطی خرد کرد.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

به عنوان مثال آخر، فرض کنید که هر نمونه از X با عطفی از سه عبارت منطقی، و هر فرضیه ی H عطف حداکثر سه عبارت منطقی نمونه ها باشد. VC(H) چیست؟ می توان نشان داد که این مقدار حداقل ۳ است. هر یک از نمونه ها را با رشته ای سه بیتی متناسب با عبارات l_1، l_2 و l_3 نشان می دهیم. مجموعه ی سه نمونه ای زیر را در نظر بگیرید:

$instance_1: 100$

$instance_2: 010$

$instance_3: 001$

این مجموعه ی سه عضوی از نمونه ها را می توان با H خرد کرد، زیرا که برای هر تقسیم دوتایی می توان به فرم زیر فرضیه ای ساخت: اگر فرضیه ای نمونه ی $instance_i$ را در بر نمی گیرد $\neg l_i$ را به فرضیه اضافه کن. برای مثال فرضیه ای را در نظر بگیرید که $instance_2$ را دربرگرفته اما $instance_1$ و $instance_3$ را دربر نمی گیرد. از فرضیه ی $\neg l_1 \wedge \neg l_2$ برای این حالت استفاده خواهیم کرد. این بحث را می توان به سادگی از ۳ ویژگی به n ویژگی تعمیم داد. در واقع، VC(H) در این حالت دقیقا n است، اما نشان دادن این کار کمی سخت تر است زیرا باید نشان دهیم که مجموعه ی n+1 عضوی ای وجود ندارد که با H خرد شود.

۷.٤.۳ پیچیدگی نمونه ای و بعد VC

در قسمتهای قبلی سوال "چه تعداد نمونه ی تصادفی برای تخمین یکی از فرضیه های C کافی است؟" (چه تعداد نمونه ی آموزشی برای اینکه با احتمال (1-δ) فضای ویژه ε-exhaust باشد؟) را بررسی کردیم. با استفاده از VC(H) به عنوان معیاری برای پیچیدگی H چگونه می توان جوابی دیگر برای این سوال پیدا کرد، مشابه آنچه پیشتر با مرز رابطه ی ۷.۲ بیان کردیم. این مرز جدید به فرم زیر است (به (Blumer et al. 1989) مراجعه کنید)

$$m \geq \frac{1}{\varepsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8VC(H)\log_2\left(\frac{13}{\varepsilon}\right)\right) \qquad (7.7)$$

توجه دارید که مشابه رابطه ی ۷.۲ تعداد نمونه های لازم m با لگاریتم 1/δ متناسب است. اما به جای رابطه ی خطی با لگاریتم برابر رابطه خطی با 1/ε متناسب است. قابل توجه است که، جمله ی ln|H| که در مرز قبلی بود با معیار جایگزین پیچیدگی فضای فرضیه ای، VC(H)، جایگزین شده است. (توجه دارید که $VC(H) \leq \log_2 |H|$).

رابطه ی ۷.۷ کران بالایی برای تعداد نمونه های آموزشی لازم برای اینکه هر فرضیه ی C را به طور PAC با ε و δ دلخواه یاد بگیریم را معلوم می کند. پیدا کردن کران پایین برای این تعداد با استفاده از قضیه ی زیر امکان پذیر است (به (Ehrenfeucht et al. 1989) مراجعه کنید).

قضیه ی ۷.۳ کران پایین پیچیدگی نمونه ای. کلاس مفاهیم دلخواه C که برای آن داریم $VC(C) \geq 2$، یادگیر دلخواه L و $0 < \varepsilon < \frac{1}{8}$ و $0 < \delta < \frac{1}{100}$ را در نظر بگیرید. توزیعی مثل $\mathcal{D}$ و مفهوم هدفی مثل c در C وجود دارد که اگر L کمتر از

$$max\left[\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right), \frac{VC(C)-1}{32\varepsilon}\right]$$

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

تعداد نمونه را مشاهده کرده باشد، با احتمال حداقل δ، L فرضیه ای را خروجی می دهد که $error_{\mathcal{D}}(h) > \varepsilon$.

این قضیه نشان می دهد که اگر تعداد نمونه های آموزشی بسیار کم باشد، هیچ یادگیری نمی تواند تمامی مفاهیم هدف C غیربدیهی را به طور PAC یاد بگیرد. بنابراین، این قضیه کران پایینی بر روی تعداد نمونه های آموزشی برای یاگیری موفق را ارائه می کند، این مرز کامل کننده ی کران بالای ذکر شده برای تعداد کافی نمونه های آموزشی است. توجه دارید که این کران پایین با پیچیدگی کلاس مفاهیم C بیان می شود، در حالی که کران بالا با H تعیین می شد. (چرا؟)

این کران پایین نشان می دهد که کران بالای نامساوی ۷.۷ به اندازه ی کافی محکم است. هر دو کران با $1/\delta$ رابطه ی لگاریتمی و با VC(H) رابطه ی خطی دارند. تنها تفاوت های باقی مانده در این دو کران وابستگی کران بالا به $\log(1/\varepsilon)$ است.

۷.٤.٤ بعد VC برای شبکه های عصبی

با درنظر داشتن بحث شبکه های عصبی مصنوعی از فصل ۴، تعیین بعد VC شبکه ای از واحدهای مرتبط، مثل شبکه های عصبی تک سویه که توسط فرایند backpropagation آموزش داده می شوند، جالب خواهد بود. این بخش نتیجه ی کلی ای از محاسبه ی بعد VC شبکه های بدون دور را برا اساس ساختار شبکه و بعد VC خود واحد ها را ارائه می کند. این بعد VC را می توان برای محدود کردن نمونه های آموزشی لازم برای یادگیری تقریبا درست شبکه ی تک سویه برای مقادیر دلخواه ε و δ به کاربرد. می توانید در اولین مطالعه ی کتاب این بخش را بدون از دست دادن پیوستگی مطلب نخوانید.

شبکه ی G متشکل از واحدها با گراف بدون دور را در نظر بگیرید. یک گراف جهت دار[1] بدون دور[2] گرافی است که یالهایش جهت دارند (واحد ها ورودی و خروجی هستند) و دور ندارد. گراف لایه ای[3] گرافی است که گره هایش را بتوان به صورتی تقسیم بندی کرد که تمامی یالهای جهت دار خروجی از گره های لایه l به گره های لایه ی l+1 بروند. گراف شبکه ی عصبی تک سویه در فصل ۴، مثالی از چنین گرافهای جهت دار لایه ای بدون دور است.

ثابت می شود که می توان بعد VC چنین شبکه هایی را بر اساس ساختارشان و بعد VC واحد های اولیه ی سازندیشان محدود کرد. برای فرموله کردن این حقیقت، باید ابتدا چندین عبارت دیگر را تعریف کنیم. بیایید فرض کنیم که n تعداد ورودی های شبکه ی G است و این شبکه تنها یک خروجی دارد. فرض کنیم که واحد های داخلی G، N_i (هر واحدی که ورودی نباشد) حداکثر r ورودی داشته و از تابعی منطقی مقدار $c_i: \mathfrak{R}^r \to \{0,1\}$ از کلاس توابع C استفاده کند. برای مثال اگر واحد های داخلی پرسپترون باشند C کلاس توابع خطی مقدار آستانه ای تعریف شده بر روی $\mathfrak{R}^r$ خواهد بود.

حال می توانیم ترکیب G[4] ی C را به عنوان کلاس تمام توابعی که شبکه G می تواند به کار ببرد با این فرض که هر واحد شبکه ی G یکی از توابع کلاس C را مورد استفاده قرار دهد تعریف کرد. به طور خلاصه، ترکیب G ی C فضای فرضیه ای است که توسط شبکه ی G قابل نمایش است.

---

[1] directed
[2] acyclic
[3] layered graph
[4] G-composition

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

قضیه ی زیر بعد VC ترکیب G ی C را بر اساس بعد VC ی C و ساختار G محدود می کند.

قضیه ی ۷.۴. بعد VC شبکه های جهت دار لایه ای بدون دور. (برای اطلاعات بیشتر به (Kearns and Vazirani 1994) مراجعه کنید). فرض کنید G گراف جهت دار لایه ای بدون دوری با n گره ورودی و $s \geq 2$ گره داخلی با حداکثر r ورودی است و C نیز کلاس مفاهیم روی $\mathfrak{R}^r$ با بعد VC، d است که متناسب با دسته توابع قابل توصیف توسط هر یک از s گره داخلی است. اگر C_G ترکیب G ی C متناسب با دسته توابع قابل توصیف با G باشد، داریم که $VC(C_G) \leq 2ds\log(es)$ که در این رابطه e پایه ی لگاریتم طبیعی (عدد نپر) است.

توجه دارید که این مرز بعد VC شبکه ی G رابطه ی خطی با بعد VC ی d تک واحد هایش و رابطه ی لگاریتمی با s، تعداد واحدهای آستانه ی شبکه دارد.

شبکه های لایه ای بدون دوری را در نظر بگیرید که از گره های پرسپترون تشکیل یافته اند. با توجه به آنچه در فصل ۴ گفته شد، پرسپترون r ورودی از سطوح تصمیم خطی برای نمایش توابع منطقی بر روی $\mathfrak{R}^n$ استفاده می کند. همانطور که در بخش ۷.۴.۲.۱ نیز گفته شد، بعد VC ی سطوح تصمیم خطی ی $\mathfrak{R}^n$، r+1 است. بنابراین، یک تک پرسپترون با r ورودی بعد VC ی r+1 خواهد داشت. از این حقیقت می توان به همراه قضیه ی بالا برای محدود کردن بعد VC شبکه ای لایه ای شامل s پرسپترون هر کدام با r ورودی استفاده کرد،

$$\mathrm{VC}\left(\mathrm{C}_{\mathrm{G}}^{\mathrm{perceprtons}}\right) \leq 2(\mathrm{r}+1)\mathrm{s}\log(\mathrm{es})$$

حال می توان تعداد m نمونه ی آموزشی کافی برای یادگیری (با احتمال حداقل (1-δ)) هر مفهوم هدف $\mathrm{C}_{\mathrm{G}}^{\mathrm{perceprtons}}$ را با خطای ε مشخص کرد. با جایگزاری رابطه ی بالا برای VC شبکه در رابطه ی ۷.۷ خواهیم داشت،

$$\begin{aligned} m &\geq \frac{1}{\varepsilon}\left(4\log\left(\frac{2}{\delta}\right) + 8\mathrm{VC}(\mathrm{H})\log\left(\frac{13}{\delta}\right)\right) \\ &\geq \frac{1}{\varepsilon}\left(4\log\left(\frac{2}{\delta}\right) + 16(\mathrm{r}+1)\mathrm{s}\log(\mathrm{es})\log\left(\frac{13}{\delta}\right)\right) \end{aligned} \tag{7.8}$$

همانطور که با این مثال شبکه ی پرسپترن نشاند داده شد، قضیه ی بالا از این نظر جالب است که متدی کلی برای محدود کردن بعد VC ی شبکه ای بدون دور و لایه ای از واحد ها را بر اساس ساختار شبکه و بعد VC تک واحد سازنده محدود می کنیم. متاسفانه نتایج بالا مستقیما در مورد شبکه هایی که با Backpropagation آموزش داده می شوند صادق نیست، به دو دلیل. اول اینکه این نتایج برای شبکه های پرسپترون، و نه واحدهای سیگموید که در backpropagation مورد استفاده است، نتیجه گیری شده است. با این وجود، توجه دارید که بعد VC ی واحدهای سیگموید حداقل به اندازه ی بعد VC ی واحدهای پرسپترون است، زیرا که واحد سیگموید می تواند پرسپترون را تا حد دلخواه با افزایش وزنها تخمین بزند. بنابراین، مرز بالا برای m حداقل مرز ممکن برای شبکه های بدون دور واحدهای سیگموید است. مشکل دوم تعمیم نتیجه ی بالا این است که Backpropagation از شبکه ای با وزنهای غیر صفر کار خود را شروع کرده و با تغییر وزنها به فرضیه ی قابل قبول می رسد. بنابراین، Backpropagation با معیار توقف cross-validation بایاسی استقرایی با ترجیح شبکه هایی با وزنهای کوچکتر دارد. این بایاس استقرایی که به طور موثری VC را کاهش می دهد در بررسی بالا در نظر گرفته نشده است.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۷.۵ مدل یادگیری مرز خطا

با وجود اینکه ما بیشتر بر روی مدل یادگیری PAC تمرکز کردیم، تئوری یادگیری محاسباتی تعریف مسئله های دیگر و دیگر سوالات را در نظر دربر می گیرد. تعریف مسئله های یادگیری مختلفی که مورد مطالعه قرار گرفته است در نحوه ی ایجاد نمونه های یادگیری (مشاهده ی سوم شخص[1] نمونه های تصادفی، انتخاب آزمایش توسط یادگیر)، نویز داده ها (با خطا یا بدون خطا)، تعریف موفق (مفهوم هدف باید دقیقا یادگرفته شود یا اینکه تقریبا یا با احتمال خاصی یادگرفته شود)، فرضهای یادگیر (شامل توزیع نمونه ای و اینکه $C \subseteq H$) و معیاری که با آن یادگیر ارزیابی می شود (تعداد نمونه های آموزشی، تعداد اشتباه ها، زمان کل یادگیری) متفاوت اند.

در این بخش به مدل یادگیری مرز خطا، که در آن یادگیر با تعداد اشتباه هایش قبل از همگرایی به فرضیه ی درست ارزیابی می شود خواهیم پرداخت. مشابه تعریف مسئله ی PAC، فرض می کنیم یادگیر سری ای از نمونه های آموزشی را دریافت می کند. با این وجود، در اینجا می خواهیم یادگیر قبل از دریافت هر نمونه ی x مقدار تابع هدف c(x) را (قبل از معلوم شدن مقدار درست هدف توسط آموزش دهنده) پیشبینی کند. سوال مطرح این است که "یادگیر قبل از یادگیری مفهوم هدف چه تعداد پیشبینی اشتباه خواهد کرد؟" اهمیت این سوال در کاربرد عملی است، زیرا که یادگیری باید زمانی که سیستم درحال استفاده واقعی است انجام شود، نه در مرحله ی آموزشی مجزا. برای مثال، اگر سیستم برای یادگیری پیشبینی اینکه چه پرداخت های credit cart ی باید ثبت شود و چه پرداخت هایی تقلبی هستند بر اساس اطلاعاتی که حین استفاده از سیستم جمع آوری می کند طراحی می شود، بنابراین علاقه خواهیم داشته که تعداد اشتباهات قبل از همگرایی به تابع هدف مینیمم شود. در اینجا تعداد کل اشتباهات می تواند اهمیت بیشتری نسبت به تعداد کل نمونه های آموزشی داشته باشد.

این مسئله ی یادگیری مرز خطا را می توان در شرایط خاص مختلفی مورد مطالعه قرار داد. برای مثال ممکن است تعداد اشتباهات قبل از یادگیری PAC تابع هدف را بشماریم. اما در مثالهای زیر ما تعداد اشتباهها قبل از اینکه یادگیر مفهوم هدف را دقیق یاد بگیرید را در نظر می گیریم. یادگیری مفهوم هدف به طور دقیق بدین معناست که به فرضیه ای میل کنیم که $(\forall x)h(x) = c(x)$.

۷.۵.۱ مرز خطا ی الگوریتم Find-S

برای تصور، دوباره فضای فرضیه ای H عطف n عبارت منطقی $l_1 \dots l_n$ و نقایضشان را در نظر بگیرید (برای مثال Rich∧¬Handsome). الگورتیم Find-S که خاصترین فرضیه سازگار با نمونه های آموزشی را محاسبه می کرد را از فصل ۲ به یاد بیاورید. یکی از سرراست ترین پیاده سازی الگوریتم Find-S برای فضای فرضیه ای H در زیر آمده:

Find-S:

فرضیه ی h را با خاص ترین فرضیه $l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \dots l_n \wedge \neg l_n$ مقدار دهی اولیه کن.
برای هر نمونه ی مثبت x
هر عبارت h را که توسط x راضی نمی شد را حذف کن.
فرضیه ی h را خروجی بده.

[1] passive

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

Find-S به طور حدی به فرضیه ای میل می کند که خطایی نخواهد داشت، به شرطی که C⊆H باشد و داده های آموزشی نیز بدون خطا باشند. Find-S با خاصترین فرضیه (فرضیه ای که تمامی نمونه ها را منفی دسته بندی می کند) شروع می کند، سپس به صورت پلکانی این فرضیه را در مواقع لازم برای پوشاندن نمونه های آموزشی مثبت کلی تر می کند. برای نمایش فرضیه ای استفاده شده در اینجا، این کلی سازی با حذف عبارات راضی نشده خواهد بود.

آیا می توان اثبات کرد که تعداد اشتباهات Find-S قبل از یادگیری دقیق مفهوم هدف c کمتر از تعداد خاصی است؟ جواب بلی است. برای درک این، توجه کنید که اگر داشته باشیم c∈H، آنگاه هیچگاه Find-S نمونه ی منفی ای را مثبت دسته بندی نخواهد کرد. دلیل این است که فرضیه ی فعلی h همیشه حداقل به اندازه ی مفهوم هدف c خاص است. بنابراین، برای محاسبه ی تعداد اشتباهات، فقط باید اشتباهاتی را که نمونه ی مثبت، منفی دسته بندی می شود را بشماریم. این گونه اشتباهات قبل از یادگیری کامل c چند بار اتفاق خواهند افتاد؟ اولین نمونه ی مثبت ارائه شده به Find-S را در نظر بگیرید. یادگیر در دسته بندی این نمونه دقیقا یک اشتباه انجام خواهد داد، زیرا که فرضیه ی اولیه ی یادگیر تمامی نمونه ها را منفی دسته بندی می کند. با این وجود، نتیجه این خواهد بود که نصف 2n عبارت فرضیه ی اولیه حذف خواهد شد و n عبارت باقی خواهد ماند. برای هر نمونه ی مثبت بعدی که به اشتباه منفی دسته بندی می شود حداقل یکی از این n عبارت از فرضیه ی h حذف خواهد شد. بنابراین، تعداد کل اشتباهات حداکثر n+1 خواهد بود. این تعداد اشتباه در بدترین حالت رخ خواهد داد، یعنی از یادگیر بخواهیم کلی ترین مفهوم هدف (∀x)c(x)=1 را یاد بگیرد بدترین سری نمونه ها، یعنی نمونه هایی که در هر بار اشتباه فقط یک عبارت را حذف می کنند به یادگیر داده شود.

۷.۵.۲ مرز خطا ی الگوریتم Halving

به عنوان مثال دوم، الگوریتمی را در نظر بگیرید که با نگه داشتن توصیفی از فضای ویژه یاد می گیرد، و پلکانی در این فضای ویژه با برخورد با نمونه های جدید تجدید نظر می کند. الگوریتم های Candidate-Elimination و List-Then-Eliminate در فصل ۲ چنین الگوریتم هایی هستند. در این بخش مرز بدترین حالت ممکن[1] را روی تعداد اشتباههایی که چنین یادگیری انجام می دهد را برای فضای فرضیه ای محدود H، دوباره با فرض اینکه تابع هدف را باید دقیقا یادبگیریم، محاسبه می کنیم.

برای بررسی تعداد اشتباهات حین یادگیری، ابتدا بادید تعیین کنیم که یادگیر دقیقا چگونه دسته بندی نمونه ی جدید را پیشبینی می کند. بیایید فرض کنیم که این پیشبینی با رای گیری در بین فرضیه های فضای ویژه فعلی انجام می گیرد. اگر اکثر فرضیه های فضای ویژه نمونه ی جدید را مثبت دسته بندی کنند، بنابراین این پیشبینی، پیشبینی یادگیر نیز خواهد بود. در غیر این صورت پیشبینی یادگیر منفی خواهد بود.

این ترکیب یادگیری فضای ویژه، به همراه رای اکثریت برای پیشبینی های بعدی، گاهی الگوریتم Halving نامیده می شود. ماکزیمم تعداد اشتباهات الگوریتم Halving برای H محدود دلخواه قبل از یادگیری مفهوم هدف چیست؟ توجه دارید که یادگیری "دقیق"[2] مفهوم هدف، متناسب با رسیدن به حالتی است که فضای ویژه فقط شامل یک فرضیه شود. (مشابه معمول، فرض می کنیم که مفهوم هدف c در H وجود دارد).

[1] wose-case bound
[2] exact

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

برای بدست آوردن مرز خطا، توجه داشته باشید که الگوریتم Halving فقط زمانی اشتباه می کند که اکثریت فضای ویژه اش نمونه ی جدید را اشتباه دسته بندی کنند. در چنین شرایطی، هنگامی که دسته بندی جدید برای یادگیر آشکار می شود، اندازه ی فضای ویژه حداقل به نصف اندازه ی فعلی اش کاهش می یابد (فقط فرضیه هایی که اقلیت بودند باقی می مانند). با معلوم بودن اینکه با هر اشتباه اندازه ی فضای ویژه حداقل به نصف کاهش می یابد و با دانستن اینکه فضای ویژه ی اولیه فقط |H| عضو داشته، حداکثر اشتباهات ممکن قبل از اینکه فضای ویژه فقط یک عضو داشته باشد $\log_2 |H|$ است. در واقع می توان نشان داد که مقدار این مرز $\lfloor \log_2 |H| \rfloor$ است. برای مثال، فرض کنید که |H|=7 است. اولین اشتباه اندازه ی |H| را به ۳ و دومین اشتباه اندازه ی آنرا به ۱ کاهش خواهد داد.

توجه دارید که مرز $\lfloor \log_2 |H| \rfloor$ مرز بدترین حالت است، و ممکن است الگوریتم Halving بدون هیچ اشتباهی مفهوم هدف را یادبگیرد. دلیل این حقیقت این است که حتی زمانی که رای اکثریت درست است، الگوریتم فرضیه های اشتباه، اقلیت، را حذف خواهد کرد. اگر چنین اتفاقی پیاپی در طول سری آموزش رخ بدهد، بنابراین ممکن است بدون هیچ اشتباهی فضای ویژه به یک عضو کاهش داده شود.

یکی از تغییرات جالب الگوریتم Halving قائل شدن وزن برای رای فرضیه هاست. فصل ۶ دسته بندی کننده ی بهینه ی بیز، که از رای گیری وزن دار میان فرضیه ها استفاده می کند را معرفی می کند. در دسته بندی کننده ی بهینه ی بیز، وزن نسبت داده شده به هر فرضیه احتمال ثانویه ی تخمینی توصیف مفهوم هدف به شرط مشاهده ی داده های آموزشی است. در ادامه ی این بخش الگوریتمی متفاوت بر پایه ی رای گیری وزن دار دیگری را به نام Weighted-Majority معرفی خواهیم کرد.

۷.۵.۳ مرز های خطای بهینه

بررسی بالا مرز بدترین حالت اشتباه را برای دو الگوریتم خاص، Find-S و Candidate-Elimination، تعیین کرد. تعیین اینکه مرز بهینه ی خطا برای کلاس مفاهیم دلخواه C با فرض اینکه H=C جالب خواهد بود. منظور از مرز خطای بهینه، کمترین مرز خطای بدترین حالت برای تمامی الگوریتم های یادگیری ممکن است. به عبارت دقیقتر، برای یادگیری الگوریتم A و مفهوم هدف c $M_A(c)$ را ماکزیمم خطای A در تمامی سری های ممکن نمونه های آموزشی برای یادگیری دقیق c تعریف می کنیم. حال برای هر کلاس فرضیه ای غیر تهی تعریف می کنیم که $M_A(C) \equiv \max_{c \in C} M_A(c)$. توجه دارید که دربالا نشان دادیم که $M_{Find-S}(C) = n + 1$ با این فرض که C کلاس مفاهیم عطف حداکثر n ویژگی منطقی باشد. همچنین نشان دادیم که $M_{Halving}(C) \leq \log_2(|C|)$ برای تمامی کلاسهای مفهوم C است.

مزر خطای بهینه را برای کلاس مفاهیم C به فرم زیر تعریف می کنیم.

تعریف. اگر C کلاس غیرتهی دلخواهی باشد، مرز بهینه ی خطای C، که با Opt(C) نمایش داده می شود، مینیمم روی تمام الگوریتم های ممکن A $M_A(C)$ است.

$$Opt(C) \equiv \min_{A \in learnig\ algorithms} M_A(C)$$

به طور غیر رسمی، این تعریف Opt(C) را تعداد اشتباهات سخت ترین C با استفاه از سخت ترین سری نمونه های آموزشی برای بهترین الگوریتم یادگیری تعریف می کند. (Littlestone 1987) نشان می دهد که برای هر کلاس مفهوم C، رابطه ای جالب میان مرز خطای بهینه ی C و مرز خطای الگوریتم Halving و بعد VC ی C وجود دارد،

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

$$\mathrm{VC} \le \mathrm{Opt(C)} \le \mathrm{M_{Halving}(C)} \le \log_2(|C|)$$

علاوه براین کلاسهای مفاهیمی وجود دارد که این چهار کمیت برای آنها دقیقا مساوی است. یکی از چنین کلاس های مفاهیم کلاس مجموعه ی توانی $\mathrm{C_p}$ برای مجموعه ی محدود X است. در چنین شرایطی $\mathrm{VC}(\mathrm{C_p}) = |X| = \log_2(|\mathrm{C_p}|)$، بنابراین تمامی چهار کمیت بالا برابرند. (Littlestone 1987) نمونه هایی از دیگر کلاسهای مفاهیمی که VC(C) مطلقا کمتر از Opt(C) و Opt(C) مطلقا از $\mathrm{M_{Halving}(C)}$ کمتر است ارائه می کند.

۷.۵.٤ الگوریتم Weighted-Majority

در این قسمت تعمیمی از الگوریتم Halving را به نام الگوریتم Weighted-Majority بررسی می کنیم. الگوریتم Weighted-Majority پیشبینی ها را بر اساس رای گیری وزن داری از استخری از الگوریتم های پیشبینی[1] انجام می دهد و با تغییر این وزن ها یاد می گیرد. این الگوریتم های پیشبینی را می توان فرضیه های متفاوت H در نظر گرفت یا در مقابل می تواند از الگوریتم های یادگیری مختلفی استفاده کرد. در کل، تنها چیزی که لازم داریم الگوریتمی پیشبینی است، که مقدار تابع هدف را برای نمونه پیشبینی کند. یکی از خواص جالب الگوریتم Weighted-Majority قابلیت سازگاری آن با داده های آموزشی غیر سازگار است. زیرا که این الگوریتم فرضیه های ناسازگار با تعدادی نمونه را حذف نمی کند و فقط وزن مربوطه را کاهش می دهد. خاصیت دوم جالب این الگوریتم است که مرز تعداد خطای این الگوریتم وابسته به مرز تعداد خطای بهترین الگوریتم استخر الگوریتم های پیشبینی اش است.

الگورتیم Weighted-Majority با مقدار دهی اولیه ی ۱ به تمامی الگوریتم های پیشبینی شروع می شود و سپس نمونه های آموزشی را دریافت می کند. هر بار که الگوریتم پیشبینی ای نمونه ی آموزشی ای را اشتباه دسته بندی می کند وزن مربوطه اش با ضریب $0 \le \beta < 1$ کاهش می یابد. تعریف دقیق الگوریتم Weighted-Majority در جدول ۷.۱ آمده است.

---

a_i پیشبینی iامین الگورتیم استخر الگوریتم های A است. w_i وزن مربوطه به a_i است.

برای تمامی i ها $w_i \leftarrow 1$
برای هر نمونه ی آموزشی $< x, c(x) >$
q_0 و q_1 را ۰ مقدار دهی اولیه کن
برای هر الگوریتم پیشبینی a_i
اگر $a_i(x) = 0$ آنگاه $q_0 \leftarrow q_0 + w_i$
اگر $a_i(x) = 1$ آنگاه $q_1 \leftarrow q_1 + w_i$
اگر $q_1 > q_0$ آنگاه پیشبینی کن c(x)=1
اگر $q_0 > q_1$ آنگاه پیشبینی کن c(x)=0
اگر $q_0 = q_1$ آنگاه یکی از دو مقدار ۰ یا ۱ را به تصادف برای c(x) پیشبینی کن
برای هر الگوریتم پیشبینی a_i در A
اگر $a_i(x) \neq c(x)$ انگاه $w_i \leftarrow \beta w_i$

---

[1] pool of prediction algorithms

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

جدول ۷.۱ الگورتیم Weighted-Majority

توجه دارید که اگر β=0 باشد، الگوریتم Weighted-Majority همان الگوریتم Halving خواهد بود. از طرف دیگر اگر مقادیر دیگر β را انتخاب کنیم، دیگر هیچ الگورتیم یادگیری ای به طور کامل حذف نخواهد شد. اگر الگوریتمی نمونه ی آموزشی ای را اشتباه دسته بندی کند، خیلی ساده، در رای با تاثیرگذاری کمتر خواهد داشت.

حال نشان خواهیم داد که مرز تعداد خطای الگوریتم Weighted-Majority را می توان با تعداد خطاهای بهترین الگوریتم پیشبینی استخر پیشبینی اش محدود کرد.

قضیه ی ۷.۵. مرز خطای نسبی Weighted-Majority. اگر D سری ای از نمونه های آموزشی باشد و A نیز مجموعه ی n الگوریتم پیشبینی باشد و k کمترین تعداد خطای تمامی الگوریتم های A برای سری آموزشی D باشد، تعداد اشتباهات الگوریتم Weighted-Majority با β=1/2 حداکثر

$$2.4(k + \log_2 n)$$

خواهد بود.

اثبات. این قضیه را با مقایسه ی وزن نهایی بهترین الگوریتم و مجموع وزنهای دیگر الگوریتم ها اثبات می کنیم. اگر a_j الگوریتمی از A باشد که مرزخطای بهینه ی k را داشته باشد، w_j وزن مربوطه این الگوریتم $\left(\frac{1}{2}\right)^k$ خواهد بود، زیرا که وزن اولیه ی برای هر بار اشتباهد ضربدر $\frac{1}{2}$ شده است. حال مجموع $W = \sum_{i=1}^{n} w_i$ را که مجموع وزنهای متناسب n الگوریتم A است را در نظر بگیرید. در ابتدا W مقدار n را داراست. برای هر اشتباه الگوریتم Weighted-Majority این مقدار حداقل به $\frac{3}{4}$W کاهش می یابد. این بدین دلیل است که اکثریت رای گیری وزن دار الگوریتم ها اشتباه کرده اند و ضریب این اکثریت با ضریب $\frac{1}{2}$ کاهش خواهد یافت. اگر M کل تعداد اشتباهات الگوریتم Weighted-Majority برای سری آموزشی D باشد، بنابراین، مجموع کل وزن W حداکثر $\mathrm{n}\left(\frac{3}{4}\right)^{\mathrm{M}}$ خواهد بود. چون وزن نهایی w_j نمی تواند بیشتر از وزن کل باشد داریم،

$$\left(\frac{1}{2}\right)^k \leq \mathrm{n}\left(\frac{3}{4}\right)^{\mathrm{M}}$$

با بازنویسی عبارات داریم که،

$$\mathrm{M} \leq \frac{\mathrm{k} + \log_2 n}{-\log_2\left(\frac{3}{4}\right)} \leq 2.4(\mathrm{k} + \log_2 n)$$

و قضیه اثبات می شود.

به طور خلاصه، قضیه ی بالا نشان می دهد که تعداد اشتباهات الگوریتم Weighted-Majority هیچگاه بیشتر از ضریب نسبتی از تعداد اشتباهات بهترین عضو استخر به علاوه ی جمله ای که رابطه ی لگاریتمی با اندازه ی استخر دارد نخواهد بود.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

این قضیه در حالت کلی توسط (Littlestone and Warmuth 1991) اثبات شده و نشان داده شده که مرز بالا برای مقدار دلخواه $0\leq\beta<1$ مرز زیر خواهد بود،

$$\frac{k\log_2\left(\frac{1}{\beta}\right)+\log_2 n}{\log_2\frac{2}{1+\beta}}$$

۷.۶ خلاصه و منابع برای مطالعه ی بیشتر

نکات اصلی این فصل شامل موارد زیر می باشد:

مدل تقریبا درست یا PAC، به الگوریتم هایی می پردازد که مفاهیم هدف را از کلاس مفاهیم هدف C، با استفاده از نمونه های آموزشی تصادفی انتخابی با یک توزیع احتمال ثابت اما نامعلوم یاد می گیرد. این مدل یادگیر را ملزم می کند که به احتمال حداقل [1-δ] فرضیه ای را بیاموزد که تقریبا (با خطای ϵ) درست باشد، با این شرط که پیچیدگی محاسباتی و نمونه ای حداکثر به صورت چند جمله ای ای از 1/ϵ، 1/δ، اندازه ی مجموعه ی نمونه ای و اندازه ی مفهوم هدف باشد.

با این تعریف از مدل یادگیری PAC، هر یادگیر با فضای فرضیه ای متناهی H که C⊆H با احتمال (1-δ) فرضیه ای را خروجی خواهد داد که خطای ϵ بر روی مفهوم هدف بعد از m نمونه ی آموزشی تصادفی خواهد داشت، به شرط آنکه

$$m\geq\frac{1}{\epsilon}\left(\ln\left(\frac{1}{\delta}\right)+\ln|H|\right)$$

این شرط مرزی برای تعداد نمونه های کافی برای یادگیر موفق با معیار PAC به ما خواهد داد.

یکی از فرضهای محدود کننده ی مدل PAC این است که یادگیر می داند که کلاس مفاهیم C شامل مفهومی که باید یاد گرفته شود می باشد. در مقابل مدل یادگیری agnostic [1] تعریف کلی تری می کند که یادگیر فرضی در مورد کلاس انتخاب مفهوم هدف ندارد. در مقابل، یادگیر فرضیه ای را از H خروجی خواهد داد که کمترین خطا بر روی نمونه های آموزشی را داشته باشد (در صورت امکان صفر). تحت این شرایط آزادانه تر یادگیری agnostic، یادگیر زمانی اطمینان دارد که با احتمال (1-δ) فرضیه ای با خطای ϵ در میان فرضیه های H را خروجی می دهد که بعد از m نمونه ی تصادفی شرط زیر درست باشد:

$$m\geq\frac{1}{2\epsilon^2}\left(\ln\left(\frac{1}{\delta}\right)+\ln|H|\right)$$

تعداد نمونه های آموزشی لازم برای یادگیری موفق به شدت تحت تاثیر پیچیدگی فضای فرضیه ای یادگیر است. یکی از معیار های مفید پیچیدگی یک فضای فرضیه ای H بعد Vapnik-Chervonenkis آن، VC(H) است. VC(H) اندازه ی بزرگترین زیر مجموعه ی نمونه هاست که می توان انرا با H خرد (به تمام روش های ممکن تقسیم) کرد.

یک مرز جایگزین تعداد نمونه های آموزشی کافی برای یادگیری موفق با مدل PAC با VC(H) بیان می شود:

[1] agnostic learning model

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

$$m \geq \frac{1}{\epsilon}(4\log_2\frac{2}{\delta} + 8VC(H)\log_2\frac{13}{\epsilon})$$

و یک مرز پایین تر نیز:

$$m \geq \max\left[\frac{1}{\epsilon}log\frac{1}{\delta}, \frac{VC(H)-1}{32\epsilon}\right]$$

مدل جایگزین دیگری به نام مدل مرز خطا برای بررسی تعداد نمونه های آموزشی ای که یادگیر قبل از یادگیری کامل مفهوم هدف اشتباه دسته بندی می کند می پردازد. برای مثال الگوریتم Halving حداکثر $\lfloor \log_2 |H| \rfloor$ خطا قبل از یادگیری کامل هر مفهوم هدف از H اشتباه خواهد کرد. برای مفهوم هدف از کلاس C هر الگوریتم در بد ترین حالت Opt(C) اشتباه خواهد داشت که:

$$VC(C) \leq Opt(C) \leq \log_2 |C|$$

الگوریتم Weighted-majority از رای وزندار چندین الگوریتم پیشبینی برای دسته بندی نمونه های جدید استفاده می کند. این الگوریتم وزنهای الگوریتم ها را بر اساس تعداد اشتباه در سری ای از نمونه ها یاد می گیرد. جالب است که بدانید که حداکثر تعداد اشتباه این الگوریتم با بهترین الگوریتم استخر رابطه دارد.

اکثر کارهای اولیه تئوری یادگیری محاسباتی با سوال اینکه آیا یادگیر می تواند مفهوم هدف را با داشتن سری ای غیر بیشمار از داده های آموزشی یاد بگیرد سر و کار دارند. دسته بندی با این محدودیت اولین بار توسط Gold (1967) معرفی شد. تحقیقات کاملی در این ضمینه در Angluin (1992) اورده شده است. Vapnik (1982) مفصلا مسئله ی همگرایی یکنواخت بحث کرده و مدل نزدیک به مدل یادگیری PAC را در Vapnik (1984) معرفی می کند. بحث ε-exhausting فضای ویژه بر اساس شرح Haussler (1988) پایه گذاری شده است. مجموعه ی مفیدی از نتایج تحت مدل PAC را می توان در Blumer et al. (1989) یافت. Kearns and Vazirani (1994) شرح کاملی از تعدادی زیادی از نتایج تئوری یادگیری محاسباتی را ارائه می کند. متون قبلی در این ضمینه شامل Anthony and Biggs (1992) و Natarjan (1991) می شود.

تحقیقات فعلی بر روی تئوری یادگیری محاسباتی به سمت طیف وسیعی از مدلهای یادگیری و الگوریتم های یادگیری میل می کند. بیشتر این تحقیقات را می توان در کنفرانس های سالانه تئوری یادگیری محاسباتی (COLT) پیدا کرد. تعدادی از مجلات یادگیری ماشین (journal machine learning) نیز به این مبحث اختصاص یافته است.

تمارین

۷.۱ پرسپترونی را با دو ورودی در نظر بگیرید. مرزی برای تعداد نمونه های لازم برای اینکه اطمینان داشته باشیم که با احتمال ۹۰٪ حداکثر خطای واقعی ۵٪ را خواهد داشت را بیابید؟ آیا این مرز واقعی به نظر می رسد؟

۷.۲ کلاس مفاهیم C را به فرم $(a \leq x \leq b)\wedge(c \leq y \leq d)$ را که در آن a,b,c,d اعداد صحیح درون بازه ی (۰،۹۹) هستند را در نظر بگیرید. توجه دارید که این کلاس متناسب با مستطیل های حقیقی مقدار در صفحه ی xy است. راهنمایی: مربع محصور در بین (0,0) و (n-1,n-1). تعداد مستطیل هایی با رئوس اعداد صحیح در این بازه $\left(\frac{n(n+1)}{2}\right)^2$ است.

www.farabar.net

(a) حد بالای تعداد نمونه های تصادفی کافی برای اینکه اطمینان داشته باشیم که برای تمامی مفاهیم هدف c از C، هر یادگیر سازگار با H = C با احتمال ۹۵٪ فرضیه ای را خروجی دهد که حداکثر ۰.۱۵ خطا داشته باشد.

(b) حال مستطیلی با مرزهای a, b, c, d را که رئوسش در نقاط حقیقی مقدار است (بجای نقاط صحیح) را در نظر بگیرید. جواب خود را به قسمت اول با این شرایط جدید بدهید.

۷.۳ در این فصل ما عبارتی برای تعداد نمونه ی آموزشی کافی برای اطمینان از اینکه هر فرضیه خطای حقیقی ϵ به اضافه ی خطای مشاهده ی $error_D(h)$ نداشته باشد پیدا کردیم. در کل، از مرزهای هاپفیلد برای پیدا کردن چنین مرزی استفاده شد (رابطه ی ۷.۳). رابطه ی دیگری برای تعداد نمونه های آموزشی کافی برای اینکه اطمینان داشته باشیم که هرف فرضیه خطای حقیقی کمتر از $(1+\gamma)error_D(h)$ را داشته باشد پیدا کنید. می توانید از مرز چرنوف و تعمیم آن برای استخراج چنین نتیجه ای استفاده کنید.

مرز چرنوف (chernoff bounds): فرض کنید که $X_1, \ldots, X_m$، m حاصل مستقل پرتاب سکه (آزمایش برنولی) باشند، با این فرض که احتمال شیر آمدن در هر آزمایش مستقل $\Pr[X_i = 1] = p$ باشد و احتمال خط آمدن $\Pr[X_i = 1] = p - 1$ باشد. اگر $S = X_1 + \cdots + X_m$ مجموع حاصل این پرتابها باشد، مقدار امید S/m مساوی $E\left[\frac{S}{m}\right] = p$ خواهد بود. مرز چرنوف بر احتمال اینکه این مرز با ضریب $0 \le \gamma \le 1$ اختلاف داشته باشد به صورت زیر است:

$$\Pr[S/m > (1+\gamma)p] \le e^{-mp\gamma^2/3}$$

$$\Pr[S/m < (1-\gamma)p] \le e^{-mp\gamma^2/2}$$

۷.۴ مسئله ی یادگیری ای را در نظر بگیرید که $X = \Re$ مجموعه ی اعداد حقیقی و C=H مجموعه ی بازه های روی اعداد حقیقی به فرم $H = \{(a < x < b) | a, b \in \Re\}$ باشد. احتمال اینکه فرضیه ای سازگار با m نمونه از این مفهوم هدف حداقل خطای ϵ داشته باشد چقدر است؟ این سوال را با استفاده از بعد VC جواب دهید. آیا راه حل دیگری بر اساس قوانین اولیه و صرف نظر کردن از بعد VC برای جواب به این سوال وجود دارد؟

۷.۵ فضای نمونه ای X را متناسب با صفحه ی x,y در نظر بگیرید. بعد VC فضاهای فرضیه ای زیر را مشخص کنید:

(a) مجموعه ی تمامی مستطیل های صفحه ی x,y. $H_r = \{((a < x < b \wedge (c < y < d)) | a, b, c, d \in \Re\}$.

(b) تمامی دایره های صفحه ی x,y. تمامی نقاط داخل دایره مثبت دسته بندی خواهند شد.

(c) مثلثهای صفحه ی x,y. نقاط داخل مثلث مثبت دسته بندی خواهند شد.

۷.۶ یادگیر سازگاری با فضای فرضیه ای H_r در مسئله ی ۷.۵ طراحی کنید. مجموعه ای از مفاهیم هدف مستطیلی تصادفی متناسب با مستطیلهای صفحه ایجاد کنید. نمونه های تصادفی مربوطه ی هر یک از این مفاهیم را با توزیع یکنواخت نمونه ها در مستطیل بین (0,0) و (100,100) ایجاد کنید. خطای تعمیم را به عنوان تابعی از تعداد نمونه های تصادفی m رسم کنید. در همان شکل رابطه ی بین ϵ و m را برای δ=.95 نیز رسم کنید. آیا نتایج با تئوری همخوانی دارد؟

Telegram: @Farabar_BI
www.farabar.net

۷.۷ فضای فرضیه ای H_{rd2} را که "درختهای تصمیم متوسط با عمق ۲" است را بر روی n متغیر منطقی در نظر بگیرید. درختهای تصمیم متوسط با عمق ۲ درختهای تصمیمی هستند که (با چهار برگ که تمامی از ریشه فاصله ی ۲ دارند) که در آنها بررسی شده در سمت راست و چپ ریشه یکی هستند. برای مثال شکل زیر نمونه ای از H_{rd2} است.

```
        x3
       /  \
    x1      x1
   / \     / \
  +   -   -   +
```

(a) برحسب n مشخص کنید که اندازه ی این مجموعه ی H_{rd2} چقدر است؟

(b) مرز بالایی برای تعداد نمونه های لازم برای یادگیری با مدل PAC برای یادگیری در H_{rd2} با خطای ϵ و اطمینان δ چقدر است.

(c) الگوریتم Weighted-Majority را برای کلاس H_{rd2} در نظر بگیرید. ابتدا الگوریتم را با تمامی درختها ی درون H_{rd2} با وزن اولیه ی یکسان ۱ شروع می کنیم. هر بار که یک نمونه ی جدید مشاهده می کنیم، آنرا با استفاده از رای وزن دار تمامی فرضیه های H_{rd2} دسته بندی می کنیم. سپس بجای حذف درختهای فرضیه ای که اشتباه کار می کنند فقط وزن تاثیر آن درخت ها نصف می شود. حداکثر تعداد خطاهای را بر حسب n و تعداد خطای بهترین فرضیه ی درون H_{rd2} بیان کنید.

۷.۸ این سوال به ارتباط بین تحلیل PAC در این فصل و بررسی فرضیه در فصل ۵ می پردازد. کار یادگیری ای را در نظر بگیرید که نمونه ها با n متغیر تصادفی (مثلا $x_1 \wedge \bar{x}_2 \wedge x_3 \ldots \bar{x}_n$) توصیف می شوند و توسط توزیع احتمال ثابت و معلوم $\mathcal{D}$ انتخاب می شوند. می دانیم که مفهوم هدف عطفی از متغیر های تصادفی و عکسشان است (مثلا $x_2 \wedge \bar{x}_5$) و الگوریتم یادگیری از این کلاس مفهوم به عنوان فضای فرضیه ای H استفاده می کند. به یک یادگیر سازگار مجموعه ای از ۱۰۰ نمونه انتخابی با $\mathcal{D}$ داده می شود. این یادگیر فرضیه ی h را از H که با تمامی ۱۰۰ نمونه سازگار است خروجی می دهد. (بدین معنا که خطای h بر روی این ۱۰۰ نمونه صفر است)

(a) علاقه داریم که خطای واقعی h را که احتمال دسته بندی اشتباه نمونه های انتخابی با $\mathcal{D}$ است را بیابیم. بر اساس اطلاعات بالا آیا می توانید بازه ای را مشخص کنید که خطای واقعی حداقل با احتمال ۹۵٪ در آن قرار گیرد؟ اگر چنین است، بازه را بیان کرده و به وضوح آنرا توجیه کنید. اگر امکان پذیر نیست مشکل را توضیح دهید.

(b) حال مجموعه نمونه ی جدید ۱۰۰ تایی دیگری به طور مستقل با همان توزیع $\mathcal{D}$ انتخاب می کنید و معلوم می شود که h ۳۰ نمونه از این ۱۰۰ نمونه را اشتباه دسته بندی می کند. آیا می توانید بازه ای ارائه کنید که این خطای واقعی با احتمال ۹۵٪ در آن قرار داشته باشد؟ (برای این قسمت از کارایی فرضیه روی نمونه های آموزش اش صرف نظر کنید.) اگر چنین است، بازه را بیان کرده و به وضوح آنرا توجیه کنید. اگر امکان پذیر نیست مشکل را توضیح دهید.

www.farabar.net

(c) ممکن است کمی عجیب به نظر برسد که با این که فرضیه نمونه های آموزشی را درست دسته بندی کرده اما در دسته بندی نمونه هـای جدید خطای ۳۰٪ داشته است. احتمال چنین اتفاقی در n های بزرگ بیشتر است یا در n های کوچکتر. برای جواب خود توجیه بیاورید.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

ε-exhausted	
آزمایش برنولی	Bernoulli trial
استخری از الگوریتم های پیشبینی	pool of prediction algorithms
پیچیدگی محاسباتی	Computational complexity
پیچیدگی نمونه ای	Sample complexity
ترکیب G	G-composition
توانی	exponential
چارچوب کران خطا	mistake bound framework
چهارچوب	framework
چهارچوب تقریبا درست	probably approximately correct (PAC)
خرد کردن مجموعه ای از نمونه ها	shattering a set of instances
خطای آموزشی	training error
خطای واقعی	True Error
دقیق	Exact
فرم نرمال فصلی k جمله ای	k-term disjunctive normal form (k-term DNF)
قابلیت یادگیری PAC	PAC-learnability
گراف جهت دار بدون دور	directed acyclic
گراف لایه ای	layered graph
مدل یادگیری تقریبا درست	probably approximately correct (PAC)
مرز بدترین حالت ممکن	wose-case bound
مرز خطا	Mistake bound
مرز های Hoeffding	Hoeffding bounds
مرز های اضافی Hoeffding	Hoeffding additive bounds

Telegram: @Farabar_BI
www.farabar.net

فصل هشتم: یادگیری مبتنی بر نمونه‌ها

برخلاف متد های یادگیری‌ای که توصیفی صریح از تابع هدف با استفاده از نمونه های آموزشی موجود ایجاد می‌کنند، یادگیری مبتنی بر نمونه‌ها[1] فقط نمونه‌ها را ذخیره می‌کند و تعمیم روی نمونه‌ها را به زمان دسته بندی نمونه‌ی جدید موکول می‌کند. هر گاه نمونه‌ی جدیدی ارائه شد، رابطه‌ی آن با نمونه‌هایی که قبلاً ذخیره شده‌اند بررسی شده تا مقدار تابع هدف را برای این نمونه‌ی جدید تشخیص دهیم. یادگیری مبتنی بر نمونه‌ها، شامل متد های نزدیک‌ترین همسایه[2] و برازش وزن دار محلی[3] که نمونه‌ها را به عنوان نقاطی در فضای اقلیدسی در نظر می‌گیرند می‌شود. این مبحث همچنین شامل متد های استدلال مبتنی بر شرایط[4] که از نمایش پیچیده‌ی نمادین برای نمایش فرضیه‌ها استفاده می‌کنند می‌شود. گاهی به متد های مبتنی بر نمونه‌ها متد های یادگیری تنبل[5] نیز می‌گویند زیرا که محاسبات را تا دسته بندی نمونه های جدید به تعویق می‌اندازند. مزیت کلیدی این نوع تأخیر یا تنبلی این متدها دسته بندی محلی و به طور جداگانه برای هر یک از نمونه های جدید بجای تخمین تابع هدف برای کل فضای نمونه ای است.

۸.۱ معرفی

متد های یادگیری مبتنی بر نمونه‌ها مثل نزدیک‌ترین همسایه و برازش وزن دار محلی، روش‌های ساده ای برای تخمین توابع هدف حقیقی مقدار و گسسته مقدارند. یادگیری در این الگوریتم‌ها به ذخیره‌ی نمونه های آموزشی در دسترس محدود می‌شود. زمانی که الگوریتم با یک نمونه‌ی جدید برخورد می‌کند، مجموعه ای از نمونه های مشابه از حافظه بازخوانی شده و برای دسته بندی نمونه‌ی جدید مورد استفاده قرار می‌گیرند. یکی از اختلاف‌های کلیدی بین این روش‌ها و متدهایی که در دیگر فصل‌های این کتاب مورد بحث قرار گرفت امکان ایجاد تخمین منحصر به فرد هر نمونه‌ی جدید است. در واقع بسیاری از تکنیک‌ها فقط یک تخمین محلی از تابع هدف که در همسایگی نمونه‌ی جدید است

[1] Instance based learning
[2] Nearest neighbor
[3] Locally weighted regression
[4] Case based reasoning
[5] lazy

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

خروجی می‌دهند و هیچ گاه تخمین جدیدی که برای کار بر روی کل فضای نمونه ای کار کند ایجاد نمی‌کنند. این خاصیت برای توابع هدفی که پیچیده اما به صورت محلی ساده‌اند مزیت دارد.

متد های مبتنی بر نمونه‌ها را نیز می‌توان برای نمایش‌های پیچیده تر نمادین نمونه‌ها به کار برد. در یادگیری مبتنی بر شرایط، نمونه‌ها به فرم نمادین نمایش داده و فرایند تعیین نمونه های "همسایه" نیز بر اساس همین نمایش بیان می‌شود. از استدلال مبتنی بر شرایط در کارهایی نظیر ذخیره و استفاده‌ی دوباره‌ی تحقیق در یک میز کمک، استدلال درباره‌ی شرایط مجاز بر اساس شرایط قبلی، و حل مسائل زمان‌بندی پیچیده با استفاده‌ی از قسمت‌های مرتبط مسائلی حل شده‌ی قبلی به کار برده شده است.

یکی از اشکال‌های روش‌های مبتنی بر نمونه‌ها هزینه‌ی بالای دسته بندی نمونه های جدید است. این اشکال ناشی از این حقیقت است که تقریباً تمامی محاسبات در زمان دسته بندی نمونه های جدید (به جای زمان مواجهه با نمونه های یادگیری) انجام می‌شود. بنابراین، تکنیک‌هایی که برای فهرست[1] کردن بهینه‌ی نمونه‌ها تأثیر قابل توجهی در کم کردن محاسبات لازم در زمان دسته بندی نمونه های جدید دارد. اشکال دوم بسیاری از روش‌های مبتنی بر نمونه‌ها، مخصوصاً روش‌های نزدیک‌ترین همسایه، این است که تمامی ویژگی‌های نمونه در زمان بازیابی نمونه های آموزشی مشابه در نظر گرفته می‌شود. اگر تابع هدف فقط به تعدادی از ویژگی‌های بسیار نمونه‌ها وابسته باشد، نمونه‌هایی که واقعاً مشابه نمونه‌ی جدید هستند با این نمونه فاصله بسیار زیاد داشته باشند.

در قسمت بعد الگوریتم k-Nearest Neighbor را به همراه چندین نسخه‌ی مختلف این الگوریتم پرکاربرد معرفی خواهیم کرد. در زیر قسمت بعد از آن برازش وزن دار محلی را مورد بحث و بررسی خواهد داد و متد یادگیری‌ای را که تخمین‌های موضعی‌ای در مورد تابع هدف می‌زند ارائه می‌کند، این متد را می‌توان تعمیم الگوریتم‌های k-Nearest Neighbor دانست. سپس شبکه‌ی radial basis را که پلی جالب بین یادگیری بر پایه‌ی نمونه‌ها و الگوریتم‌های شبکه های عصبی را بررسی خواهیم کرد. در قسمت بعدی به استدلال مبتنی بر شرایط، روشی مبتنی بر نمونه‌ها که از نمایش نمادین و استنتاج مبتنی بر دانش قبلی کمک می‌گیرد، بحث خواهد شد. این بخش شامل مثالی از کاربرد استدلال مبتنی بر شرایط در یک مسئله‌ی طراحی مهندسی را نیز در بر می‌گیرد. بالاخره، تفاوت‌های پایه ای بین ظرفیت‌هایی متدهای یادگیری تنبل که در این فصل مطرح می‌شوند و متد های کوشا که در دیگر فصل‌های این کتاب مطرح می‌شوند را بررسی خواهیم کرد.

۸.۲ یادگیری k-Nearest Neighbor

ساده‌ترین متد یادگیری مبتنی بر نمونه‌ها، الگوریتم k-Nearest Neighbor است. این الگوریتم فرض می‌کند که تمامی نمونه‌ها نقاطی در فضای n بعدی $\Re^n$ هستند. نزدیک‌ترین همسایه‌ها یک نمونه با استفاده از تعریف استاندارد فاصله اقلیدسی تعریف می‌شوند. به عبارت دقیق‌تر اینکه هر نمونه‌ی دلخواه x با بردار زیر نمایش داده شود،

$$< a_1(x), a_2(x), \dots, a_n(x) >$$

که در آن $a_r(x)$ نشان دهند ی r امین ویژگی نمونه‌ی x است. پس فاصله‌ی بین دو نمونه‌ی x_i و x_j که با $d(x_i, x_j)$ نشان داده می‌شود به صورت زیر تعریف می‌شود:

[1] indexing

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} \left(a_r(x_i) - a_r(x_j)\right)^2}$$

در یادگیری نزدیک‌ترین همسایه[1] ، تابع هدف ممکن است گسسته مقدار یا حقیقی مقدار باشد. بیایید فرض کنیم که تابع هدف گسسته مقدار است، $f: \Re^n \to V$، که در آن V مجموعه‌ی محدود $\{v_1, \dots, v_s\}$ است. الگوریتم k-Nearest Neighber برای تخمین تابع گسسته مقدار در جدول ۸.۱ آورده شده است. همان طور که در جدول نیز نشان داده شده است، مقدار خروجی این الگوریتم $\hat{f}(x_q)$ تخمینی از $f(x_q)$ است که متداول‌ترین مقدار تابع هدف در بین نزدیک‌ترین همسایه های x_q است. اگر مقدار k را ۱ بگیریم الگوریتم *1-nearest neighbor* مقدار $f(x_i)$ را به $\hat{f}(x_q)$ نسبت می‌دهد (x_i نزدیک‌ترین همسایه‌ی x_q است.) برای مقادیر بزرگ‌تر k الگوریتم متداول‌ترین مقدار تابع هدف را میان k نزدیک‌ترین همسایه را به نمونه نسبت می‌دهد.

الگوریتم یادگیری:

- برای هر نمونه‌ی <x,f(x)>، نمونه را به مجموعه‌ی training_examples اضافه کن.

الگوریتم دسته بندی:

- نمونه‌ی x_q را برای دسته بندی بگیر
- $x_1 \dots x_k$، k نمونه‌ی نزدیک‌تر به x_q را از training_examples پیدا کن.

مقدار زیر را خروجی بده

$$\hat{f}(\mathrm{x_q}) \leftarrow \arg\max_{\mathrm{v} \in \mathrm{V}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

در این رابطه $\delta(a,b) = 1$ اگر a=b باشد، و در غیر این صورت $\delta(a,b) = 0$.

جدول ۸.۱ الگوریتم k-Nearest Neighbor برای تخمین تابع گسسته مقدار $\boldsymbol{f: \Re^n \to V}$.

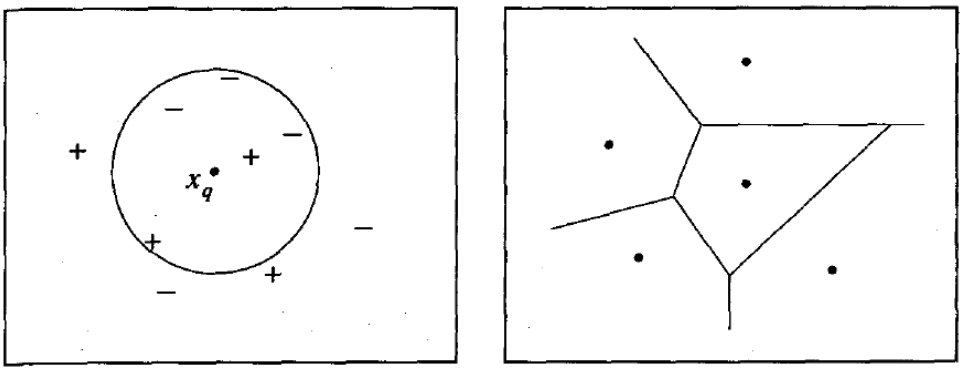
شکل ۸.۱ عملیات الگوریتم k-Nearest Neighbor را در حالتی که نقاط در فضای دو بعدی‌اند و تابع هدف نیز منطقی است نشان می‌دهد. نمونه های مثبت و منفی به ترتیب با "+" و "−" در شکل نشان داده شده‌اند. نقطه‌ی x_q نیز در شکل نشان داده شده است. توجه دارید که در این شکل الگوریتم 1-Nearest Neighbor نمونه‌ی x_q را مثبت دسته بندی خواهد کرد در حالی که 5-Nearest Neighbor آن‌را منفی دسته بندی خواهد کرد.

---

[1] Nearest Neighber

www.farabar.net

Telegram: @Farabar_BI



شکل ۸.۱ k-Nearest Neighbor.

دسته ای از نمونه های مثبت و منفی به همراه یک نمونه‌ی جدید x_q در سمت چپ نشان داده شده است. الگوریتم 1-Nearest Neighbor نمونه‌ی جدید را مثبت دسته بندی می‌کند در حالی که الگوریتم 5-Nearest Neighbor همان نمونه را منفی دسته بندی می‌کند. در سمت راست شکل سطح تصمیم الگوریتم 1-Nearest Neighbor برای یک مجموعه‌ی متوسط از نمونه های آموزشی نشان داده شده است. چند ضلعی‌های نشان داده شده دور هر نمونه، محدوده نقاطی را که نمونه به آن‌ها از نمونه های دیگر نزدیک‌تر است را نشان می‌دهد (در 1-Nearest Neighbor نمونه های داخل هر محدوده بر اساس نمونه‌ی آن محدوده دسته بندی خواهند شد).

طبیعت فضای فرضیه ای H که توسط الگوریتم k-Nearest Neighbor جستجو می‌شود چیست؟ توجه دارید که الگوریتم k-Nearest Neighbor هیچ گاه صریح فرضیه‌ی $\hat{f}$ را که تخمینی از f است معرفی نمی‌کند. این الگوریتم فقط نمونه های جدید را با توجه به نمونه های موجود دسته بندی می‌کند. با این وجود، هنوز می‌توان بررسی کرد که تابع ضمنی تخمین زده شده چیست یا چه دسته بندی با ثابت نگه داشتن نمونه های آموزشی و انتخاب نمونه های مختلف X به الگوریتم بدست می‌آید. شکل 8.1 دسته بندی الگوریتم 1-Nearest Neighbor را برای روی کل فضای نمونه ای نشان می‌دهد. فضای تصمیم نیز چند وجهی‌هایی هستند که هر کدام یک نمونه‌ی آموزشی را در بر می‌گیرند. برای هر نمونه‌ی آموزشی، چند وجهی محدوده ای را که نمونه‌هایش فقط تحت تأثیر یک نمونه‌ی آموزشی خاصند را مشخص می‌کند. نمونه های خارج هر چند وجهی به نمونه‌ی دیگری نزدیک‌ترند. به این نوع نمودار گاهی اوقات نمودار ورونوی[1] مجموعه‌ی نمونه های آموزشی نیز می‌گویند.

الگوریتم k-Nearest Neighbor به سادگی به توابع هدف پیوسته مقدار تعمیم می‌یابد. برای این تعمیم کافی است که به جای متداول‌ترین مقدار تابع هدف از میانگین k نمونه‌ی همسایه استفاده کنیم. به عبارت دقیق‌تر، برای تخمین تابع هدف حقیقی مقدار $f: \mathfrak{R}^n \to \mathfrak{R}$ کافی است که خط آخر الگوریتم بالا را با عبارت زیر جایگزین کنیم.

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{\mathrm{k}} \tag{8.1}$$

[1] Voronoi diagram

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۸.۲.۱ الگوریتم Distance-Weighted Nearest Neighbor

الگوریتم k-Nearest Neighbor را می‌توان با اضافه کردن وزن به k نمونه‌ی همسایه بر اساس فاصله‌ی آن‌ها از نمونه‌ی x_q بهبود بخشید. برای مثال، در الگوریتم جدول ۸.۱ که توابع گسسته مقدار را تخمین می‌زند، می‌توانیم میزان تأثیر رای هر یک از همسایه‌ها را متناسب با عکس فاصله‌شان از x_q تاثیر دهیم. این تغییر را می‌توان با تبدیل خط آخر الگوریتم به عبارت زیر اعمال کرد،

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \tag{8.2}$$

در این رابطه داریم،

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \tag{8.3}$$

در مواقعی که نمونه‌ی x_q با نمونه‌ی آموزشی x_i مساوی است، مقدار $d(x_q, x_i)^2$ صفر خواهد شد، در چنین شرایطی مقدار $f(x_i)$ را به $\hat{f}(x_q)$ نسبت می‌دهیم، اگر چندین نمونه‌ی آموزشی چنین شرایطی را داشتند متداول‌ترین مقدار هدف آن‌ها را انتخاب می‌کنیم.

برای توابع حقیقی مقدار نیز می‌توان به همین ترتیب با عوض کردن سطر آخر الگوریتم به رابطه‌ی زیر بهبود ذکر شده را اعمال کرد،

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \tag{8.4}$$

در این رابطه نیز w_i همان کمیت تعریف شده در رابطه‌ی ۸.۳ است. توجه داشته باشید که مقدار مخرج رابطه‌ی ۸.۴ مقدار کسر را نرمال می‌کند (برای مثال اگر برای تمامی x_i ها داشته باشیم $f(x_i) = c$، این مخرج به ما اطمینان می‌دهد که $\hat{f}(x_q) \leftarrow c$).

توجه دارید که تمامی نسخه های الگوریتم k-Nearest Neighbor که در بالا ذکر شد فقط k نقطه‌ی نزدیک‌تر به نمونه‌ی جدید را در نظر می‌گیرند. با اضافه کردن عامل وزن، واقعاً در نظر گرفتن تأثیر کل نمونه های آموزشی بر x_q ضرری نخواهد داشت، زیرا که نمونه های بسیار دور تأثیر بسیار کمی بر $\hat{f}(x_q)$ خواهند داشت. تنها اثر منفی این تغییر این است که دسته بندی کننده‌ی ما کند تر خواهد شد. متدی که از تمامی نمونه‌ها برای دسته بندی نمونه های جدید استفاده می‌کند را متد جهانی[1] می‌نامند و متدهایی که فقط نزدیک‌ترین همسایه‌ها را در نظر می‌گیرند متد های محلی[2] می‌نامند. با تغییر قانون ۸.۴ به یک متد جهانی الگوریتم یادگیری به متد (Shepard 1968) تبدیل می‌شود.

۸.۲.۲ نکاتی در مورد الگوریتم k-Nearest Neighbor

الگوریتم Distance-weighted k-Nearest Neighbor متد استقرایی بسیار موثری است که در بسیاری از مسائل کاربردی مورد استفاده قرار می‌گیرد. این الگوریتم نسبت به داده های آموزشی خطا دار حساس نیست و زمانی که مجموعه‌ی داده های آموزشی به اندازه‌ی

[1] global method
[2] local method

www.farabar.net

کافی بزرگ باشد بسیار موثر است. توجه داشته باشید که با گرفتن میانگین وزن دار k نمونه‌ی همسایه‌ی نزدیک x_q، الگوریتم می‌تواند اثر داده های خطا دار آموزشی را برطرف کند.

بایاس استقرایی k-Nearest Neighbor چیست؟ بایاس دسته بندی نمونه های جدید با توجه به شکل ۸.۱ به راحتی مشخص می‌شود. بایاس استقرایی این الگوریتم این است که فرض می‌کند که دسته بندی نمونه‌ی x_q بسیار شبیه دسته بندی نمونه های دیگر نزدیک به آن است.

یک از مشکلات کاربردی استفاده‌ی الگوریتم k-Nearest Neighbor این است که فاصله‌ی بین نمونه‌ها با توجه به تمامی ویژگی‌های نمونه‌ها محاسبه می‌شود (در این مثال، تمامی محور های فضای اقلیدسی). این رفتار الگوریتم با متد های دیگر چون سیستم‌های یادگیری قوانین و یادگیری درختی که فقط تعدادی از ویژگی‌ها را در ساخت فرضیه به کار می‌برند متفاوت است. برای درک تأثیر این موضوع، کاربرد k-Nearest Neighbor را در مسئله ای که از ۲۰ ویژگی نمونه‌ها فقط ۲ ویژگی بر دسته بندی تابع هدف تأثیر دارند را در نظر بگیرید. در چنین شرایطی، نمونه‌هایی که این دو ویژگی‌شان نزدیک به هم است ممکن است در فضای ۲۰ بعدی بسیار دور از هم باشند. حاصل اینکه متد ساده‌ی به کار گرفته شده در k-Nearest Neighbor، که کل ۲۰ ویژگی را در نظر می‌گیرد، ما را گمراه خواهد کرد. فاصله‌ی بین همسایه‌ها از تعداد زیادی از ویژگی‌های نامربوط محاسبه می‌شود. این مشکل، زمانی که تعداد ویژگی‌های نامربوط حاضر زیادند رخ می‌دهد بعضی اوقات طلسم بعد[1] نامیده می‌شود. Nearest-neighbor نسبت به این مشکل به شدت حساس است.

یکی از روش‌های جالب مقابله با این مشکل اعمال وزن در محاسبه‌ی فاصله‌ی بین نمونه‌هاست. این تغییر مثل کشیدن و فشرده کردن محور های فضای اقلیدسی است، محور های ویژگی‌های نامربوط فشرده و محور های ویژگی‌های مربوط کشیده می‌شوند. میزان کشش هر محور از طریق یک روش ارزیابی[2] مشخص می‌شود. برای مشخص شدن روش کار، فرض کنید که می‌خواهیم jامین محور را به اندازه‌ی z_j بکشیم (ضرب کنیم)، در این روش $z_1, \dots, z_n$ طوری انتخاب می‌شوند تا خطای واقعی دسته بندی الگوریتم یادگیری مینیمم شود. دوم، توجه داشته باشید که این خطای واقعی را می‌توان با استفاده از ارزیابی بدست آورد. بنابراین، یکی از الگوریتم‌های ممکن انتخاب یک زیرمجموعه‌ی تصادفی از داده های موجود برای نمونه های آموزشی و مشخص کردن $z_1, \dots, z_n$ به صورتی است که خطا را برای بقیه‌ی نمونه‌ها مینیمم کند. با چندین بار تکرار این فرایند تخمین این وزن‌ها دقیق‌تر می‌شوند. این فرایند کشیدن محورها برای بهینه کردن کارایی k-Nearest Neighbor مکانیسمی برای کم کردن تأثیر ویژگی‌های نامربوط ایجاد می‌کند.

روش جایگزین موثر تر دیگر حذف ویژگی‌های نامربوط از فضای نمونه ای است. این کار مشابه صفر کردن مقدار z_i در متد قبلی است. (Moore and Lee 1994) درباره‌ی کارایی متد های cross-validation در انتخاب ویژگی‌های مربوط از مجموعه‌ی ویژگی‌های موجود برای الگوریتم k-Nearest Neighbor را بررسی کرده‌اند. در کل، آن‌ها متدهای cross-validation و leave-one-out را که در آن مجموعه‌ی m تایی از نمونه های آموزشی متناوباً به مجموعه های m-1 تایی از نمونه های آموزشی و مجموعه‌ی تست یک عضوی در تمام حالات ممکن بررسی می‌شود. این روش leave-one-out به راحتی در k-Nearest Neighbor به کار برده می‌شود، زیرا که هر بار فقط مجموعه‌ی تست باز تعریف می‌شود هیچ تلاش آموزشی ی اضافی‌ای لازم نیست. توجه داشته باشید که روش‌های بالا را می‌توان با دید تغییر طول محورها با ضریب بررسی کرد. به طور مشابه می‌توان از ضرایبی برای تغییر طول محورها استفاده کرد که در فضای نمونه ای

[1] curse of dimensionality
[2] cross-validation

Telegram: @Farabar_BI
www.farabar.net

ثابت نیست. با این وجود، زمانی که با این دید درجه‌ی آزادی الگوریتم را برای باز تعریف معیار فاصله زیاد می‌کنیم احتمال پدیده‌ی overfit نیز زیاد می‌شود. بنابراین، روش تغییر ناحیه ای محورها خیلی متداول نیست.

یکی از مشکلات کاربردی k-Nearest Neighbor فهرست بندی[1] بهینه‌ی حافظه است. از آنجایی که این الگوریتم تمامی محاسبات را به زمان دریافت نمونه‌ی جدید موکول می‌کند، و ممکن است محاسبات قابل توجهی برای هر نمونه‌ی جدید لازم باشد، متد های بسیاری برای فهرست بندی نمونه های آموزشی ایجاد شده است تا با هزینه کردن مقداری حافظه پیدا کردن نزدیک‌ترین همسایه‌ها راحت‌تر گردد. یکی از این متد های فهرست بندی، متد kd-tree (Bentley 1975; Friedman 1977) است که در آن نمونه‌ها در برگ‌های درختی ذخیره می‌شوند و نمونه های مشابه نیز در همان گره یا گره های نزدیک ذخیره شده‌اند. گره های داخلی درخت، نمونه‌ی جدید x_q را با بررسی ویژگی‌هایش به سمت برگ مربوطه می‌فرستند.

۸.۲.۳ نکته ای در نماد گذاری

اکثر ادبیات به کار رفته در توضیح متد های nearest-neighbor و weighted local regression عبارات تخصصی است که از مبحث تشخیص الگو آماری[2] بر گرفته شده است. در خواندن چنین ادبیاتی بد نیست که با عبارات زیر آشنا باشید:

- برازش[3] یعنی تخمین یک تابع هدف حقیقی مقدار.
- باقیمانده[4] خطای $\hat{f}(x) - f(x)$ در تخمین تابع هدف است.
- تابع هسته[5] تابعی از فاصله است که برای مشخص کردن وزن هر یک از نمونه های آموزشی به کار می‌رود. به عبارت دیگر، تابع هسته تابعی مثل K است که $w_i = K(d(x_i, x_q))$.

۸.۳ برازش وزن دار محلی

روش nearest-neighbor که در قسمت قبل توضیح داده شد را می‌توان تخمینی از تابع هدف f(x) برای یک نمونه‌ی $x = x_q$ دانست. الگوریتم برازش وزن دار محلی تعمیمی از این الگوریتم است. این الگوریتم تخمینی صریح از f در محدوده‌ی اطراف x_q می‌سازد. برازش وزن دار محلی از نمونه های آموزشی نزدیک یا کل نمونه‌ها به صورت وزن دار متناسب با فاصله[6] برای ایجاد این تخمین محلی f استفاده می‌کند. برای مثال، ممکن است تابع هدف را در همسایگی اطراف x_q با استفاده از یک تابع خطی، یا یک تابع درجه دو، یا یک شبکه‌ی عصبی چند لایه، یا هر تابع دیگری تخمین بزنیم. عبارت "برازش وزن دار محلی" محلی است زیرا که دسته بندی فقط با توجه به نمونه های نزدیک نمونه‌ی جدید انجام می‌شود، وزن دار است زیرا که فاصله‌ی هر نمونه‌ی آموزشی بر تأثیر آن بر دسته بندی نمونه‌ی جدید اثر دارد، برازش است زیرا که این روش در یادگیری تخمین توابع حقیقی مقدار به کار می‌رود.

---

[1] indexing
[2] statistical pattern recognition
[3] Regression
[4] Residual
[5] Kernel function
[6] distance-weighted

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

با داشتن نمونه‌ی جدید x_q، روش کلی در برازش وزن دار محلی ساختن تخمینی مثل $\hat{f}$ است که در اطراف x_q نمونه های آموزشی را بپوشاند. سپس این تخمین برای محاسبه‌ی $\hat{f}(x_q)$ به کار می‌رود. مشخصات $\hat{f}$ از حافظه‌ی الگوریتم پاک خواهد شد زیرا که برای هر نمونه‌ی جدید باید تخمین محلی جدیدی ساخته شود.

۸.۳.۱ برازش وزن دار خطی محلی

حالتی را فرض کنید که برازش وزن دار محلی‌ای برای تخمین f در اطراف x_q از تابعی خطی به فرم کلی زیر استفاده می‌کند،

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

مثل قبل، در این رابطه نیز $a_i(x)$ نشان دهنده‌ی iامین ویژگی نمونه‌ی x است.

با توجه به آنچه که در فصل ۴ در مورد متدهایی مثل شیب نزول گفته شد، ضرایب مناسب $w_0, \ldots, w_n$ برای مینیمم کردن خطا بر روی نمونه های آموزشی استفاده می‌شود. در فصل ۴ علاقه‌ی ما به تخمین جهانی تابع هدف بود، بنابراین، خطای تعریف شده را برای تمامی نمونه های D تعریف کردیم،

$$E \equiv \frac{1}{2}\sum_{x \in D}\left(f(x) - \hat{f}(x)\right)^2 \tag{8.5}$$

که با آن به قانون شیب نزول زیر رسیدیم،

$$\Delta w_j = \eta \sum_{x \in D}\left(f(x) - \hat{f}(x)\right) a_j(x) \tag{8.6}$$

در این رابطه η ثابت یادگیری است. در اینجا قانون دوباره ارائه شده تا با نماد گذاری این فصل سازگاری داشته باشد (برای مثال، $t \to f(x), o \to \hat{f}(x), x_j \to a_j(x)$).

حال چگونه می‌توان این قانون را طوری تغییر داد تا به جای تخمینی جهانی، تخمینی محلی داشته باشد؟ راه حل ساده این است که خطای E را طوری تعریف کنیم تا تاکید بیشتری بر نمونه های محلی داشته باشد. سه روش مختلف اعمال این تغییر در زیر آورده شده. توجه داشته باشید برای تاکید بر اینکه این خطا تابعی از نمونه‌ی جدید x_q است می‌نویسیم $E(x_q)$.

- مینیمم کردن مربع خطای بین k نمونه‌ی نزدیک‌تر

$$E_1(x_q) \equiv \frac{1}{2}\sum_{x \in k\ nearest\ nbrs\ of\ x_q}\left(f(x) - \hat{f}(x)\right)^2$$

- مینیمم کردن مربع خطا برای کل نمونه‌ها، با این تفاوت که با افزایش فاصله بر اساس تابعی نزولی مثل K تأثیر نمونه‌ها کمتر می‌شود،

$$E_2(x_q) \equiv \frac{1}{2}\sum_{x \in D}\left(f(x) - \hat{f}(x)\right)^2 K(d(x_q, x))$$

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- ترکیبی از دو روش اول

$$E_3 \equiv \frac{1}{2} \sum_{x \in k\ nearest\ nbrs\ of\ x_q} \left(f(x) - \hat{f}(x)\right)^2 K(d(x_q, x))$$

شاید روش دوم زیباترین تعریف خطا باشد، زیرا که در آن تمامی نمونه‌ها اثری بر دسته بندی x_q خواهند داشت. با این وجود، این روش محاسباتی لازم دارد که با افزایش تعداد نمونه های آموزشی به طور خطی افزایش می‌یابند. روش سوم که بین دو روش اول است، هزینه‌ی محاسباتی مستقل از تعداد کل نمونه های آموزشی دارد؛ این روش فقط k همسایه‌ی نزدیک‌تر را در نظر می‌گیرد.

اگر روش سوم را انتخاب کنیم قانون شیب نزول به صورت زیر بدست می‌آید (تمرین ۸.۱):

$$\Delta w_j = \eta \sum_{x \in k\ nearest\ nbrs\ of\ x_q} K\left(d(x_q, x)\right)\left(f(x) - \hat{f}(x)\right) a_j(x) \qquad (8.7)$$

توجه داشته باشید که تنها تفاوت بین این قانون جدید و قانون قبل در رابطه‌ی ۸.۶ این است که وزن هر نمونه آموزشی در مقدار $K\left(d(x_q, x)\right)$ ضرب شده است و خطا نیز فقط روی k نمونه‌ی نزدیک‌تر محاسبه می‌شود. در واقع اگر می‌خواهیم از توابع خطی برای تخمین تابع هدف بر روی دسته‌ی ثابتی از نمونه های آموزشی استفاده کنیم، متد های موثر تری نیز وجود دارد تا مسئله را از راه مستقیم برای بدست آوردن ضرایب $w_0 \dots w_n$ حل کنیم. (Atkeson 1997a) و (Bishop 1995) بررسی‌ای روی انواع این نوع متدها انجام داده‌اند.

۸.۳.۲ نکاتی در مورد برازش وزن دار محلی

در بالا تخمین تابع f را با استفاده از تابعی خطی در همسایگی نمونه‌ی جدید x_q بررسی کردیم. ادبیات برازش وزن دار محلی، شامل طیف وسیعی از متد های جایگزین برای وزن دهی بر اساس فاصله‌ی نمونه های آموزشی است، و همچنین طیف وسیعی از متد های تقریب محلی برای تابع هدف موجود است. در اکثر موارد، تابع هدف با یک تابع ثابت، خطی و یا درجه دو تخمین زده می‌شود. به دو دلیل از توابع پیچیده تر استفاده نمی‌شود: (۱) هزینه‌ی سازگار کردن توابع پیچیده تر برای نمونه های جدید به شدت زیاد است و (۲) این تخمین‌های ساده تابع هدف را در یک محدوده کوچک از فضای نمونه ای خیلی خوب مدل سازی می‌کنند.

۸.٤ توابع پایه ای شعاعی

یکی از روش‌های تخمین توابع که بسیار مشابه برازش وزن دار محلی و شبکه های عصبی است یادگیری با توابع پایه ای شعاعی[1] است (Powell 1987; Broomhead and Lowe 1988; Moody and Darken 1989). در این روش تابع یاد گرفته شده تابعی به فرم زیر است.

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u\left(d(x_u, x)\right) \qquad (8.8)$$

[1] Radial basis functions

www.farabar.net

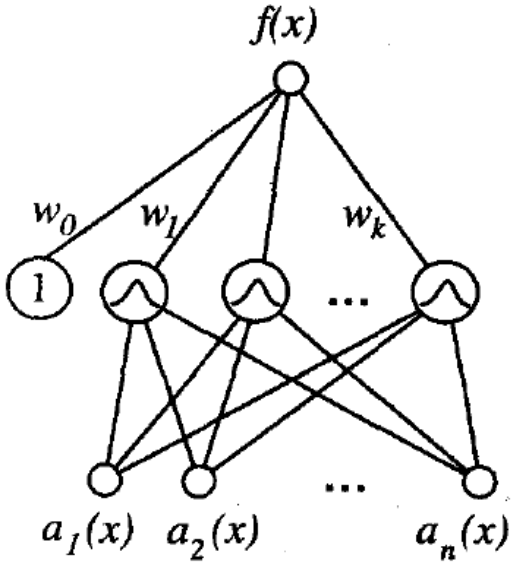
Telegram: @Farabar_BI

در این رابطه هر x_u نمونه ای از X است که در آن $K_u(d(x_u,x))$ با افزایش $d(x_u,x)$ کاهش می‌یابد. در اینجا k ثابتی است که توسط کاربر تعیین می‌شود و تعداد توابع هسته‌ی مشمول را مشخص می‌کند. حتی اگر $\hat{f}(x)$ تخمینی جهانی‌ای از f باشد، تأثیر هر یک از این $K_u(d(x_u,x))$ به منطقه‌ی نزدیک به نقطه‌ی x_u محدود می‌شود. معمولاً تابع $K_u(d(x_u,x))$ را تابع گوسی (جدول ۵.۴) با میانگین x_u و واریانس σ_u^2 در نظر می گیرند.

$$K_u(d(x_u,x)) = e^{\frac{1}{2\sigma_u^2}d^2(x_u,x)}$$

در اینجا بحث مان را محدود به چنین توابع هسته‌ی گوسی می‌کنیم. همان طور که در (Harman 1990) نیز آمده است فرم تابعی رابطه‌ی ۸.۸ می‌تواند با افزایش k به میزان لازم و تغییر σ^2 در هر یک از این توابع به طور دلخواه، هر تابعی را تخمین بزند.

تابعی که در رابطه‌ی ۸.۸ آمده است را می‌توان مشابه یک شبکه‌ی دو لایه دانست که در آن لایه‌ی اول مقادیر مختلف $K_u(d(x_u,x))$ را محاسبه می‌کند و لایه‌ی دوم ترکیبی خطی از این مقادیر خروجی را ایجاد می‌کند. مثالی از تابع پایه ای شعاع (RBF) در شکل ۸.۲ نشان داده شده است.



شکل ۸.۲ تابع شبکه‌ی پایه ای شعاعی.

هر واحد پنهان یک تابع فعالیت[1] را که گوسی با میانگین x_u است محاسبه می‌کند. بنابراین، اگر مقدار x به x_u نزدیک نباشد مقدار تابع فعالیت تقریباً صفر خواهد بود. هر واحد خروجی ترکیبی خطی از توابع فعالیت لایه‌ی پنهان است. با وجود اینکه شکل نشان داده شده فقط یک خروجی دارد، اما شبکه‌هایی را می‌توان ساخت که چندین خروجی داشته باشند.

[1] Activation function

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

با معلوم بودن مجموعه‌ی نمونه های آموزشی، شبکه‌ی RBF در فرایندی ۲ مرحله ای آموزش داده می‌شود. ابتدا تعداد واحد های پنهان k معلوم و هر واحد پنهان u با تعیین مقدار x_u و σ_u^2 تعریف می‌گردد. سپس، وزن‌های w_u با استفاده از رابطه‌ی خطای جهانی ۸.۵ برای ماکزیمم کردن تناسب[1] شبکه بر روی داده های آموزشی آموزش داده می‌شوند، چون توابع هسته ثابت نگه داشته شده‌اند، در این مرحله تعیین کردن وزن‌های خطی w_u می‌تواند بسیار موثر باشد.

روش‌های جایگزین بسیاری برای تعیین تعداد واحد های لایه‌ی پنهان یا همان تعداد توابع هسته ارائه شده است. یکی از این روش‌ها در نظر گرفتن یک تابع هسته‌ی گوسی‌ای با میانگین x_i برای هر نمونه‌ی $< x_i, f(x_i) >$ است. می‌توان برای همه‌ی این توابع هسته میزان پهنای یکسانی را در نظر گرفت (واریانس‌ها را مساوی گرفت). یکی از مزیت‌های این روش تناسب کامل شبکه‌ی RBF با داده های آموزشی است. به عبارت دیگر، می‌توان برای هر دسته‌ی دلخواه آموزشی m نمونه ای با تعیین مناسب $w_0, \dots, w_m$ توابع گوسی را طوری ترکیب کرد که برای تمامی نمونه های $< x_i, f(x_i) >$ داشته باشیم، $\hat{f}(x) = f(x_i)$.

روش دوم، انتخاب مجموعه ای از توابع هسته ای است که تعدادشان از تعداد نمونه های آموزشی کمتر است. این روش مخصوصاً زمانی که تعداد نمونه های آموزشی زیاد است بسیار موثر تر از روش اول است. مراکز توابع هسته می‌توانند توزیع یکنواختی بر روی فضای نمونه ای X داشته باشد. یا در مقابل، ممکن است بخواهیم مراکز توابع هسته را به طور غیر یکنواخت در فضای نمونه ای پخش کنیم، معمولاً زمانی که خود نمونه های آموزشی توزیع غیر یکنواخت در فضای نمونه ای دارند چنین روش مورد استفاده قرار می‌گیرد. در چنین شرایطی می‌توان مراکز توابع هسته را زیر مجموعه ای تصادفی از نمونه های آموزشی قرار داد، با این کار از توزیع احتمال حاکم بر توزیع نمونه‌ها نمونه برداری می‌کنیم. یا ممکن است خوشه های نمونه‌ها[2] را تشخیص داده و تابع هسته ای با مرکز هر کدام را در نظر بگیریم. تعیین مکان توابع هسته به این صورت را می‌توان با استفاده از الگوریتم‌های خوشه یابی[3] که با نمونه های آموزشی (و نه مقادیر هدفشان) را با ترکیبی از توابع گوسی تخمین می‌زنند انجام داد. الگوریتم EM، که در بخش ۶.۱۲.۱ بررسی شد، الگوریتمی برای پیدا کردن ترکیبی از k تابع گوسی برای تناسب بهینه با نمونه های مشاهده شده را ارائه می‌کند. در الگوریتم EM، میانگین‌ها طوری انتخاب می‌شوند که احتمال مشاهده‌ی نمونه های x_i به شرط داشتن میانگین‌های تخمینی حداکثر شود. توجه دارید که مقادیر تابع هدف $f(x_i)$ در محاسبات مراکز توابع هسته در متد های خوشه یابی درگیر نمی‌شود و تنها تأثیر مقادیر تابع هدف $f(x_i)$ در این حالت مشخص کردن وزن‌های لایه‌ی خروجی است.

به طور خلاصه، شبکه های توابع پایه ای شعاعی، تخمینی جهانی با ترکیب خطی توابع هسته‌ی محلی برای تابع هدف ایجاد می‌کنند. مقدار توابع هسته فقط زمانی ناچیز نیست که نمونه‌ی x در ناحیه تعریف شده با مرکز و عرض باشد. بنابراین به شبکه های توابع پایه ای شعاعی می‌توان به دید ترکیب خطی هموار تعداد زیادی تخمین محلی از تابع هدف نگاه کرد. یکی از مزیت‌های کلیدی شبکه های RBF این است که این شبکه‌ها را می‌توان راحت‌تر از شبکه های feedforward با الگوریتم Backpropagation آموزش داد. این حقیقت از این رو است که لایه‌ی اول و لایه دوم شبکه های RBF به طور جداگانه آموزش داده می‌شوند.

[1] fitness
[2] prototypical clusters of instances
[3] clustering algorithms

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۸.۵ استدلال مبتنی بر شرایط

روش‌های مبتنی بر شرایط[1] مثل k-Nearest Neighbor و برازش وزن دار محلی در سه ویژگی کلیدی مشترکند. ابتدا اینکه این متدها تنبلند[2]، بدین معنا که تعمیم بر روی نمونه های آموزشی را به زمانی که یک نمونه‌ی جدید ارائه می‌شود موکول می‌کنند. دوم اینکه نمونه‌ی جدید را بر اساس نمونه های مشابه دسته بندی می‌کنند و با نمونه های متفاوت با نمونه‌ی جدید کاری ندارند. سوم اینکه به نمونه‌ها به شکل نقاطی در فضای n بعدی اقلیدسی نگاه می‌کنند. استدلال مبتنی بر شرایط (CBR) [3] دو ویژگی اول را دارد، اما در ویژگی سوم مشابه دو متد قبلی نیست. در CBR، معمولاً نمونه‌ها با توضیحاتی غنی‌تر نمایش داده می‌شوند و متد های استخراج نمونه های مشابه نیز استادانه تر هستند. CBR در مسائلی چون طراحی مفهومی قطعات مکانیکی بر اساس پایگاه داده ای از طراحی‌های قبلی (Sycara et al. 1992) ، استدلال درباره‌ی پرونده های قانونی بر اساس حکم‌های قبلی (Ashley 1990) ، و حل مسائل برنامه ریزی بر اساس استفاده‌ی دوباره و باز ترکیب حل‌های قبلی مسائل مشابه (Veloso 1992) به کار گرفته شده است.

بیایید بحث را با مطرح کردن یک فرم کلی از مسائل مبتنی بر شرایط شروع کنیم. سیستم CADET (Sycara et al. 1992) از استدلال مبتنی بر شرایط برای همکاری در طراحی مفهومی قطعات مکانیکی ساده چون شیر آب استفاده می‌کند. این سیستم از پایگاه داده ای با حدود ۷۵ طراحی قطعه طراحی قبلی برای پیشنهاد دادن طراحی مفهومی متناسب با خاصیت‌های مسئله‌ی طراحی جدید استفاده می‌کند. هر نمونه‌ی ذخیره شده در حافظه (برای مثال، یک لوله‌ی آب) با دو ویژگی ساختار و قابلیت‌ها توصیف شده است. مسائل طراحی جدید معلوم کردن ساختار با دانستن قابلیت‌های لازم خواهد بود. این تعریف مسئله در شکل ۸.۳ نشان داده شده است. نیمه‌ی بالایی شکل توصیف یک نمونه‌ی ذخیره شده در حافظه به نام لوله‌ی اتصال T شکل[4] را نشان می‌دهد. قابلیت‌های این قطعه با روابط بین جریان‌های آب و دمای ورودی و خروجی نمایش داده شده است. در نمایش قابلیت‌ها در سمت راست علامت + به این معناست که متغیر در سمت انتهای پیکان با افزایش متغیر سمت ابتدای پیکان افزایش خواهد یافت. برای مثال، جریان آب خروجی Q_3 با افزایش جریان آب ورودی Q_1 افزایش می‌یابد. به طور مشابه، علامت – به این معناست که متغیر انتهای پیکان با افزایش متغیر ابتدای پیکان کاهش می‌یابد. این نوع قابلیت‌ها رفتار لازم برای نوعی شیر آب را مشخص می‌کند. در اینجا Q_c نشان دهنده‌ی جریان آب سرد و Q_h جریان آب گرم به شیر و Q_m جریان ترکیبی خروجی شیر را نشان می‌دهد. به طور مشابه T_c و T_h و T_m دمای آب سرد، گرم و ترکیبی را نشان می‌دهند. متغیر C_t سیگنال کنترل دمای آب خروجی شیر و C_f سیگنال کنترل جریان خروجی شیر است. توجه دارید که کنترل‌های C_t و C_f بر جریان‌های Q_c و Q_h تأثیر می‌گذارند، و به طور غیر مستقیم بر جریان Q_m و دمای خروجی آب شیر T_m تأثیر دارند.

با معلوم بودن قابلیت انتظاری طراحی جدید، CADET در پایگاه داده‌ی خود به دنبال نمونه‌هایی با قابلیت‌های مشابه می‌گردد. اگر نمونه ای دقیقاً با قابلیت انتظاری هماهنگی داشت، پس می‌توان از همان طراحی برای حل مسئله استفاده کرد، پس طراحی نمونه به عنوان راه حل پیشنهادی سیستم ارائه می‌شود. اگر هیچ نمونه ای قابلیت دقیقاً با قابلیت‌های انتظاری وجود نداشت CADET حالت‌هایی را که گراف قابلیت‌هایشان با قسمتی از گراف قابلیت انتظاری تطابق داشته باشد پیدا خواهد کرد. برای مثال در شکل ۸.۳ قابلیت‌های اتصال T شکل با یکی از زیر گراف‌های قابلیت‌های شیر مورد نظر تطابق دارد. در حالت کلی‌تر، CADET به دنبال زیر گراف‌هایی همشکل[5] بین در گراف قابلیت

---

[1] Cased-Based
[2] lasy
[3] Cased-Based reasoning
[4] T-junction pipe
[5] isomorphism

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

می‌گردد، بنابراین ممکن است قسمت‌هایی از یک قابلیت انتظاری ممکن است با قسمت‌های از یک طراحی خاص تطابق داشته باشد. علاوه بر این، سیستم می‌تواند به طرز استادانه ای قابلیت‌های اصلی را برای پیدا کردن گراف‌هایی با قابلیت‌های معادل تغییر دهد تا بتواند طراحی‌های بیشتری را که با گراف معادل تطابق دارند پیدا کند. این سیستم از دانش کلی تأثیرات فیزیکی برای ایجاد چنین گراف‌های قابلیت معادلی استفاده می‌کند. برای مثال، این سیستم از قانون بازنویسی‌ای استفاده می‌کند که عبارت

$$A \xrightarrow{+} B$$

را به عبارت

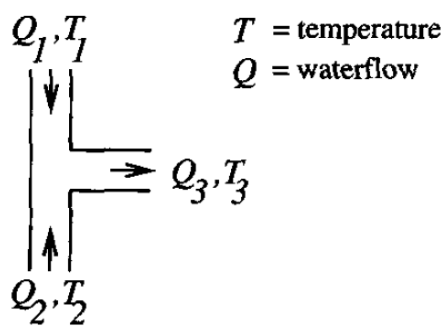
$$A \xrightarrow{+} x \xrightarrow{+} B$$

تبدیل می‌کند. این قانون بازنویسی را می‌توان به فرم اینکه "اگر A باید با افزایش B افزایش بیابد، کافی است که کمیتی مثل x را پیدا کنیم که B با افزایش x افزایش بیابد و x نیز با افزایش A افزایش بیابد. در اینجا x کمیتی جهانی است که مقدارش در تطابق گراف قابلیت‌های انتظاری با نمونه های پایگاه داده محدود است. در حقیقت گراف قابلیت‌های انتظاری شیر در شکل ۸.۳ همان قابلیت‌های اصلی است که با این قانون بازنویسی شده است.

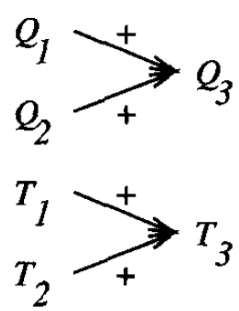
با بازیابی نمونه‌هایی که با زیر گراف‌هایی از قابلیت‌های انتظاری تطابق دارند، طراحی کلی را می‌توان کنار هم قرار داد. در کل، فرایند ایجاد راه حل نهایی از چندین نمونه‌ی بازیابی شده می‌تواند بسیار پیچیده باشد. این فرایند ممکن است علاوه بر ترکیب نمونه های بازیابی شده نیاز به طراحی قسمت‌هایی از سیستم از قوانین اولیه نیاز داشته باشد. همچنین ممکن است نیاز به بازگشت به انتخاب‌های قبلی طراحی زیر هدف‌ها، و متعاقباً رد کردن نمونه بازیابی شده داشته باشد. CADET قابلیت‌های بسیار محدودی در ترکیب و تجزیه‌ی نمونه های بازیابی شده دارد و بنابراین در این مرحله از فرایند به شدت به کاربر وابسته است. همان طور که در (Sycara et al. 1992) نیز توصیف شده، CADET یک سیستم کلی تحقیقاتی برای کاوش نقش بالقوه‌ی استدلال مبتنی بر شرایط در طراحی مفهومی است. این سیستم الگوریتم‌های لازم برای تغییر طراحی‌های اولیه و رسیدن به طراحی نهایی را شامل نمی‌شود.

A stored case: T–junction pipe

Structure:



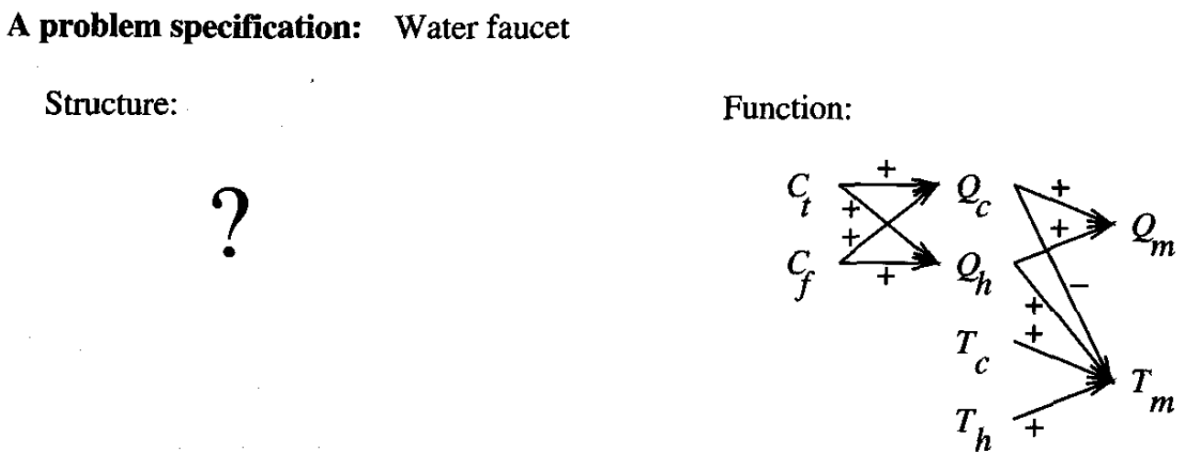
Function:



www.farabar.net

Telegram: @Farabar_BI

www.farabar.net



شکل ۸.۳ یک نمونه‌ی ذخیره شده و یک مسئله‌ی جدید.

نیمه‌ی بالای یک طراحی جزئی را در پایگاه داده‌ی CADET نشان می‌دهد. کارایی‌ها با نمودار وابستگی کمیت‌ها در میان متغیر های اتصال T شکل (که در متن توضیح داده شده) نمایش داده شده است. نیمه‌ی پایین شکل نمونه ای از مسئله‌ی معمول طراحی را نشان می‌دهد.

بررسی تناظر بین تعریف مسئله های CADET و تعریف مسئله‌ی کلی متد های چون k-Nearest Neighbor می‌تواند بسیار مفید باشد. در CADET هر نمونه‌ی آموزشی ذخیره شده گراف قابلیت و ساختاری که این قابلیت را عملی می‌کند در برمی گیرد و نمونه های جدید گراف‌های قابلیت جدید هستند. بنابراین، می‌توان تعریف مسئله‌ی CADET را با تعریف X به عنوان فضای تمامی گراف‌های قابلیت به شکل نمادگذاری استانداردمان بیان کرد. تابع هدف f این گراف‌های قابلیت را به ساختارهای عملی آن‌ها تبدیل می‌کند. پس هر نمونه‌ی ذخیره شده به فرم $< x, f(x) >$ را می‌توان با گراف قابلیت x و ساختار f(x) که ساختار عملی x است نمایش داد. این سیستم باید با استفاده از نمونه های آموزشی یاد بگیرد تا ساختار خروجی $f(x_q)$ را برای قابلیت انتظاری x_q پیدا کند.

این طرح سیستم CADET تعداد زیادی از خواص عمومی سیستم‌های استدلال مبتنی بر شرایط را که آن‌ها را از الگوریتم‌های چون k-Nearest Neighbor تمییز می‌دهند نشان می‌دهد.

- نمونه‌ها یا حالات[1] را ممکن است با توضیحات نمادین غنی، مثل گراف‌های قابلیت در CADET نمایش دهیم. این سیستم نیز به معیاری استاندارد مشابه فاصله‌ی اقلیدسی، مثل اندازه‌ی بزرگ‌ترین زیر گراف مشترک بین دو گراف قابلیت، برای تعیین تشابه بین نمونه‌ها نیاز دارد.
- ممکن است برای ایجاد راه حل جدید از ترکیب نمونه های بازیابی شده استفاده کنیم. این خاصیت مشابه روش k-Nearest Neighbor است، در k-Nearest Neighbor نیز از نمونه های مشابه برای ساخته دسته بندی نمونه‌ی جدید استفاده می‌شود. با این وجود، این فرایند در استدلال مبتنی بر شرایط ممکن است بسیار پیچیده باشد و نیاز به استدلال مبتنی بر دانش به جای متد های آماری خواهد داشت.
- ممکن است رابطه‌ی نزدیکی بین بازیابی نمونه‌ها، استدلال مبتنی بر دانش و حل مسئله وجود داشته باشد. یکی از مثال‌های ساده‌ی چنین ارتباط نزدیکی در CADET دیده می‌شود، این سیستم از دانش کلی درباره‌ی تأثیرات برای بازنویسی گراف‌های قابلیت در تلاشش برای پیدا کردن نمونه های مشابه استفاده می‌کند. سیستم‌های دیگری نیز طراحی شده‌اند که کاملاً استدلال مبتنی بر

[1] cases

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

شرایط را به سیستم‌های حل مسئله‌ی مبتنی بر جستجو[1] تبدیل می‌کنند. Anapron (Golding and Rosenbloom 1991) و Prodigy/Analogy (Veloso 1992) چنین سیستم‌هایی هستند.

به طور خلاصه استدلال مبتنی بر شرایط متد یادگیری مبتنی بر نمونه‌هاست که نمونه‌ها در ان با توضیحات غنی نسبی بیان شده و بازیابی و ترکیب نمونه‌ها برای حل نمونه‌ی جدید وابسته به دانش قبلی و متدهای قوی جستجویی حل مسئله باشد. مشکلات فعلی استدلال مبتنی بر شرایط ایجاد متد های بهینه‌ی فهرست بندی نمونه‌هاست. مشکل اصلی معیار تشخیص تشابهات (برای مثال، زیر گراف‌های مشابه در گراف‌های قابلیت) که تخمینی از رابطه‌ی نمونه جزئی با مسئله‌ی جزئی است می‌باشد. زمانی که سیستم CBR برای استفاده از نمونه های بازیابی شده تلاش می‌کند ممکن است مشکلات ناشی از این معیار تشابه را نپوشاند. برای مثال در CADET تکه طراحی‌های بازیابی شده ممکن است با یکدیگر در تضاد باشد، و در نتیجه دیگر ترکیب و ساخت طراحی سازگار غیرممکن خواهد بود. در کل زمانی که چنین اتفاقی می‌افتد، سیستم CBR می‌تواند بازگشته و به دنبال نمونه های دیگری در میان نمونه های موجود بگردد، یا مسئله را به متد حل مسئله ای دیگر واگذار کند. زمانی که چنین مشکلاتی تشخیص داده می‌شوند، نمونه های آموزشی برای بهبود معیار تشابه، یا به طور معادل، فهرست بندی ساختار در پایگاه داده فراهم می‌شود. در کل زمانی که نمونه ای بر اساس معیاری بازیابی می‌شود، اما بر اساس بررسی‌های بعدی نامربوط تشخیص داده می‌شود، باید معیار طوری بازبینی شود که این نمونه را در جستجو های آینده رد کند.

۸.۶ نکاتی در مورد یادگیری‌های تنبل و کوشا[2]

در این فصل سه متد تنبل را بررسی کردیم: k-nearest neighbors، برازش وزن دار محلی و استدلال مبتنی بر شرایط. این متدها برای اینکه چگونگی تعمیم روی نمونه های آموزشی را به زمانی که نمونه‌ی جدید ارائه می‌شود واگذار می‌کنند، تنبل خوانده می‌شوند. همچنین متد های کوشایی را نیز بررسی کردیم: متدهایی که برای یادگیری شبکه های توابع پایه ای شعاعی استفاده می‌شود. این متدها را چون تعمیم را قبل از مواجهه با نمونه‌ی جدید انجام می‌دهند کوشا می‌نامیم، این تعمیم با ساختار شبکه و وزن‌های تعریف شده برای تخمین تابع هدف انجام می‌شود. به این ترتیب تمامی الگوریتم‌های معرفی شده در این کتاب (مثل، Backpropagation و C4.5) الگوریتم‌های یادگیری کوشا هستند.

آیا آنچه الگوریتم‌های تنبل می‌توانند یاد بگیرند با آنچه الگوریتم‌های کوشا می‌توانند یاد بگیرند تفاوت چشم گیری دارد؟ بیایید ابتدا دو نوع تفاوت را مشخص کنیم: تفاوت در زمان محاسبات و تفاوت در دسته بندی‌های تولید شده برای نمونه های جدید. تفاوت‌های واضحی در زمان محاسبه‌ی الگوریتم‌های یادگیری تنبل و کوشا وجود دارد. برای مثال، متد های تنبل در طول آموزش محاسبات کمتری لازم دارند، اما در هنگام پیش بینی ویژگی هدف برای نمونه‌ی جدید محاسبات زیادی انجام می‌دهند.

سؤال اساسی‌تر این است که آیا تفاوت‌های اساسی‌ای در بایاس‌های استقرایی الگوریتم‌های تنبل و الگوریتم‌های کوشا وجود دارد. از این نظر تفاوت‌های زیر بین متد های تنبل و کوشا وجود دارد.

- در متد های تنبل گاهی تصمیم گیری برای چگونگی تعمیم بر روی داده های آموزشی D به نمونه‌ی آموزشی ارائه شده نیز وابسته می‌شود.

[1] search-based
[2] eager

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

- متد های کوشا این وابستگی را نمی‌توانند داشته باشند، زمانی که یک متد کوشا با یک نمونه‌ی جدید مواجه می‌شود، تخمین جهانی را انجام داده است.

آیا این تمایز دقت تعمیم یادگیر را تحت تأثیر قرار می‌دهد؟ اگر دو یادگیر کوشا و تنبل از فضای فرضیه ای یکسانی مثل H استفاده کننده این تمایز تأثیر گذار می‌شود. برای تصور، فرض کنید فضای فرضیه ای تمام توابع خطی است. اگر از متد برازش وزن دار محلی که قبلاً مطرح شد برای این فضای فرضیه استفاده کنیم، برای هر نمونه‌ی جدید x_q برای تعمیم رو نمونه های آموزشی این متد فرضیه ای را انتخاب می‌کند که نزدیک x_q باشد. در نقطه‌ی مقابل، یک یادگیر کوشا که از همان فضای فرضیه ای توابع خطی استفاده می‌کند، تخمین خود را از تابع هدف قبل از مواجهه با نمونه های جدید مشخص می‌کند. بنابراین یادگیر کوشا فقط بر اساس یک تابع خطی نمونه های جدید و کل فضای نمونه ای را دسته بندی می‌کند. پس متد های تنبل که از فضای فرضیه ای شاملی استفاده می‌کنند زیرا که با استفاده از مینیمم‌های موضعی توابع خطی برای تشکیل تخمینشان از تابع هدف استفاده می‌کنند. توجه داشته باشید که این شرایط برای یادگیر های دیگر و فضای فرضیه های دیگر نیز صادق است. برای مثال نسخه‌ی تنبل Backpropagation می‌تواند برای هر نمونه‌ی جدید یک شبکه‌ی عصبی یاد بگیرد، اما نسخه‌ی کوشا (فصل ۴) فقط یک شبکه‌ی عصبی برای کل نمونه‌ها یاد خواهد گرفت.

نکته‌ی کلیدی در پاراگراف بالا این است که یادگیر تنبل می‌تواند با ترکیب تخمین‌های موضعی تابع هدف را یاد گیرد، در حالی که یادگیر کوشا فقط یک تخمین جهانی را با توجه به نمونه های آموزشی یاد می‌گیرد. این تفاوت بین یادگیری کوشا و تنبل به تفاوت بین تخمین موضعی و جهانی تابع هدف بر می‌گردد.

آیا می‌توان متد های کوشایی ساخت که از تخمین‌های موضعی استفاده کند؟ شبکه های RBF تلاشی برای دست یابی به چنین متدهایی است. متد های یادگیری RBF که ما بررسی کردیم تخمین جهانی از تابع هدف را در زمان یادگیری ایجاد می‌کنند، با این وجود، یک شبکه‌ی RBF این تابع جهانی را به صورت ترکیب خطی چندین تابع هسته‌ی موضعی بیان می‌کند. اما چون یادگیری RBF باید قبل از مواجهه با نمونه‌ی جدید این فرضیه را مشخص کند، این تخمین‌های موضعی مشابه توابع موضعی یادگیر تنبل که مخصوصاً برای نمونه‌ی جدید ساخته شده‌اند، منحصراً برای نمونه‌ی جدید ساخته نشده‌اند. در عوض، شبکه های RBF به صورت کوشا از توابع موضعی که در حوالی نمونه های آموزشی یا دسته نمونه های آموزشی ایجاد می‌شود، تشکیل شده‌اند، اما این تخمین موضعی‌ای حوالی نمونه‌ی مجهول جدید ساخته نمی‌شود (بدین مفهوم که هر تخمین منحصر به نمونه‌ی جدید نیست و برای تمامی نمونه های جدید ارائه می‌گردد).

به طور خلاصه، متد های تنبل حق انتخاب بین فرضیه‌ها یا تخمین‌های موضعی تابع هدف برای هر نمونه‌ی جدید دارند. در حالی که متد های کوشا محدودترند و باید با استفاده از یک فرضیه باید کل فضای نمونه ای را پوشش دهند. البته متد های کوشا نیز می‌توانند از ترکیبی از تخمین‌های موضعی استفاده کنند (مثل شبکه های RBF). با این وجود، حتی این ترکیب تخمین‌های موضعی قابلیت کامل متد های تنبل در تغییر بر اساس نمونه‌ی آموزشی مجهول را به یادگیر کوشا نمی‌دهد.

۸.۷ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر می‌شود:

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

- متدهای یادگیری مبتنی بر نمونه‌ها با دیگر روش‌های تخمین توابع از این جهت متفاوتند که پردازش مربوطه به نمونه های آموزشی را به زمانی که لازم است نمونه‌ی جدیدی دسته بندی شود به تعویق می‌اندازند. در نتیجه، نیازی ندارند که فرضیه را به طور صریح در تمامی فضای نمونه ای مستقل از نمونه‌ی جدید بیان کنند. در مقابل برازش محلی از تابع هدف برای هر نمونه به ما می‌دهند.
- مزیت متدهای مبتنی بر نمونه شامل قدرت مدل سازی توابع هدف پیچیده با استفاده از تقریب‌هایی با پیچیدگی کمتر و اینکه داده های نمونه های آموزشی هیچ گاه از دست نخواهند رفت (نمونه‌ها به طور صریح ذخیره خواهند شد) است. مشکل عملیاتی اصلی پرهزینه بودن دسته بندی نمونه های جدید (مخصوصاً زمانی که نمونه‌ها با خواص سمبولیک پیچیده توصیف می‌شوند) و تأثیر منفی ویژگی‌های نامربوط است.
- الگوریتم k-nearest meighbor الگوریتمی مبتنی بر نمونه‌ها برای تقریب توابع هدف حقیقی مقدار گسسته یا پیوسته است با این فرض که نمونه‌ها متناسب با فضای اقلیدسی n بعدی‌اند. مقدار هدف یک نمونه‌ی جدید با مقادیر k نزدیک‌ترین نمونه‌ی آموزشی تقریب زده می‌شود.
- متدهای برازش وزن دار محلی یک تعمیم از الگوریتم k-nearest meighbor اند با این فرض که برای هر نمونه‌ی جدید تقریبی خطی ایجاد می‌گردد. برازش محلی تابع هدف ممکن است بر پایه‌ی فرم‌های مختلفی از توابع مثل ثابت، خطی، درجه دو و یا توابع هسته‌ی محدود در فضا باشد.
- شبکه های RBF نوعی از شبکه های عصبی هستند که از توابع هسته‌ی محدود در فضا ساخته می‌شوند. این شبکه‌ها را می‌توان مخلوطی از روش‌های مبتنی بر نمونه‌ها (تأثیر محدود در فضای هر تابع هسته) و روش شبکه های عصبی (تقریب کلی تابع هدف بر اساس نمونه های آموزشی و در زمان آموزش و نه در هنگام دسته بندی نمونه های جدید) دانست. استفاده از شبکه های عصبی RBF به طور موفق در کاربردهای از جمله تشخیص تصویر که در آن بررسی محلی‌ای کاملاً توجیه شده است به کار رفته است.
- استدلال مبتنی بر حالت نیز نوعی روش مبتنی بر نمونه است که در آن نمونه‌ها توسط توضیحات پیچیده‌ی منطقی به جای نقاط فضای اقلیدسی نمایش داده می‌شوند. با این توصیفات نمادین پیچیده‌ی نمونه‌ها، متدهای بسیار و غنی‌ای برای نگاشت نمونه های آموزشی به مقادیر توابع هدف پیشنهاد شده است. متدهای استدلال مبتنی بر حالت در بسیاری از کاربردها مثل مدل سازی استدلال قانونی و برای راهنمایی جستجو در تولیدهای پیچیده و مسائل حمل نقل به کار رفته است.

الگوریتم k-nearest neighbor از جمله الگوریتم‌های یادگیری ماشینی است که کاملاً مورد تحلیل و بررسی قرار گرفته، دلیل این بررسی‌ها سادگی و قدمت این الگوریتم است. Cover and Hart (1967) نتایج تئوری اولیه را مطرح می‌کنند، Duda and Hart (1973) دید کلی خوبی از این الگوریتم ارائه می‌کند. Bishop (1995) بحثی در مورد الگوریتم k-nearest neighbor مطرح کرده و رابطه‌ی آن را با تخمین چگالی توزیع احتمال را بررسی می‌کند. تحقیق جدید خوبی که در باره ی متدهای برازش خطی محلی در Atkeson et al. (1997) انجام شده است. کاربرد این متدها در کنترل ربات نیز توسط Atkeson et al. (1997b) انجام شده است.

بحث جامعی از توابع پایه ای شعاعی در Bishop (1995) انجام گرفته شده است. دیگر کاربردها نیز در Powell (1987) و Poggio and Girosi (1990) بررسی شده‌اند. برای الگوریتم EM بحث شده در این کتاب به قسمت ۶.۱۲ که در آن تخمین میانگین ترکیبی از چندین تابع گوسی آمده رجوع کنید.

Kolodner (1993) نیز معرفی‌ای بر استدلال مبتنی بر حالت انجام می‌دهد. دیگر تحقیقات کلی و مجموعه های تحقیقات جدید در Aamodt et al. (1994), Aha et al. (1991), Haton et al. (1995), Riesbeck and Schank (1989), Schank et al. (1994), Veloso and Aamodt (1995), Watson (1995), Wess et al. (1994) آمده است.

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

تمرینات

۸.۱ قانون شیب نزول را برای یک برازش وزن دار محلی که در رابطه‌ی ۸.۱ آمده استخراج کنید.

۸.۲ متد جایگزین زیر را برای فاصله در برازش وزن دار محلی در نظر بگیرید. مجموعه ای مجازی از نمونه های آموزشی به نام D' ایجاد کنید که: برای هر نمونه‌ی آموزشی $<x,f(x)>$ در مجموعه‌ی واقعی D تعداد $K(d(x_q,x))$ کپی در D' قرار دهید. حال تقریبی خطی برای مینیمم کردن معیار خطای زیر انجام دهید:

$$E_4 \equiv \frac{1}{2}\sum_{x\in D'}\left(f(x)-\hat{f}(x)\right)^2$$

ایده‌ی اصلی کپی کردن نمونه های آموزشی که بیشتر نزدیک نمونه هستند و کمتر آن‌هایی که دورتر هستند. شیب نزول را برای این معیار استخراج کنید. این قانون را بر حسب مجموع روی اعضای D بیان کنید نه اعضای D' و قانون را با قوانین روابط ۸.۶ و ۸.۷ مقایسه کنید.

۸.۳ نمونه‌ی تنبلی از الگوریتم کوشای ID3 پیشنهاد کنید (فصل ۳). مزیت‌ها و مضرت‌های الگوریتم پیشنهادی شما نسبت به الگوریتم کوشای اصلی چیست؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

ارزیابی	cross-validation
وزن دار متناسب با فاصله	distance-weighted
باقیمانده	Residual
برازش	Regression
برازش وزن دار محلی	locally weighted regression
تابع پایه ای شعاعی	radial basis function
تابع هسته	Kernel function
تشخیص الگو آماری	statistical pattern recognition
تنبل	Lasy
حالات	Cases
طلسم بعد	curse of dimensionality
کوشا	Eager
الگوریتم نزدیک‌ترین همسایه	Nearest neighbor algorithm
الگوریتم‌های خوشه یابی	clustering algorithms
مبتنی بر شرایط	Cased-Based
متد جهانی	global method
متد محلی	local method
نزدیک‌ترین همسایه	Nearest Neighber

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

نمودار ورونوی مجموعه‌ی نمونه های آموزشی	Voronoi diagram
همشکل	Isomorphism
یادگیری مبتنی بر نمونه‌ها	instance based learning

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

فصل نهم: الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک روشی برای یادگیری‌هایی ارائه می‌دهد که از تکامل الهام گرفته شده است. بعضی مواقع فرضیه‌ها با رشته بیت‌هایی نمایش داده می‌شوند که تفسیر این رشته بیت‌ها به مسئله وابسته است، با این وجود فرضیه‌ها ممکن است حتی به صورت نشانه های نمادین[1] یا برنامه های کامپیوتری بیان شوند. جستجو در میان فرضیه‌ها با یک جمعیت، مجموعه و یا فرضیه های اولیه آغاز می‌شود. اعضای جمعیت فعلی توسط اعمال تولید مثل و جهشی که تصادفی انجام می‌شوند تغییر می‌یابند. این اعمال از اعمال مشابهی در تکامل زیستی الگو برداری شده‌اند. در هر مرحله فرضیه های جمعیت فعلی با معیاری به نام تناسب ارزیابی می‌شوند. توابعی که تناسب بیشتری داشته باشند متناسباً احتمال بیشتری برای انتخاب برای تولید مثل، جهش و یا حضور مستقیم در جمعیت نسل بعد خواهند داشت. الگوریتم‌های ژنتیک در مسائل یادگیری و غیر یادگیری زیادی به کار رفته‌اند. برای مثال، از آن‌ها برای یادگیری دسته قوانین برای کنترل ربات و بهینه سازی توپولوژی و یادگیری پارامتر های شبکه های عصبی استفاده می‌شود. در این فصل هم به الگوریتم‌های ژنتیک، که از فرضیه های رشته بیتی استفاده می‌کنند، و هم به برنامه نویسی ژنتیک، که فرضیه‌هایش برنامه های کامپیوتری‌اند، می‌پردازیم.

9.1 انگیزه

اساس انگیزه‌ی الگوریتم‌های ژنتیک[2] یا GA ها تکامل زیستی است. به جای استفاده از ترتیب کلی‌تری یا ترتیب ساده به پیچیده، الگوریتم‌های ژنتیک با توسعه دادن و ترکیب فرضیه های درست‌تر شناخته شده به فرضیه های درست جدید تری می‌رسند. در هر مرحله مجموعه ای از فرضیه‌ها ، جمعیت[3]، در نظر گرفته می‌شود. در هر مرحله کسری از جمعیت با فرزندانی که از بهترین فرضیه‌ها توسط عمل تولید مثل[4] ایجاد می‌شود جایگزین می‌شوند. چنین فرایندی یک سری آزمون و خطا را ایجاد می‌کند، در هر مرحله قدرت فرضیه‌ی ایجاد شده مورد آزمون قرار

[1] symbolic expressions
[2] Genetic algorithms
[3] population
[4] offspring

www.farabar.net

می‌گیرد. در تولید فرضیه های جدید معمولاً از خواصی از فرضیه های بهتر استفاده می‌شود. این نوع بررسی در الگوریتم‌های ژنتیک از چندین عامل انگیزه گرفته است:

- تکامل در طبیعت به عنوان یکی از متدهای موفق تطبیق پذیری پذیرفته شده است.
- الگوریتم ژنتیک می‌توانند فضاهای فرضیه ای را که ویژگی‌های متقابل پیچیده ای دارند جستجو کند. در چنین فرضیه‌هایی تغییر هر یک از خواص فرضیه تأثیر بسزایی در کل سازگاری فرضیه دارد.
- این الگوریتم‌ها هزینه‌ی استفاده از ابر کامپیوترها را ندارند، الگوریتم‌های ژنتیک را می‌توان به سادگی به چندین قسمت تقسیم و با کامپیوترهای مجزایی بررسی کرد.

در این بخش الگوریتم‌های ژنتیک را توضیح داده، کاربردهایشان و طبیعت جستجوی فضای فرضیه اییشان را بررسی می‌کنیم. همچنین به مقوله‌ی برنامه نویسی ژنتیک نیز می‌پردازیم، مقوله ای که در آن تمامی برنامه های کامپیوتری تا اندازه‌ی خاصی تکامل پیدا می‌کنند. الگوریتم‌های ژنتیک و برنامه نویسی ژنتیک هر دو زیر مجموعه‌ی محاسبات تکاملی[1] هستند. در قسمت آخر نیز سر تیترها در مطالعه‌ی تکامل زیستی، از جمله اثر بالدوین را بررسی می‌کنیم و رابطه‌ی بین تکامل افراد و تکامل کل جمعیت را نیز بررسی خواهیم کرد.

9.2 الگوریتم‌های ژنتیک

در این مسئله که توسط الگوریتم‌های ژنتیک مطرح می‌شود، مرحله‌ی اول پیدا کردن فضایی از فرضیه های کاندید است تا در بین آن‌ها فرضیه‌هایی با بهترین عملکرد را پیدا کنیم. در الگوریتم‌های ژنتیک تعریف "بهترین فرضیه" از این قرار است: فرضیه ای که معیار های پیش تعریف شده عددی‌ای را که تابع تناسب[2] نامیده می‌شود را برای مسئله‌ی موجود بهینه کند (حداکثر یا حداقل). برای مثال، اگر کار یادگیری تخمین یک تابع مجهول با نمونه های آموزشی ورودی و خروجی آن است، تابع تناسب می‌تواند به صورت دقت فرضیه بر روی نمونه های آموزشی تعریف شود. یا اگر هدف یادگیری روشی برای شطرنج بازی کردن است تابع تناسب می‌توان تعداد برد هر فرد در مقابل افراد دیگر در همان جمعیت باشد.

با اینکه الگوریتم‌های ژنتیک در کاربرد های متفاوتی به کار می‌روند و در جزئیات با هم متفاوتند اما معمولاً در ساختار های ذیل مشابهند: در هر مرحله الگوریتم مجموعه ای از فرضیه‌ها را به نام جمعیت تغییر می‌کند، در هر مرحله تمامی فرضیه های جمعیت با تابع تناسب مورد بررسی قرار می‌گیرند، جمعیت جدید با انتخاب تصادفی چند فرضیه از فرضیه های بهتر جمعیت ایجاد می‌شود، تعدادی از این فرضیه‌ها مستقیماً به نسل بعد منتقل می‌شوند و بقیه در تولید مثل فرضیه های جدید از طریق اعمال ژنتیکی‌ای چون تولید مثل[3] و جهش[4] مورد استفاده قرار می‌گیرند.

حالت کلی الگوریتم‌های ژنتیک در جدول 9.1 آورده شده است. ورودی‌های این الگوریتم شامل تابع تناسب، برای رده بندی فرضیه‌ها، مقدار آستانه، برای تشخیص میزان تناسب و پایان دادن به الگوریتم، تعداد جمعیتی که باید باقی بمانند و پارامترهایی مربوط به تشکیل جمعیت‌های موفق است: درصدی از جمعیت که جایگزین می‌شوند و سرعت جهش.

---

[1] evolutionary computation
[2] fitness function
[3] crossover
[4] mutation

GA(Fitness,Fitness_treshold,p,r,m)

Fitness: تابعی که به فرضیه‌ها مقداری برای ارزیابی نسبت می‌دهد.

Fitness_treshold: مقدار آستانه ای که شرط پایانی را مشخص می‌کند

p: تعداد فرضیه ای که باید در هر جمعیت باشد

r: نسبتی از جمعیت که در هر مرحله با استفاده از تولید مثل جایگزین می‌شوند

m: ضریب جهش

- مقدار دهی اولیه: جمعیت P را با فرضیه های اتفاقی‌ای ایجاد کن.
- ارزیابی: برای هر فرضیه‌ی h در P مقدار Fitness(h) را محاسبه کن.
- تا زمانی که $\max_h Fitness(h) < Fitness_threshold$ حلقه ی زیر را اجرا کن.

نسل جدید P_s را ایجاد کن:

1. انتخاب: با توزیع احتمال زیر (1-r)p عضو از P را به P_s اضافه کن.

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{h=1}^{p} Fitness(h_j)}$$

2. تولید مثل: با توزیع احتمال بالا را به $\frac{r.p}{2}$ جفت فرضیه انتخاب کن. برای هر جفت $< h_1, h_2 >$ با استفاده از عمل تولید مثل دو فرزند ایجاد کن و در مجموعه‌ی P_s قرار بده.
3. جهش: m عضو از P_s را با توزیع احتمال یکنواخت انتخاب کن و یکی از بیت‌های نمایشش را به دلخواه عوض کن.
4. تغییر: $P \leftarrow P_s$
5. ارزیابی: برای هر فرضیه‌ی h در P مقدار Fitness(h) را محاسبه کن.

- فرضیه ای در P که بالاترین تناسب را دارد را خروجی بده.

جدول 9.1 حالت کلی الگوریتم‌های ژنتیک.

در انتها جمعیتی با p فرضیه باقی می‌ماند. در هر بار تکرار حلقه جمعیت موفق‌تر P_s با انتخاب احتمالی فرضیه‌ها و اضافه کردن فرضیه های جدید شکل می‌گیرد. فرضیه های جدید از عمل تولید مثل بر روی جفت فرضیه‌ها ایجاد می‌شوند و بر روی بعضی از فرضیه‌ها نیز عمل جهش اتفاق می‌افتد. این فرایند تا زمانی که فرضیه با متناسب مورد نظر ایجاد شود ادامه دارد. اعمال تولید مثل و جهش معمول در جدولی در ادامه‌ی بحث آورده شده‌اند.

توجه داشته باشید که در این الگوریتم هر بار اجرای حلقه‌ی اصلی نسلی جدید از فرضیه‌ها را از روی جمعیت فعلی ایجاد می‌کند. در گام اول، تعداد خاصی از فرضیه‌ها از جمعیت فعلی انتخاب می‌شوند تا نسل بعدی را تشکیل دهند. چنین فرضیه‌هایی با توزیع احتمال زیر انتخاب می‌شوند:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{h=1}^{p} Fitness(h_j)} \tag{9.1}$$

پس احتمال انتخاب هر فرضیه با تناسبش رابطه‌ی مستقیم و با مجموع تناسب فرضیه های رقیب رابطه‌ی عکس دارد.

زمانی که اعضای نسل فعلی برای تشکیل نسل بعد انتخاب می‌شوند، از طریق عمل تولید مثل فرضیه های جدیدی نیز ساخته و اضافه می‌شوند. عمل تولید مثل، انتخاب دو فرضیه به عنوان والدین و ترکیب قسمت‌های مختلف آن‌هاست، در قسمت‌های آینده عمل تولید مثل را مفصلاً توضیح خواهیم داد. فرضیه های والد به صورت تصادفی از جمعیت فعلی با توزیع احتمال رابطه‌ی 9.1 انتخاب می‌شوند. بعد از تولید مثل و ایجاد اعضای جدید، نسل جدید جمعیت تعداد کافی اعضا را خواهد داشت. سپس کسر خاصی از این اعضا به صورت اتفاقی انتخاب می‌شوند و عمل جهش بر روی آن‌ها انجام می‌گیرد.

این الگوریتم ژنتیک جستجویی موازی و ستونی[1] در میان فرضیه‌هایی که تناسب بهتری دارند انجام می‌دهد. در قسمت‌های بعدی، فرضیه‌ها و اعمال ژنتیک را دقیق‌تر بررسی خواهیم کرد.

9.2.1 معرفی فرضیه‌ها

فرضیه‌هایی که در الگوریتم‌های ژنتیک استفاده می‌شوند معمولاً به صورت رشته های بیت نمایش داده می‌شوند تا بتوان اعمال تولید مثل و جهش را به راحتی روی آن‌ها انجام داد. این نمایش فرضیه‌ها می‌تواند بسیار پیچیده باشد. برای مثال، مجموعه‌ی دستورهای if-then را می‌توان با در نظر گرفتن کدی برای هر دستور به راحتی در این نمایش نشان داد. مثال‌های انواع نمایش این نوع دستورها در (Holland 1986)، (Grefenstette 1988) و (DeJong 1993) و دیگر مقالات آورده شده است.

برای تصور اینکه چگونه می‌توان دستور های if-then را با رشته بیت‌ها نمایش داد، ابتدا طرز نمایش یک شرط[2] را بر روی مقدار یک ویژگی در نظر بگیرید. برای مثال، ویژگی outlook را که سه مقدار Sunny، Rainy و Cloudy را می‌تواند بپذیرد در نظر بگیرید. یکی از ساده‌ترین راه‌های نمایش چنین ویژگی‌ای قرار دادن 1 در مکان مربوطه‌ی هر یک از این مقادیر است. مثلاً رشته‌ی 010 نشان می‌دهد که outlook=Cloudy است. به طور مشابه اگر رشته 011 باشد نشان دهنده‌ی این است که outlook یکی از دو مقدار ممکن را دارد (outlook=Rainy v Cloudy). توجه داشته باشید که کلی‌ترین فرضیه در چنین نمایشی 111 است که نشان دهنده‌ی این است که مقدار ویژگی برای فرضیه تفاوتی نمی‌کند.

با توجه به این متد برای نمایش ویژگی‌ها، می‌توان روابط فصلی را با بهم پیوستن رشته‌ها نشان داد. برای مثال ویژگی دوم Wind را با دو مقدار Strong و Weak در نظر بگیرید. فرضیه ای مثل فرضیه‌ی

$$(\text{outlook} = \text{Cloudy} \vee \text{Rainy}) \wedge (\text{Wind} = \text{Strong})$$

را می‌توان به راحتی به شکل پنج بیت زیر نشان داد:

Outlook	Wind
011	10

حکمی (PlayTennis = yes) را می‌توان به همین صورت نشان داد. پس کل رابطه را می‌توان با سرهم کردن بیت‌های شرط و حکم نشان داد. مثلاً رابطه‌ی

[1] beam search
[2] constraint

IF Wind = Strong THEN PlayTennis = yes

به صورت زیر نمایش داده خواهد شد:

Outlook	Wind	PlayTennis
011	10	10

که در آن سه بیت اول نشان دهنده‌ی عدم اهمیت Outlook و دو بیت بعدی وضعیت Wind و دو بیت آخر حکم را نشان می‌دهد (در اینجا فرض کرده‌ایم که PlayTennis دو مقدار yes و no را می‌تواند داشته باشد). توجه داشته باشید که نمایش بیتی قوانین برای هر ویژگی در فرضیه یک زیر رشته[1] در نظر می‌گیرد حتی اگر آن ویژگی جزو ویژگی‌های شرط نباشد. همین امر باعث می‌شود که همیشه فرضیه‌ها، رشته های بیتی‌هایی با طول ثابت باشند که جای هر یک از زیر رشته‌ها نیز در آن ثابت است. با چنین نمایشی برای قوانین، می‌توان دسته قوانین[2] را با پشت سر هم آوردن چنین نمایشی از قوانین ساخت.

بد نیست که در طراحی رشته بیت‌ها برای کد سازی فضای فرضیه ای برای هر مقدار مجاز هر ویژگی یک بیت در نظر بگیریم تا فرضیه‌ها قابلیت ارتجاع داشته باشند. برای درک بهتر، توجه دارید که در بالا در یکی از قوانین ذکر شده (11 10 111) هیچ اطلاعاتی در باره ی ویژگی PlayTennis به ما نمی‌دهد. اگر بخواهیم از چنین نمونه‌هایی دوری کنیم می‌توانیم نمایش دیگری را انتخاب کنیم (مثلاً فقط یک بیت را به بازی تنیس اختصاص داده و برای مقدار بله 1 و برای مقدار خیر 0 را در نظر بگیریم)، اما بهتر است اعمال ژنتیکی را طوری تغییر دهیم که چنین فرضیه‌هایی ایجاد نشوند یا به روش دیگر تناسب پایینی را به چنین فرضیه‌هایی نسبت دهیم.

در بعضی الگوریتم‌های ژنتیک، فرضیه‌ها با نشانه های نمادین (به جای رشته بیت‌ها) نمایش داده می‌شوند. برای مثال، در قسمت 9.5 الگوریتم ژنتیکی را بررسی خواهیم کرد که فرضیه‌هایش برنامه های کامپیوتری هستند.

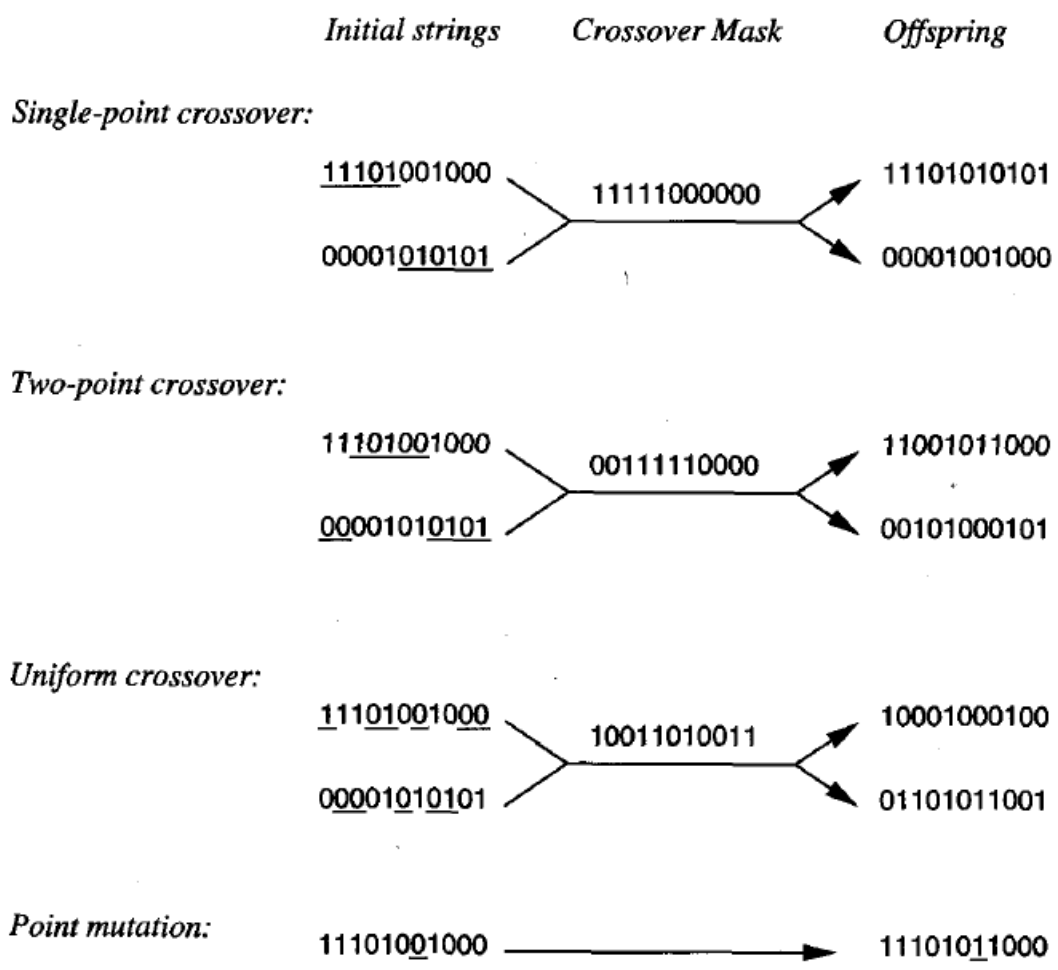
9.2.2 اعمال ژنتیکی

ایجاد جمعیت‌های جدید در الگوریتم‌های ژنتیک با استفاده از اعمالی چون تولید مثل و جهش انجام می‌شود. اعمال متداول در الگوریتم‌های ژنتیک برای ایجاد رشته بیت‌های جدید در جدول 9.1 آورده شده است. این عملگرها متناسب با اعمال ژنتیکی ایده آل سیستم‌های تکامل زیستی طراحی شده‌اند. معمول‌ترین این اعمال تولید مثل و جهش است.

عمل تولید مثل دو فرزند را از دو والد با ترکیب کپی بیت‌هایشان ایجاد می‌کند. هر بیت در موقعیت i ام فرزندان کپی بیتی در همان موقعیت یکی از والدینشان است. انتخاب اینکه کدام بیت را از کدام والد انتخاب می‌کنیم توسط رشته ای دیگر به نام نقاب تولید مثل[3] انجام می‌شود می‌شود. برای تصور، تولید مثل تک نقطه ای[4] که در بالای جدول 9.2 آمده است را در نظر بگیرید. به فرزند بالایی توجه کنید، این فرزند تمامی پنج بیت اول را از والد اول و شش بیت باقی مانده را از والد دوم به ارث برده است، زیرا که نقاب تولید مثلش 11111000000 بوده است. فرزند دوم نیز از عکس همان نقاب استفاده کرده است، به همین دلیل فرزند دوم بیت‌هایی را کپی کرده که فرزند اول از آن‌ها استفاده ای نکرده بود. همیشه در تولید مثال تک نقطه ای ابتدای نقاب n بیت 1 و بقیه 0 هستند. به همین دلیل فرزندان n بیت اول از یکی از والدین و

[1] substring
[2] sets of rules
[3] crossover mask
[4] single-point crossover

بقیه از والد دیگر به ارث ببرند. هر بار که تولید مثل تک نقطه ای به کار برده می‌شود، ابتدا نقطه‌ی تولید مثل n به طور تصادفی انتخاب می‌شود سپس نقاب ساخته شده و در تولید مثل به کار برده می‌شود.



جدول 9.2 اعمال معمول در الگوریتم‌های ژنتیک.

این اعمال فرزندانی را با استفاده از نمایش بیتی فرضیه‌ها ایجاد می‌کنند. عمل تولید مثل دو فرزند را از دو والد تولید می‌کند، نقاب معلوم می‌کند که هر فرزند کدام بیت‌ها را به ارث ببرد. جهش با تغییر یکی از بیت‌ها از یک والد یک فرزند ایجاد می‌کند.

در تولید مثل دو نقطه ای[1] فرزندان قسمتی از وسط بیت‌ها را از یکی از والدین و بقیه را از والدی دیگر دریافت می‌کنند. به عبارت دیگر نقاب تولید مثل دو نقطه ای رشته بیتی است که با n_0 بیت 0 شروع شده و با n_1 بیت 1 ادامه پیدا می‌کند و بقیه بیت‌ها نیز 0 خواهد بود. هر بار که تولید مثل دو نقطه ای به کار می‌رود ابتدا دو عدد n_0 و n_1 به صورت تصادفی انتخاب شده سپس نقاب لازم ایجاد و تولید مثل انجام می‌شود. مثلاً در مثالی که در جدول 9.2 آمده $n_0 = 2$ و $n_1 = 5$ است. مشابه دیگر تولید مثل‌ها، فرزند دوم با جابجا کردن نقش دو والد ایجاد می‌شود.

[1] two-point crossover

تولید مثل یکنواخت[1] به طور یکنواخت بیت‌های والدین را با هم ترکیب و فرزندان را ایجاد می‌کند (همان طور که در جدول 9.2 نیز نشان داده شده است). در چنین تولید مثلی نقاب به صورت کاملاً تصادفی ایجاد می‌شود (هر بیت نقاب به طور تصادفی و مستقل از بقیه بیت‌ها انتخاب می‌شود).

علاوه بر اعمال تولید مثل که از دو والد برای ایجاد فرزند استفاده می‌کنند، عمل دیگری که از یک والد فرزند ایجاد می‌کند نیز وجود دارد. در کل، جهش تغییر کوچکی در فرضیه‌هاست که با تغییر یک بیت به وجود می‌آید. در بعضی سیستم‌ها بجای الگوریتم جدول 9.1 از الگوریتم‌هایی استفاده می‌شود که بعد از تولید مثل جهش را اعمال می‌کنند.

بعضی از سیستم‌های ژنتیکی اعمال دیگری را نیز به کار می‌گیرند که منحصر به طرز نمایش فرضیه‌هایشان است. برای مثال، (Grefenstette 1991) سیستمی را معرفی می‌کند که دسته قوانینی را برای کنترل ربات یاد می‌گیرد، این سیستم علاوه بر اعمال تولید مثل و جهش، از عملی اضافی برای خاص کردن قوانین کمک می‌گیرد. (Janikow 1993) نیز سیستمی را معرفی می‌کند که دسته قوانینی را با استفاده از اعمال کلی سازی و خاص سازی قوانین از لحاظ‌های مختلف به کار می‌گیرد. (برای مثال با تغییر یکی از شروط رابطه‌ی if-then با عامل "فرقی نمی‌کند").

9.2.3 تابع تناسب و انتخاب

تابع تناسب معیاری برای ترتیب کردن فرضیه‌ها و انتخاب تصادفی آن‌ها برای حضور در نسل بعدی جمعیت است. اگر هدف یادگیری دسته قوانین باشد، تابع تناسب قسمتی خواهد داشت تا دقت قانون را بر روی نمونه های آموزشی موجود اندازه گیری کند. بعضی موارد معیار های دیگری نیز در تابع تناسب تأثیر گذارند (مثلاً پیچیدگی قانون یا کلی بودن قانون). در کل برای فرضیه های رشته بیتی‌ای که فرایند پیچیده ای دارند (مثلاً رشته بیتی که از تعداد زیادی قانون زنجیروار if-then تشکیل یافته تا یک دستگاه رباتیک را کنترل کند)، تابع تناسب کارایی کلی فرضیه را در نظر می‌گیرد.

در حالت کلی در الگوریتم ژنتیکی که در جدول 9.1 آورده شده، احتمال انتخاب یک فرضیه با تناسب خودش نسبت مستقیم و با تناسب دیگر فرضیه های جمعیت فعلی نسبت عکس دارد (رابطه‌ی 9.1). این متد را گاهی انتخاب نسبی تناسبی[2] و یا رولت[3] می‌نامند. متد های دیگری نیز برای انتخاب فرضیه‌ها بر حسب تناسبشان ارائه شده است. برای مثال، در انتخاب مسابقه ای[4] ابتدا دو فرضیه به صورت اتفاقی از جمعیت فعلی انتخاب می‌شوند. سپس با احتمالی از پیش تعریف شده مثل p فرضیه‌ی متناسب‌تر و با احتمال (1-p) فرضیه ای که تناسب کمتری دارد انتخاب می‌شود. با چنین انتخابی معمولاً به جای اینکه فرضیه های متناسب‌تر انتخاب شوند فرضیه های گوناگونی انتخاب می‌شوند (Goldberg and Deb 1991). در متد دیگری که انتخاب رتبه ای[5] نامیده می‌شود، در ابتدا تمامی فرضیه های جمعیت فعلی متناسب با تناسبشان ترتیب شده و سپس متناسب با رتبه‌هایشان احتمالی برای انتخاب شدن می‌گیرند (و نه متناسب با تناسبشان).

---

[1] uniform crossover
[2] fitness proportionate selection
[3] roulette wheel selection
[4] tournament selection
[5] rank selection

9.3 یک مثال

می‌توان به الگوریتم ژنتیک به دید یک متد بهینه برای جستجوی فضای بزرگی از اشیاء با هدف پیدا کردن متناسب‌ترین فرضیه‌ها (بر اساس تابع تناسب) نگاه کرد. با این وجود هیچ تضمینی نیست که خروجی این الگوریتم‌ها همیشه متناسب‌ترین فرضیه باشد، فقط می‌توان گفت که معمولاً خروجی این الگوریتم‌ها فرضیه‌هایی با تناسب بالاست. الگوریتم‌های ژنتیک در مسئله‌هایی در خارج قلمروی یادگیری ماشین مثل طراحی مدار[1] و برنامه ریزی مغازه داری[2] نیز به کار گرفته شده‌اند. در داخل قلمرو یادگیری ماشین نیز هم در تخمین توابع و هم در مسائلی همچون انتخاب نوع شبکه های عصبی به کار رفته‌اند.

برای تصور بهتر از کاربرد الگوریتم‌های ژنتیک در یادگیری مفهوم، در اینجا به طور خلاصه به سیستم GABIL که توسط (DeJong 1993) معرفی شده می‌پردازیم. در GABIL از الگوریتم ژنتیک برای یادگیری یک مفهوم منطقی با قانون‌هایی که قانون‌های فصلی گزاره ای[3]، استفاده شده است. در آزمایشات انجام شده بر روی مسائل مختلف یادگیری مفهوم، GABIL قدرت تامیم قابل توجهی داشته است. این قدرت تامیم را بعداً با قدرت تامیم الگوریتم‌های C4.5 و AQ14 مقایسه می‌کنیم. در این تحقیق هم از مسائل مصنوعی و هم از نمونه های واقعی در تشخیص سرطان سینه برای مشاهده‌ی قدرت تامیم استفاده شده است.

الگوریتم به کار رفته در GABIL همان الگوریتم جدول 9.1 است. در تحقیقات (DeJong 1993)، پارامتر r، کسری از جمعیت را که به نسل بعدی منتقل می‌شوند 0.6 و پارامتر m، ضریب جهش 0.001 بوده است. (چنین شرایطی، شرایطی متداول محسوب می‌شوند). و تعداد اعضای جمعیت نیز بسته به مسئله بین 100 تا 1000 بوده است.

اطلاعات خاص الگوریتم ژنتیک بکار رفته در GABIL مختصراً به شرح زیر است:

- **نمایش.** هر فرضیه در GABIL مجموعه ای از گزاره های فصلی که در قسمت 9.2.1 مفصلاً به آن‌ها پرداختیم است. در کل، فضای فرضیه ای از قانون‌های گزاره ای ای که از روابط فصلی بر روی تعداد خاصی از ویژگی‌ها تشکیل شده است. برای نمایش یک دسته قانون رشته بیت‌های قانون‌های مختلف به پشت هم می‌آیند. برای تصور، فرض کنید که فضای فرضیه ای داریم که دو ویژگی منطقی به نام‌های a_1 و a_2 دارند و تابع هدف نیز یک تابع منطقی به نام c است پس دو قانون زیر

$$IF\ a_1 = T \wedge a_2 = F \text{ Then } c = T\ ;\ \text{IF } a_2 = T \text{ Then } c = F$$

به فرم نشان داده شده نشان داده می‌شوند:

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

توجه دارید که طول رشته متناسب با تعداد قوانین بکار رفته در فرضیه تغییر می‌کند. با متغیر بودن طول نمایش رشته‌ی بیت لازم می‌شود که عمل تولید مثل متناسب با آن تغییر کند:

[1] circuit layout
[2] job-shop scheduling
[3] disjunctive set of propositional rules

- **اعمال ژنتیکی.** GABLIL از همان تعریف جهش که در جدول 9.2 آمده بود استفاده می‌کند، عمل تولید مثل نیز یک تولید مثل دو نقطه ای ساده است که در جدول 9.2 نیز توضیح داده شده بود. در کل برای تطبیق با این حقیقت که طول رشته‌ها متغیر است و همچنین برای اینکه بیت‌ها در فرزندان نیز در جای خود باشند، از روش ذیل استفاده می‌شود: برای انجام چنین تولید مثلی از دو والد ابتدا نقاط تولید مثل به صورت تصادفی انتخاب می‌شوند. اگر d_1 و d_2 دو فاصله‌ی نقاط از چپ و راست رشته بیت‌ها (هر یک از قوانین نه کل فرضیه) باشد، مسئله‌ی مهم این است که این نقاط برای والد دوم باید d_1 و d_2 مشابهی داشته باشند، مثلاً اگر دو والد به شکل

	a_1	a_2	c	a_1	a_2	c
h_1:	10	01	1	11	10	0

و

	a_1	a_2	c	a_1	a_2	c
h_2:	01	11	0	10	01	0

باشند و اگر نقاط تولید مثل برای والد اول 1 و 8 باشد:

	a_1	a_2	c	a_1	a_2	c
h_1:	1[0	01	1	11	1]0	0

برای این نقاط دو مقدار $d_1 = 1$ و $d_2 = 3$ هستند. بنابراین احتمال‌های موجود برای انتخاب دو نقطه‌ی تولید مثل والد دوم با توجه به مقادیر محدود به <1,3>، <1,8> و <6,8> می‌شود. حال اگر برای والد دوم نیز <1,3> انتخاب شود داریم که:

	a_1	a_2	c	a_1	a_2	c
h_2:	0[1	1]1	0	10	01	0

پس دو فرزند به شکل‌های

	a_1	a_2	c
h_3:	11	10	0

و

	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
h_4:	00	01	1	11	11	0	10	01	0

خواهند بود.

همان طور که در مثال نیز نشان داده شد، این روش باعث می‌شود تا فرزندان بتوانند تعداد متغیری (نه الزاماً مساوی والدینشان) قانون داشته باشند. این روش همچنین تضمین می‌کند که تمامی فرضیه های تولید شده معنی دار هستند.

- **تابع تناسب.** تناسب هر فرضیه را بر اساس دقت دسته بندی نمونه های آموزشی می‌سنجند. در اینجا تابع استفاده شده برای پیدا کردن تناسب هر فرضیه به شکل زیر است:

$$Fitness(h) = \left(correct(h)\right)^2$$

در این رابطه correct(h) تعداد نمونه های آموزشی‌ای است که h درست دسته بندی می‌کند.

در آزمایش مقایسه‌ی رفتار GABIL و الگوریتم‌های یادگیری درختی‌ای مثل C4.5 و ID5R و قانون آموزش AQ14 که (DeJong 1993) مطرح کرده تفاوت‌های قابل مقایسه ای در عملکرد این سیستم‌ها بر روی مسائل مختلف دیده می‌شود. برای مثال، برای 12 مسئله‌ی ترکیبی، عملکرد الگوریتم GABILL 92.1٪ بود در حالی که دیگر سیستم‌ها عملکردی بین 91.2٪ تا 96.6٪ داشتند.

9.4 جستجوی فضای فرضیه ای

همان طور که در بالا نیز نشان داده شد، الگوریتم‌های ژنتیک با استفاده از یک متد جستجوی ستونی به دنبال فرضیه ای که تناسب حداکثر را داشته باشند می‌گردند. این جستجو با دیگر متد های یادگیری که در این کتاب آمده متفاوت است. مثلاً در Backpropagation در شبکه های عصبی این جستجو با تغییر اندک اندک در فرضیه انجام می‌شد، در حالی که در الگوریتم ژنتیک این تغییرات به شدت و ناگهانی است، همان طور که واضح است فرزندان ممکن است تفاوت بسیاری با والدین داشته باشند. توجه داشته باشید که در الگوریتم‌های ژنتیک احتمال اینکه در مینیمم نسبی به دام بیفتیم بسیار کم است (مشکل اصلی Backpropagation).

یکی دیگر از تفاوت‌های کاربردی الگوریتم ژنتیک مشکل تراکم[1] است. تراکم پدیده ای است که در آن افرادی که تناسب بالاتری دارند سریعاً تولید مثل می‌کنند و تمامی جمعیت را با کپی‌های مشابه خود پر می‌کنند. مشکل در اینجاست که با کم شدن تنوع سرعت تکامل نیز کاهش می‌یابد. راه‌های بسیاری برای مقابله با مشکل تراکم ارائه شده است. یکی از این راه‌ها تغییر تابع انتخاب است، مثلاً می‌توان از انتخاب مسابقه ای یا انتخاب رتبه ای به جای رولت استفاده کرد. استراتژی مشابهی نیز به نام اشتراک تناسب[2] وجود دارد که تناسب یک فرد را با افزایش افراد مشابه کم می‌کند. راه سوم محدود کردن اجازه‌ی افراد برای جفت گیری و تولید مثل است، حتی می‌توانیم در بین نمونه های مشابه تعدادی را حذف کنیم یا زیر گونه های[3] مشابه و متعدد را حذف کنیم. راه حل مشابه معلوم کردن فاصله‌ی فرضیه‌ها و اجازه‌ی جفت گیری به افراد نزدیک به هم است. بسیاری از این تکنیک‌ها از تکامل زیستی نشأت می‌گیرند.

9.4.1 تکامل جمعیت و تئوری الگو[4]

حال این سؤال مطرح است که آیا می‌توان ریاضی‌وار سیر تکامل جمعیت در الگوریتم ژنتیک در طول زمان را مشخص کرد. تئوری الگوی (Holland 1975) یک توصیف از تکامل جمعیت ارائه می‌کند. این توصیف مبتنی بر مفهوم الگو[5] است که دسته رشته بیت‌ها را توصیف می‌کند. دقیق‌تر اینکه یک الگو رشته ای از 0 و 1 و * است. هر الگو دسته ای از رشته کدها را مشخص می‌کند که همان 0 و 1 ها را دارند و مقدار * نیز برایشان مهم نیست. برای مثال الگوی 0*10 دسته رشته‌ی شامل دو رشته بیت 0010 و 0110 را نشان می‌دهد.

هر رشته بیت را می‌توان نمایشگر تمامی الگوهایی که آن‌را توصیف می‌کند دانست. برای مثال، 0010 را می‌توان نمایشگر 2^4 الگو مثل **00، 0*10 و **** و غیره دانست. به طور مشابه جمعیتی از رشته کدها را می‌توان با تعدادی الگو و تعداد اعضای متناسب با آن الگو مشخص کرد.

---

[1] Crowding
[2] fitness sharing
[3] subspecies
[4] Scheme Theorem
[5] Schema/pattern

تئوری الگو می‌تواند مشخصه پدیده تکامل جمعیت در الگوریتم ژنتیک را بر اساس تعداد نمونه های هر الگو بیان نماید. اگر m(s,t) تعداد نمونه های الگوی s در جمعیت در زمان t باشد (تکرار t ام الگوریتم)، تئوری الگو با توجه به m(s,t) و دیگر ویژگی‌های الگو و جمعیت و پارامتر های الگوریتم ژنتیک، m(s,t+1) را توصیف می‌کند.

تکامل جمعیت در الگوریتم ژنتیک وابسته به مرحله های انتخاب، جفت گیری و جهش است. بیایید فعلاً فقط تأثیر مرحله‌ی انتخاب را در نظر بگیریم. فرض کنید که f(h) میزان تناسب رشته بیت h، $\bar{f}(t)$ متوسط تناسب تمامی رشته بیت‌های جمعیت در زمان t ، n تعداد افراد جمعیت، $\hat{u}(s,t)$ متوسط تناسب تمامی اعضای الگوی s باشد، و می‌دانیم که h هم عضو الگوی s است و هم در زمان t در جمعیت حضور دارد $h \in s \cap p_t$.

حال می‌خواهیم مقدار m(s,t+1) را پیش بینی کنیم، این پیش بینی را با امید مقدار مذکور نشان می‌دهیم: E[m(s,t+1)] . با استفاده از رابطه‌ی تابع توزیع احتمال مطرح شده در جدول 9.1 می‌توانیم مقدار E[m(s,t+1)] را محاسبه کنیم. رابطه‌ی 9.1 را می‌توان با فرضیاتی که کردیم به شکل زیر بازنویسی کرد:

$$\Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)}$$

$$= \frac{f(h)}{n\bar{f}(h)}$$

حال اگر h را یکی از اعضای جمعیت جدید در نظر بگیریم طبق تابع توزیع احتمال، احتمال اینکه عضوی از s را انتخاب کنیم را خواهیم داشت:

$$\Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)}$$

$$= \frac{\hat{u}(s,t)}{n\bar{f}(h)} m(s,t) \qquad (9.2)$$

در نتیجه گیری دوم از این حقیقت استفاده کردیم که:

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

رابطه‌ی 9.2 احتمال این را نشان می‌دهد که فرضیه‌ی انتخاب شده توسط الگوریتم ژنتیک نمونه ای از s باشد. چون n انتخاب مستقل از هم صورت می‌گیرد امید تعداد اعضای انتخاب شده n برابر احتمال انتخاب هر یک از اعضا خواهد بود و خواهیم داشت:

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(h)} m(s,t) \qquad (9.3)$$

رابطه‌ی 9.3 نشان می‌دهد که امید تعداد نمونه های الگوی s در نسل t+1 با متوسط تناسب نمونه های الگوی s، $\hat{u}(s,t)$ رابطه‌ی مستقیم و با متوسط تناسب تمامی فرضیه‌ها در زمان t نسبت عکس دارد. پس می‌توانیم انتظار داشته باشیم که الگوهایی که متوسط تناسبشان بالای

متوسط تناسب کل است در نسل‌های بعدی نمونه های بیشتری را به خود اختصاص خواهند داد. اگر به الگوریتم ژنتیک به نگاه جستجویی در میان الگوها، همزمان با جستجو در میان افراد برای پیدا کردن متناسب‌ترین‌ها نگاه کنیم، رابطه‌ی 9.3 نشان می‌دهد که در طی زمان الگوهایی که تناسب بیشتری دارند رشد بیشتری خواهند داشت.

در عبارت بالا فقط مرحله‌ی انتخاب را در نظر گرفتیم، در حالی دو مرحله‌ی تولید مثل و جهش نیز باید در نظر گرفته شوند. تئوری الگو فقط اثر منفی احتمالی این اعمال ژنتیکی را در نظر می‌گیرد (برای مثال جهش ممکن است تعداد نمونه های موجود از الگوی s را کاهش دهد)، و در تولید مثل نیز فقط تولید مثل تک نقطه ای را در نظر می‌گیرد. فرم کامل نظریه‌ی الگو کران پایینی برای امید تعداد نمونه های الگوی s بیان می‌کند:

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(h)} m(s,t)\left(1 - \frac{p_c d(s)}{l-1}\right)(1-p_m)^{o(s)} \qquad (9.4)$$

در این رابطه p_c احتمال این است که عمل تولید مثل تک نقطه ای به هر فرد دلخواهی اعمال شود p_m نیز احتمال این است که بیتی از فردی دلخواه جهش کند. o(s) در این رابطه بیت‌های معلوم[1] الگوی s است، در شمارش این بیت‌ها فقط بیت‌های 0 و 1 شمرده می‌شوند و بیت‌های * شمرده نمی‌شوند. d(s) نیز در این رابطه فاصله‌ی بیت چپ‌ترین و راست‌ترین بیت معلوم s است. L نیز طول هر رشته بیت در جمعیت است. توجه می‌کنید که قسمت اول رابطه‌ی 9.4 همان رابطه‌ی 9.3 است که تأثیر مرحله‌ی انتخاب بر تئوری الگو است. قسمت بعدی رابطه اثر تولید مثل تک نقطه ای است، در کل این قسمت احتمال اینکه فرزندان هر نمونه‌ی بعد از تولید مثل در s باشند را مشخص می‌کند. و قسمت آخر رابطه نیز اثر مرحله‌ی جهش است، این قسمت نیز احتمال اینکه بعد از جهش هنوز نمونه عضو s باشد را مشخص می‌کند. توجه دارید که دو عمل تولید مثل تک نقطه ای و جهش تعداد بیت‌های معلوم الگو o(s) و فاصله‌ی بین بیت‌های معلوم d(s) را افزایش می‌دهند. بنابراین تئوری الگو به این نتیجه گیری می‌رسد که در کل، الگو های متناسب‌تر بیشتر رشد خواهند کرد، مخصوصاً الگوهایی که تعداد بیت معلوم کمی دارند (تعداد * هایشان بالاست)، و الگوهایی که این بیت‌های معلوم به هم نزدیک‌ترند.

تئوری الگو شاید کلی‌ترین توصیف از تکامل در الگوریتم ژنتیک‌ها باشد. تنها ضعف این تئوری این است که در آن اثر مثبت احتمالی تولید مثل و جهش یک چیز نادیده گرفته شده است. اخیراً نظریه های زیادی در مورد تکامل ارائه شده که بعضی از آن‌ها بر پایه‌ی مدل‌های زنجیروار مارکوو[2] و بعضی دیگر بر پایه‌ی مدل‌های مکانیزم‌های آماری هستند. برای اطلاعات بیشتر به (Whitley and Vose 1995) و (Mitchell 1996) مراجعه کنید.

9.5 برنامه نویسی ژنتیک

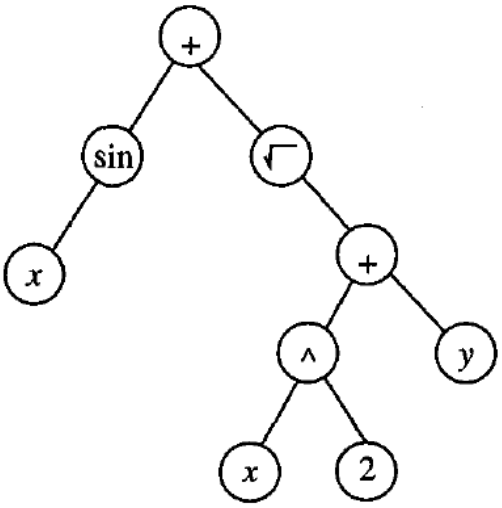
برنامه نویسی ژنتیک[3] نوعی از محاسبات تکاملی است که در آن اعضای جمعیت‌ها به جای رشته بیت‌ها برنامه های کامپیوتری هستند. (Koza 1992) روش برنامه نویسی ژنتیک را توصیف کرده و دسته‌ی بزرگی از برنامه های ساده را که توسط GP می‌توان یاد گرفت را معرفی می‌کند.

---

[1] defined bits
[2] Markov chain models
[3] genetic programming

9.5.1 نمایش برنامه‌ها

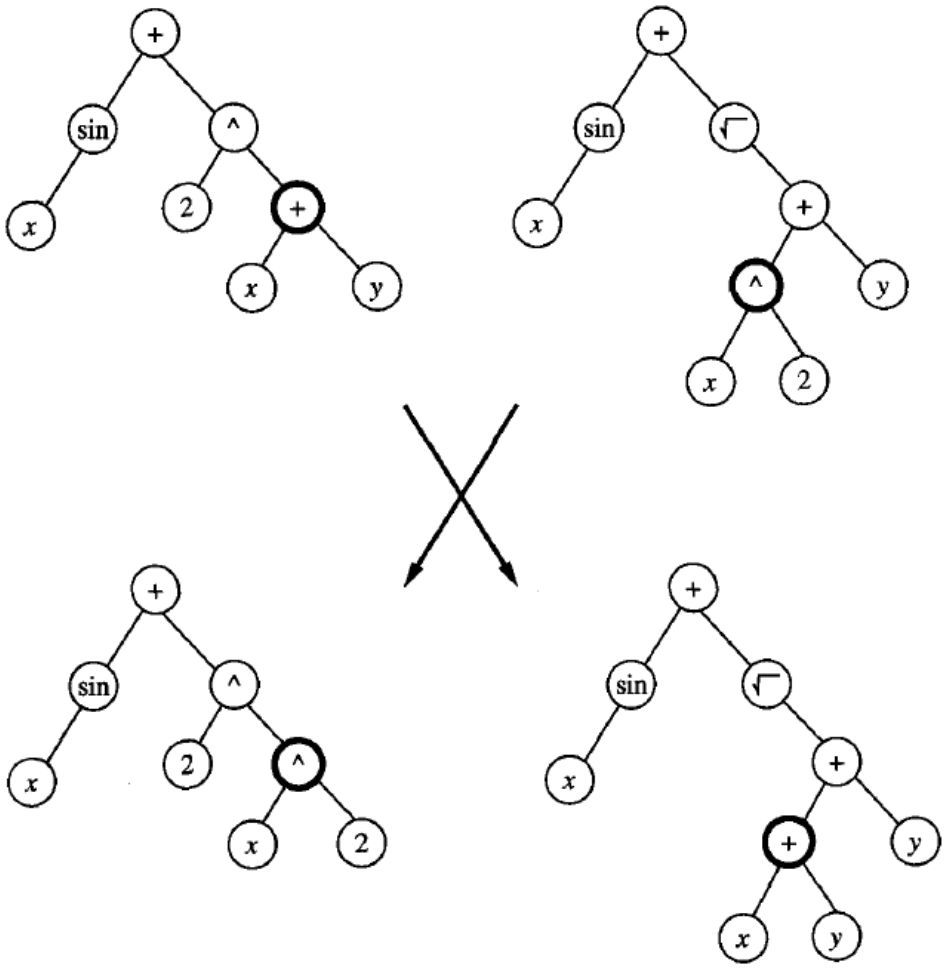
برنامه‌هایی که در GP مورد بحث قرار می‌گیرند معمولاً به صورت درخت‌هایی نمایش داده می‌شوند. هر تابع توسط یک گره در درخت و هر مقدار توسط یک یال مشخص می‌شود. برای مثال، شکل 9.1 تابع $\sin(x) + \sqrt{x^2 + y}$ را نشان می‌دهد. برای استفاده از برنامه نویسی ژنتیک باید ابتدا توابع پایه ای (مثل sin، cos، √، +، −، exp و ...) و ترمینال‌ها[1] (مثل x، y، اعداد ثابت، و...) را مشخص کرد. برنامه نویسی ژنتیک با استفاده از محاسبات تکاملی جستجویی در فضای بزرگ فرضیه های که برنامه های ساخته شده توسط توابع پایه است را فراهم می‌کند.



شکل 9.1 درخت نمایش برنامه‌ها در برنامه نویسی ژنتیک.
در برنامه نویسی ژنتیک برنامه های دلخواه با درخت‌های متناسبشان نشان داده می‌شوند.

درست مشابه الگوریتم‌های ژنتیک در قالب کلی برنامه نویسی ژنتیک نیز جمعیت‌ها (این بار به شکل برنامه های درختی) حضور دارند. در هر تکرار حلقه‌ی اصلی، نسلی جدید از افراد در سه مرحله‌ی انتخاب، تولید مثل و جهش ایجاد می‌شوند. تناسب هر برنامه‌ی جمعیت نیز با اجرای آن برای چند نمونه‌ی آموزشی بدست می‌آید. اعمال تولید مثل نیز با انتخاب و عوض کردن جای دو زیر شاخه‌ی درخت برنامه های والد انجام می‌گیرد. شکل 9.2 یک تولید مثل را نشان می‌دهد.

[1] terminal



شکل 9.2 عمل تولید مثل برای دو درخت والد (بالای شکل).
نقاط تولید مثل (گره های نشان داده شده) به تصادف انتخاب می‌شوند. و سپس زیر شاخه‌ها جای خود را عوض می‌کنند تا فرزندان (پایین شکل) شکل بگیرند.

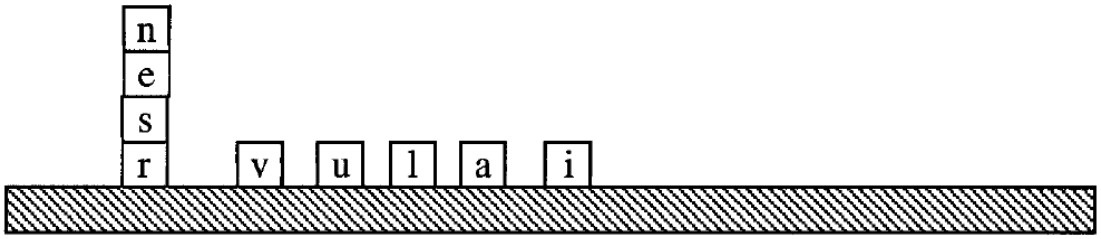
(Koza 1992) مجموعه ای GP را در تعدادی از مسائل به کار برد. در آزمایش‌های وی 10% از جمعیت فعلی با توجه به تناسبشان به صورت احتمالی مستقیماً به جمعیت نسل بعد انتقال داده می‌شدند. بقیه‌ی جمعیت نسل بعد از طریق عمل تولید مثل جفت برنامه های نسل فعلی، که دوباره به صورت احتمالی و با توجه به تناسبشان انتخاب شده بودند، ایجاد می‌شد. در این آزمایش‌ها از عمل جهش استفاده نشده است.

9.5.2 یک مثال

یکی از مثال‌های ارائه شده توسط (Koza 1992) درباره‌ی یادگیری الگوریتمی برای روی هم چیدن مکعب‌های شکل 9.3 بود. هدف پیدا کردن الگوریتمی کلی است که بدون تأثیر چینش اولیه‌ی مکعب‌ها، آن‌ها را طوری روی هم بچینند که در آخر در یک ستون کلمه‌ی "universal" را نمایش دهند. در هر حرکت تنها می‌توان یک مکعب را جابجا کرد. در کل دو حرکت مجاز وجود دارد، یکی اینکه یک مکعب را از سطح زمین به بالای ستونی ببریم و دیگری اینکه مکعب بالایی ستون را به زمین منتقل کنیم.

مثل اکثر مسئله های برنامه نویسی ژنتیک، انتخاب نحوه‌ی نمایش مسئله نقش بسیار مهمی در آسان شدن حل آن دارد. در نمایشی که Koza برای مسئله انتخاب کرد، سه ترمینال زیر را انتخاب کرد:

- CS (ستون فعلی)، که نشان دهنده‌ی حرف مکعب بالای ستون است، در صورت عدم وجود ستون فعلی F خواهد بود.
- TB (بالاترین مکعب درست)، که نشان دهنده‌ی حرف مکعب بالای ستون درست است، در چنین ستونی تمامی حروف در ترتیب درست قرار دارند.
- NN (نیاز بعدی)، حرفی است که در ترتیب درست بالای ستون TB باید قرار بگیرد تا کلمه‌ی "universal" را تشکیل دهد، اگر تعداد مکعب‌ها کافی نبود مقدار آن F خواهد بود.



شکل 9.3 مسئله چینش مکعب. هدف از برنامه نویسی ژنتیک پیدا کردن برنامه ای است که بدون توجه به ترتیب اولیه مکعب‌ها آن‌ها را به فرمی بچیند که کلمه "universal" را ایجاد کنند. مجموعه ای از 166 حالت اولیه برای تشخیص تناسب برنامه به کار می‌رود، (Koza 1992).

همان طور که دیده می‌شود، این انتخاب ترمینال‌های ورودی یک نمایش طبیعی برای توصیف این مسئله را ایجاد می‌کند. فرض کنید، در مقابل، به جای این ورودی، ورودی x و y تمامی مکعب‌های موجود است.

علاوه بر این ترمینال‌ها در برنامه های استفاده شده این توابع پایه ای به کار می‌روند:

- (MS x) (حرکت به روی ستون) اگر مکعب x روی زمین باشد این عمل آن‌را به بالای ستون می‌برد و مقدار T را بر می‌گرداند. در غیر این صورت عملی صورت نمی‌گیرد و مقدار F بر گردانده می‌شود.
- (EQ x y) (تساوی) اگر x و y مساوی باشند مقدار T برگردانده می‌شود، در غیر این صورت F برگردانده می‌شود.
- (NOT x) اگر x=F، مقدار T را برمی گرداند و اگر x=T مقدار F را بر می‌گرداند.
- (DU x y) ("Do Until") دستور x را تا T شدن y تکرار خواهد کرد.

برای اینکه سیستم بتواند تناسب برنامه های تولیدی را ارزیابی کند، Koza، 166 نمونه‌ی آموزشی از چینش اولیه‌ی مختلف مکعب‌ها با سختی‌های متفاوت ایجاد کرد. تناسب هر یک از برنامه‌ها تعداد نمونه های آموزشی حل شده توسط برنامه تلقی می‌شد. در ابتدا جمعیتی با 300 برنامه‌ی به تصادف ایجاد می‌شود. بعد از ده نسل سیستم به برنامه‌ی زیر رسید که تمامی نمونه های آموزشی را حل می‌کرد:

(EQ (DU (MT CS)(NOT CS)) (DU (MS NN)(NOT NN)))

توجه دارید که این برنامه از دو دستور DU یا همان "Do Until" استفاده کرده است. در دسته حرکت اول تمامی مکعب‌ها به روی زمین انتقال داده می‌شوند تا مجموعه‌ی ستون‌ها خالی شود. در دسته حرکت دوم پشت سر هم نیاز بعدی بر روی ستون درست قرار می‌گیرد. نقش دستور EQ اول در اینجا ایجاد قاعده ای دستوری برای ترکیب دو حلقه‌ی "Do Until" است.

جای تعجب است که این برنامه نویسی ژنتیک بعد تنها چند نسل به برنامه ای می‌رسد که تمامی 166 نمونه‌ی آموزشی را حل می‌کند. البته قدرت سیستم برای حل مسئله رابطه‌ی بسیار نزدیکی با ترمینال‌ها و توابع پایه ای و نمونه های آموزشی دارد.

9.5.3 نکاتی درباره‌ی برنامه نویسی ژنتیک

همان طور که در مثال بالا آورده شد، برنامه نویسی ژنتیک از الگوریتم‌های ژنتیک به تکامل برای برنامه های کامل کامپیوتری رسید. برخلاف اندازه‌ی بزرگ فضای فرضیه ای موجود که جستجو را سخت‌تر می‌کند، برنامه نویسی ژنتیک اثبات کرده است که می‌تواند نتایج خیره کننده ای در بعضی کاربردها بدهد. مقایسه ای بین برنامه نویسی ژنتیک و دیگر متد های جستجوی فضای برنامه های کامپیوتری، مثل hillclimbing و simulated annealing در کتاب (O'Reilly and Oppacher 1994) انجام شده است.

با این وجود که مثالی که در بالا آمده کمی ساده بود، (Koza 1996) و دیگر محققان کاربرد برنامه نویسی ژنتیک را در کاربردهای پیچیده تر مثل طراحی مدارهای فیلتر الکتریکی و دسته بندی مولکول‌های پروتئین‌ها نشان داد. مسئله‌ی طراحی فیلتر مدار نمونه ای از مسئله های نسبتاً پیچیده است. در این مسئله، برنامه‌ها تکامل می‌یابند تا بتوانند از مدارات پایه ای[1] مداری پیشرفته (مثل فیلتر) طراحی کنند. در این مسئله توابع پایه ای توابعی هستند که مدارات پایه ای را با اضافه یا کم کردن عناصر مداری و اتصالات تغییر می‌دهند. تناسب هر برنامه توسط برنامه های شبیه ساز (مثل SPICE) و از طریق مقایسه‌ی خروجی و خروجی مطلوب فیلتر محاسبه می‌شود. به عبارت دیگر امتیاز تناسب مجموع اندازه های تمام خطاهای بین مطلوب و خروجی مدار در 101 فرکانس متفاوت است. در این مسئله در هر نسل جمعیتی با 640000 عضو ایجاد می‌شد که 10٪ از جمعیت قبلی، 89٪ حاصل تولید مثل و 1٪ نیز حاصل جهش بودند. سیستم بر روی یک پردازنده‌ی 64 پایه ای موازی اجرا می‌شد. در نسل تصادفی مدارات آنقدر بی محتوا بودند که SPICE نمی‌توانست 98٪ شان را شبیه سازی کند. در نسل بعدی یا همان نسل اول این میزان به 84.9٪ و در نسل دوم به 75٪ کاهش یافت. در نسل‌های موفق این مقدار حتی تا 9.6٪ نیز کاهش یافت. امتیاز تناسب بهترین مدار در نسل اولیه 159 بود، بعد از 20 نسل این میزان به 39 و بعد از 137 نسل به 0.8 کاهش یافت. بهترین مدار بعد از 137 نسل ایجاد شد که رفتاری بسیار شبیه به رفتار مطلوب داشت.

در اکثر موارد، کارایی برنامه نویسی ژنتیک مستقیماً به طرز نمایش و انتخاب تابع تناسب وابسته است. به همین دلیل، قسمت عمده ای از تحقیقات به چگونگی انتخاب خودکار توابع پایه و الحاق آن‌ها برای بهبود توابع پایه ای اصلی می‌پردازد، این کار به سیستم اجازه می‌دهند که به صورت پویا توابع پایه‌ی برنامه‌ها را تغییر دهد. برای اطلاعات بیشتر به (Koza 1994) مراجعه کنید.

9.6 مدل‌های تکاملی و یادگیری

در بسیاری از سیستم‌های طبیعی، بسیاری از افراد در طول زندگی‌شان تطابق‌های مهمی را یاد می‌گیرند. در همین زمان، فرایند های زیستی و اجتماعی به گونه‌شان این امکان را می‌دهد تا در طول نسل‌ها تطابق پیدا کنند. یکی از سؤال‌های بسیار جالب در مورد سیستم‌های تکاملی این است که "رابطه‌ی بین یادگیری در طول عمر یک فرد با یادگیری در طول چندین نسل توسط تکامل چیست؟"

9.6.1 تکامل لامارکی

لامارک (Lamarck) محققی بود که در سال‌های آخر قرن نوزدهم می‌زیست. وی اعتقاد داشت که تکامل در طول نسل‌ها مستقیماً به تجربه های فردی در طول عمر وابسته است. در کل، وی اعتقاد داشت که تجارب یک فرد مستقیماً بر چینش ژنتیکی فرزندانش تأثیر می‌گذارد: اگر فردی در طول زندگی‌اش بیاموزد که از غذایی سمی پرهیز کند می‌تواند این آموزش را از طریق ژنتیکش به فرزندانش منتقل کند، پس

[1] simple fixed seed circuit

فرزندانش دیگر نیازی به یاد گرفتن چنین چیزی ندارند. این حدس بسیار جذاب است، زیرا که فرایند تکامل موثر تر خواهد شد بـه جـای اینکـه فقط یک آزمون و خطا باشد که تجارب فردی در طول عمر را فراموش کند (مثل نمونه ای که در الگوریتم‌های ژنتیک و برنامه نویسـی ژنتیـک بود). با وجود تمامی جذابیت‌های این نظریه، محققین عصر حاضر مدارک انکار ناشدنی‌ای برای رد کردن مدل لامارک دارند. نظریـه‌ی پذیرفتـه شده‌ی فعلی این است که نقشه‌ی ژنتیکی هر فرد، در واقع، هیچ تأثیری از تجارب طول زندگی والدینش نمی‌پذیرد. برخلاف این حقیقت زیستی، تحقیقات جدید کامپیوتری نشان داده که فرایند لامارکی گاهی می‌تواند کارایی الگوریتم‌های ژنتیکی کامپیوتری را بهبود ببخشد (برای اطلاعات بیشتر به Grefenstette 1991 و Ackley and Littman 1994 و Hart and Belew 1995 مراجعه کنید).

9.6.2 اثر بالدوین

با وجود اینکه مدل تکاملی لامارکی برای تکامل زیستی رد شد، مکانیزم‌های دیگری پیشنهاد شده که در آن‌ها یادگیری‌های فردی مسیر تکامل را می‌تواند عوض کند. یکی از این مکانیزم‌ها اثر بالدوین[1] است که نامش نیز از بالدوین (J. M. Baldwin 1896)، اولین کسی که این نظر را ارائه داد، گرفته شده است. اثر بالدوین بر پایه‌ی مشاهدات زیر نتیجه گیری شده:

- اگر گونه ای در حال تکامل در یک محیط در حال تغییر باشد، تمایل تکامل به سمتی خواهد بود تا افراد در طول عمـر خـود قابلیـت یادگیری بیشتری داشته باشند. برای مثال، با ظاهر شدن شکارچی جدید، افرادی که قابلیت یادگیری پرهیز از این شکارچی را دارند از افرادی که این قابلیت را ندارند موفق‌تر خواهند بود. پس قابلیت یادگیری به فرد اجازه می‌دهد تا جستجویی منطقه ای انجام دهـد تـا تناسبش را به حداکثر برساند. در مقابل، افرادی که این قابلیت فردی را ندارند و تناسبشـان نیـز توسـط ژنتیکشـان محـدود شـده در نقطه‌ی پایین‌تری نسبت به گروه اول قرار می‌گیرند.
- افرادی که می‌توانند بسیاری از آموزش‌های لازم را یاد بگیرند برای یادگیری کمتر به کد ژنتیکیشان متکی خواهند بود. نتیجه اینکـه این افراد تعداد جمعیت‌هایی با گوناگونی ژنتیکی بیشتری را تشکیل می‌دهند و از قابلیت‌های فردیشان بـرای غلبـه بـر کمبـود هـای ژنتیکی استفاده می‌کنند. چنین جمعیت ژنتیکی گوناگونی می‌تواند باعث تکامل بیشتر ژنتیکی شوند. پس، قابلیت افراد برای یـادگیری می‌تواند اثری غیر مستقیم بر افزایش سرعت تکامل کل جمعیت داشته باشد.

برای تصور فرض کنید که در محیط گونه‌ی خاصی تغییراتی جدید ایجاد می‌شود، مثلاً یک شکارچی اضافه می‌شود. چنین تغییری باعث می‌شود که فقط گونه‌هایی که قابلیت فردی پرهیز از شکارچی را دارند زنده بمانند. به تناسب میزان تطبیق پذیری فردی هـر فـرد در افـزایش جمعیـت، جمعیت می‌تواند انواع گوناگون‌تری از نمونه های ژنتیکی را در خود داشته باشد و سرعت فرایند های تکامـل را تسـریع ببخشـد . ایـن افـزایش سرعت تطبیق ممکن است باعث شود که گونه‌هایی به وجود بیایند که به طور ژنتیکی از شکارچی پرهیز کنند (مثلاً حسی غریزی برای پرهیز از بعضی مناطق). پس، اثر بالدوین مکانیزم‌های غیر مستقیمی را ایجاد می‌کند تا یادگیری‌های فردی نیز بـر سـیر تکامـل تـأثیر داشـته باشـند. بـا افزایش مقاومت[2] و گوناگونی ژنتیکی در میان گونه‌ها، یادگیری‌های فردی سرعت تکامل را سریع‌تر خواهند کرد و شانس ایجاد گونه‌هایی که به طور ژنتیکی در محیط جدید برتر از گونه های قبلی هستند افزایش می‌یابد.

تلاش‌های بسیاری برای مطالعه‌ی اثر بالدوین در مدل‌های محاسباتی شـده اسـت. بـرای مثـال، آزمـایش‌هـای (Hinton and Nowlan 1987) بر روی نمونه‌ها ساده ای از شبکه های عصبی انجام شد، در این آزمایش وزن‌های بعضی شبکه‌ها در تمام طول زندگی‌اش ثابت بود در حالی که بعضی دیگر قابل آموزش بودند. نقشه‌ی ژنتیکی هر فرد مشخص می‌کرد که شبکه آموزش پذیر باشد یا نه. در این آزمایش، در افـرادی

[1] Baldwin effect
[2] survivability

که نمی‌توانستند یاد بگیرند، بعد از نسل‌ها هیچ تکاملی در تناسب افراد ایجاد نشد، اما در افرادی که می‌توانستند یاد بگیرند تناسب جمعیت به شدت افزایش یافت. در نسل‌های ابتدایی تکامل جمعیت تعداد نسبی افرادی که می‌توانستند یاد بگیرند بسیار زیاد بود. با این وجود با ادامه یافتن تکامل تعداد شبکه‌هایی که وزن‌های ثابت داشتند و درست کار می‌کردند تمایل به افزایش یافت و جمعیت به سمت قابلیت ژنتیکی میل کرد تا اینکه بر توانایی‌های فردی اتکا بزند. تحقیقات دیگری نیز در مورد اثر بالدوین در الگوریتم ژنتیک‌ها توسط افرادی مثل (Belew 1990)، (Harvey 1993)، و (French and Messinger 1994) انجام شده است. بررسی کاملی نیز در (Mitchell 1996) انجام شده است. قسمت خاصی از مجله‌ی (journal Evolutionary Computation) نیز در این باره مطالب مفیدی دارد (Turney 1997).

9.7 موازی سازی الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک ذاتاً مناسب پیاده سازی موازی‌اند، و روش‌های متعددی برای موازی سازی آن‌ها پیدا شده است. روش‌های coarse grain برای موازی سازی و تقسیم جمعیت به گروه های مجزا افراد به نام بخش[1] به کار می‌روند. هر بخش به گره های محاسبه گر مختلف ارجاع می‌شود تا جستجویی استاندارد بر اساس الگوریتم ژنتیک در آنجا انجام بپذیرد. ارتباطات و جفت گیری مشترک بین بخش‌ها نیز اتفاق می‌افتند اما احتمال آن از جفت گیری درون بخش خیلی کمتر است. انتقال بین بخش‌ها نیز توسط فرایند مهاجرت[2] اتفاق می‌افتد، که در آن افرادی از یک بخش به بخش یا بخش‌های دیگری کپی یا انتقال داده می‌شوند. این فرایندها از جفت گیری و مهاجرت بین زیر جمعیت‌های موجود در گونه های زیستی الهام گرفته شده است. یکی از فواید این گونه تقسیم نسبت به سیستم‌های غیر موازی کاهش تراکم آن‌هاست، در مشکل تراکم بخاطر ظهور زودهنگام یکی از گونه های برتر تمامی نمونه‌ها به سمت آن متمایل می‌شوند و جامعه توسط این گونه پر می‌شود. نمونه‌ی الگوریتم ژنتیک‌هایی که از موازی سازی coarse-grain بهره می‌برند در (Tanese 1989) و (Cohoon 1987) و... آمده است.

در نقطه‌ی مقابل موازی سازی coarse-grain؛ موازی سازی fine-grain است که معمولاً یک پردازنده برای هر فرد در جمعیت در نظر می‌گیرد. و عمل جفت گیری فقط در افراد همسایه رخ می‌دهد. چندین نوع دیگری از تعریف همسایگی نیز ارائه شده است. در بعضی تعریف‌ها هر فرد فقط دو همسایه دارد و در بعضی دیگر نیز هر فرد با محدوده‌ی دایره ای خاص اطراف خود همسایه است. نمونه های چنین سیستم‌هایی در (Spiessens and Manderick 1991) آورده شده. منتخبی از تحقیقات الگوریتم ژنتیک نیز در (Stender 1993) آمده است.

9.8 خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر است:

- الگوریتم‌های ژنتیک (GA) جستجویی تصادفی، موازی و hill-climbing برای پیدا کردن فرضیه‌هایی که تابع از پیش تعریف شده‌ی تناسب را بهینه می‌کنند انجام می‌دهند.

---

[1] deme
[2] migration process

- این جستجوی الگوریتم‌های ژنتیک بر اساس تشابه با تکامل بیولوژیکی انجام می‌شود. جمعیت‌های گوناگون فرضیه های رقیب ایجاد می‌شوند. در هر حلقه، متناسب‌ترین اعضای جمعیت انتخاب شده و فرزندان از آن‌ها تولید می‌شوند، این فرزندان جایگزین اعضای کم تناسب تر جمعیت خواهند شد. فرضیه‌ها به صورت رشته بیت‌ها کد می‌شوند و توسط اعمال تولید مثل ترکیب شده و یا توسط جهش تغییر می‌یابند.
- الگوریتم‌های ژنتیک کار یادگیری را به عنوان نوعی بهینه سازی مطرح می‌کنند. در کل، عمل یادگیری در این دیدگاه پیدا کردن فرضیه های بهینه، برای تابع از پیش تعیین شده‌ی تناسب، است. این دیدگاه باعث می‌شود که بتوان دیگر روش‌های بهینه سازی مثل simulated annealing را نیز در یادگیری ماشین به کار برد.
- الگوریتم‌های ژنتیک معمولاً در مسائل بهینه سازی خارج محدوده‌ی یادگیری ماشین، مثل مسائل بهینه سازی طراحی به کار رفته‌اند. در یادگیری ماشین، الگوریتم‌های ژنتیک معمولاً برای کارهای یادگیری پیچیده (یادگیری دسته قوانین کنترل ربات و یادگیری برنامه های کامپیوتری) به کار می‌روند، در این مسائل هدف بهینه سازی تابع غیر صریحی از فرضیه‌هاست (مجموعه قوانین ربات را به طور صحیح کنترل کند).
- برنامه نویسی ژنتیک نسخه ای از الگوریتم‌های ژنتیک است که در آن فرضیه‌هایی که توسط کامپیوترها دست‌کاری می‌شوند، برنامه های کامپیوتری هستند، بجای رشته بیت‌ها. اعمالی چون تولید مثل و جهش به برنامه‌ها بجای رشته بیت‌ها تعمیم داده می‌شود. برنامه نویسی ژنتیک اثبات کرده که می‌تواند برنامه‌هایی برای کارهایی نظیر کنترل ربات Koza (1992) و تشخیص اشیا در تصاویر را انجام دهد Teller and Veloso (1994).

روش‌های محاسباتی مبتنی بر تکامل از روزهای اولیه‌ی علم کامپیوتر مورد بررسی قرار گرفتند (Box 1957 and Bledsoe 1961). روش‌های تکاملی مختلف بسیاری در دهه‌ی 1960 معرفی شد و در آن زمان مورد تحقیق بیشتر قرار گرفت. استراتژی‌های تکاملی، توسط (Rechengerg 1965,1973) برای بهینه کردن پارامتر های عددی در طراحی مهندسی طراحی شده و با کارهای Schwefel (1975, 1977, 1995) و دیگران ادامه یافت. برنامه نویسی تکاملی توسط Folgel, Owens, and Walsh (1966) به عنوان متدی برای ساخت ماشین‌های finite-state طراحی شد و توسط محققان (Fogel and Atmar 1993) عددی ادامه یافت. الگوریتم‌های ژنتیک، معرفی شده توسط Holland (1962, 1975) شامل مفهوم حفظ جمعیتی بزرگ از افراد و تاکید بر تولید مثل به عنوان عملگر کلیدی در چنین سیستم‌هایی می‌شد. الگوریتم‌های ژنتیک، معرفی شده توسط Koza (1992) استراتژی جستجوی الگوریتم‌های ژنتیک را به فرضیه‌ها برنامه های کامپیوتری اعمال می‌کند. با کاهش قیمت کامپیوترها و افزایش سرعتشان، علاقه به روش‌های تکاملی بیشتر می‌شود.

یکی از روش‌های استفاده از الگوریتم‌های ژنتیک یادگیر دسته قوانین است که توسط K. DeJong و دانشجویانش در دانشگاه Pittsburg است (Smith 1980). در این روش، هر مجموعه قوانین یکی از اعضای جمعیت رقابتی فرضیه‌هاست، همان طور که در سیستم GABIL در این فصل نیز توضیح داده شد. روش دیگری که در دانشگاه Michigan توسط Holland و دانشجویانش (Holland 1986) ایجاد شده، روشی است که در آن هر قانون عضوی از جمعیت است و جمعیت خود یک دسته قانون است. تصور زیستی از نقش جهش، جفت گیری، جفت گیری بین نژادی[1] و انتخاب در تکامل در Wright (1977) آورده شده است.

[1] cross-breeding

Mitchell (1996) و Goldberg (1989) دو کتاب مربوطه‌ی الگوریتم‌های ژنتیک هستند. Forrest (1993) نیز کتابی است که نگاه کلی‌ای از مسائل الگوریتم‌های ژنتیک را بررسی می‌کند، Goldberg (1994) نیز نگاه کلی‌ای از چندین کاربرد جدید الگوریتم‌های ژنتیک را در بر دارد. کتاب Koza (1992) نیز درباره‌ی برنامه نویسی ژنتیک منبع استاندارد تعمیم الگوریتم‌های ژنتیک برای تغییر برنامه های کامپیوتری است. کنفرانس‌های اولیه که در آن نتایج اولیه انتشار می‌شود کنفرانس International Conference on Genetic Algorithms است. دیگر کنفرانس‌های مربوطه شامل Conference on Simulation of Adaptive Behavior و the International Conference on Artificial Neural Networks and Genetic Algorithms و IEEE International Conference on Evolutionary Computation هستند. کنفرانس سالانه ای نیز درباره‌ی برنامه نویسی ژنتیک برگزار می‌شود (Koza et al. 1996b). The Evolutionary Computation Journal نیز یکی از منابع اخیر نتایج تحقیقات در این زمینه است. بسیاری از قسمت‌های the journal Machine Learning نیز به الگوریتم‌های ژنتیک اختصاص یافته است.

تمرینات

9.1 الگوریتم ژنتیکی طراحی کنید که قوانین دسته بندی عطفی را برای مفهوم PlayTennis مطرح شده در فصل 3 را یاد بگیرد. رشته های کد برای فرضیه‌ها و عملگرهای ژنتیک را دقیقا مشخص کنید.

9.2 نمونه‌ی ساده ای از الگوریتم ژنتیک تمرین 9.1 را پیاده سازی کنید. الگوریتم را با اندازه جمعیت‌های مختلف p، نسبت جایگزینی‌های مختلف r و ضریب جهش‌های مختلف m امتحان کنید.

9.3 برنامه‌ی ایجاد شده توسط برنامه نویسی ژنتیک در بخش 9.5.2 را به صورت درخت پیدا کنید. عملگر تولید مثل برنامه نویسی ژنتیک را می‌توان با استفاده از یک درخت به عنوان هر دو والد در نظر گرفت.

9.4 استفاده از برنامه نویسی ژنتیک را در پیدا کردن بردار وزن متناسب با یک شبکه‌ی عصبی مصنوعی را در نظر بگیرید (در کل شبکه ای تک سویه مشابه مواردی که در فصل 4 با backpropagation آموزش دادیم). شبکه ای 1×2×3 لایه ای و تک سویه را در نظر بگیرید. کد سازی‌ای برای وزن‌های شبکه ارائه دهید و مجموعه ای از اعمال ژنتیک مناسب روی این کدها را توصیف کنید. یک مزیت و یک مشکل استفاده از ژنتیک بجای backpropagation را برای آموزش شبکه های عصبی بیان کنید.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

اثر بالدوین	Baldwin effect
اشتراک تناسب	fitness sharing
انتخاب رتبه ای	rank selection
انتخاب مسابقه ای	tournament selection
انتخاب نسبی تناسبی	fitness proportionate selection
برنامه ریزی مغازه داری	job-shop scheduling

برنامه نویسی ژنتیک	genetic programming
بیت‌های معلوم	defined bits
تابع تناسب	fitness function
تراکم	Crowding
ترمینال	Terminal
تناسب	Fitness
مقاومت	Survivability
تولید مثل	crossover, offspring
تولید مثل تک نقطه ای	single-point crossover
تولید مثل یکنواخت	uniform crossover
تئوری الگو	Schema Theorem
جستجویی ستونی	beam search
جمعیت	Population
جهش	Mutation
دسته قوانین	sets of rules, classification rules
زیر رشته	Substring
زیر گونه‌ها	Subspecies
طراحی مدار	circuit layout
قانون‌های فصلی گزاره ای	disjunctive set of propositional rules
شرط	Constraint
الگو	Schema, pattern
الگوریتم‌های ژنتیک	Genetic algorithms
محاسبات تکاملی	evolutionary computation
مدارات پایه ای	simple fixed seed circuit
مدل‌های زنجیروار مارکوو	Markov chain models
مشکل تراکم	crowding problem
نشانه های نمادین	symbolic expressions
نقاب تولید مثل	crossover mask
نقشه‌ی ژنتیکی	Genetic makeup
نمایش نمادین	symbolic expressions
یادگیری ماشین	machine learning

Telegram: @Farabar_BI
www.farabar.net

فصل دهم: یادگیری دسته قوانین

یکی از راحتر ترین و قابل درک ترین فرم بیان فرضیه ها دسته قوانین if-then است. این فصل چندین الگوریتم برای یادگیری این دسته قوانین را بیان و بررسی می کند. یکی از مهم ترین حالات مسئله در یادگیری دسته قوانین با متغیر ها horn clause درجه اول[1] نامیده می شود، زیرا که دسته قوانین horn clase درجه اول را می توان به عنوان برنامه به زبان برنامه نویسی منطقی Prolog داد، یادگیری این دسته قوانین گاهی برنامه نویسی استقرایی منطقی[2] (ILG) نیز نامیده می شود. این فصل چندین روش مختلف برای یادگیری دسته قوانین، شامل روش مبتنی بر وارون کردن عملگر های نتیجه گیری[3] ثابت کننده ی تئوری[4] را بررسی می کند.

۱۰.۱ معرفی

در بسیاری از شرایط یادگیری تابع هدف استفاده از دسته قوانین if-then که با هم تابع یادگرفته شده را نشان می دهند بسیار مفید است. همانطور که در فصل ۳ نیز نشان داده شد، یکی از راههای یادگیری قوانین ساخت درخت تصمیم و تبدیل آن به دسته قوانین هم ارز است، به ازای هر برگ درخت یک قانون ساخته خواهد شد. روش دوم که در فصل ۹ نشان داده شد، استفاده از الگوریتم ژنتیک است، در این روش رشته بیتهایی متناسب با دسته قوانین ایجاد شده و از الگوریتم های ژنتیک برای جستجو ی فضای فرضیه ای تعریفی استفاده می شود. در این فصل ما الگوریتم های مختلفی که مستقیما دسته قوانین را یاد می گیرند و با الگوریتم های بالا در دو نکته کلیدی متفاوت اند را بررسی می کنیم. ابتدا اینکه این الگوریتم ها برای یادگیری دسته قوانین درجه اولی که متغیر دارند طراحی شده اند، اهمیت این نکته در قدرت بیان این قوانین

[1] first-order horn clase
[2] inductive logic programming
[3] deductive operators
[4] mechanical theorem provers

www.farabar.net

نسبت به دسته قوانین گزاره ای است. دوم اینکه الگوریتم های مطرح شده الگوریتم های ترتیبی[1] هستند که قوانین را به ترتیب و برای کامل کردن دسته قوانین موجود یاد می گیرند.

برای مثال، دسته قوانین درجه اول زیر را که باهم مفهوم هدف Ancestor (جَد) را نشان می دهند در نظر بگیرید.

IF Parent(x,y) THEN Ancestor(x,y)

IF Parent(x,z)∧Ancestor(z,y) THEN Ancestor(x,y)

توجه داشته باشید که قوانین بالا تابعی بازگشتی را که به سختی می توان با درخت تصمیم گیری یا نمایش های گزاره ای[2] نشان داد را نمایش می دهد. یکی از راههای پی بردن به قدرت نمایشی قوانین درجه اول توجه به هدف کلی زبان برنامه نویسی Prolog است. در Prolog برنامه ها دسته قوانین درجه اول، مشابه دو قانونی که در بالا ذکر شد، هستند (این نوع قوانین گاهی Horn clause نامیده می شوند). در واقع، قوانین بالا برنامه ای در زبان Prolog است که رابطه ی Ancestor را تشخیص می دهد. در این سبک، الگوریتم کلی ی یادگیری اینگونه دسته قوانین را می توان برنامه نویسی خودکار Prolog از نمونه های آموزشی دانست. در این فصل چنین الگوریتم هایی که دسته قوانین را از نمونه های آموزشی یاد می گیرند بررسی خواهیم کرد.

در عمل، سیستم های یادگیری که از دسته قوانین درجه اول استفاده می کنند در مسائلی چون یادگیری قوانین شیمی در طیف سنج جرمی (Buchanan 1976; Lindsay 1980)، یادگیری ساختار های جهش ژنتیکی (در رابطه با سرطان) (Srinivasan 1994) و یادگیری طراحی عناصر متناهی برای بررسی استرس در ساختار های فیزیکی (Dolsak and Muggleton 1992) به کار رفته اند. در هر یک از این کاربرد ها، فرضیه ای که معرفی می شود، ادعا هایی است که به راحتی توسط دسته قوانین درجه اول بیان می شوند، و توصیف این ادعا ها در بیان های گزاره ای بسیار سخت است.

در این فصل، ابتدا از الگوریتم هایی شروع خواهیم کرد که دسته قوانین گزاره ای را یاد می گیرند؛ دسته قوانین گزاره ای دسته قوانینی هستند که متغیر ندارند. الگوریتم هایی که برای جستجوی فضای فرضیه ای ی دسته قوانین فصلی در چنین شرایطی قابل درک اند. سپس الگوریتم هایی که دسته قوانین درجه اول یاد می گیرند را بررسی خواهیم کرد. در ادامه نیز دو روش کلی برای استقرا در برنامه نویسی منطقی و روابط اساسی بین استنباط استقرایی و استنتاجی را بررسی خواهیم کرد.

۱۰.۲ الگورتیم های ترتیبی

در این بخش خانواده ای از الگوریتم ها که دسته قوانین را بر اساس استراتژی یادگیری یک قانون و حذف داده های سازگار با آن قانون یاد می گیرند بررسی خواهیم کرد. چنین الگوریتم هایی، الگوریتم های ترتیبی نامیده می شوند. توضیح بیشتر اینکه، فرض کنید که یک زیر روال[3] به نام Learn-one-rule داریم که دسته ای از نمونه های مثبت و منفی را دریافت کرده و قانونی را خروجی می دهد که اکثر نمونه های مثبت و تعداد بسیار کمی از نمونه های منفی را می پوشاند. می خواهیم که دقت این قانون خروجی حداکثر شود، لازم نیست پوشش قانون

[1] sequential covering algorithms
[2] propositional representation
[3] subroutine

خروجی زیاد باشد. منظورمان از افزایش دقت قانون افزایش تعداد دسته بندی های درست آن است. با پذیرفتن پوشش کم، منظورمان این است که لازم نیست که قانون برای همه ی نمونه های آموزشی پیشبینی ای داشته باشد.

با داشتن زیر روال Learn-one-rule، یکی از ساده ترین راهها، دادن همه ی نمونه های آموزشی موجود به زیر روال و حذف تمامی نمونه های مثبتی تحت پوشش قانون خروجی و تکرار این روال با بقیه ی نمونه هاست. این فرایند را می تواند آنقدر ادامه داد تا فصلی از قوانین که با هم تمامی نمونه های مثبت را می پوشانند بدست آید. در آخر نیز می توان قوانین را بررسی کرد تا هنگام دسته بندی نمونه های جدید اول قوانینی بررسی شوند که حداکثر دقت را دارند. یک حالت کلی از الگوریتم های ترتیبی در جدول ۱۰.۱ آورده شده است.

Sequential-Covering(Target_attribute,Attributes,Examples,Threshold)

- Learned_rules←{}
- Rule←Learn-One-Rule(Target_attribute,Attributes,Examples)
- تا زمانی که شرط Performance(Rule,Examples)>Threshold است حلقه ی زیر را ادامه بده
  - Learned_rules←Learned_rules+Rule
  - Example←Examples-{نمونه هایی که به درست توسط Rule دسته بندی می شوند}
  - Rule←Learn-one-rule(Target_attribute,Attributes,Examples)
- مرتب شده ی Learn_rules بر اساس Performance بر Examples ← Learned_rules Learn_rules را برگردان

جدول ۱۰.۱ الگوریتم ترتیبی برای یادگرفتن دسته قوانین فصلی.
Learn-one-rule قانونی را بر می گرداند که حداقل تعدادی از نمونه های Examples را بپوشاند. Performance زیر روالی است که کیفیت قانون را بررسی می کند. این الگوریتم تا زمانی که Performance کمتر از مقدار Threshold است به یادگیری قوانین ادامه خواهد داد.

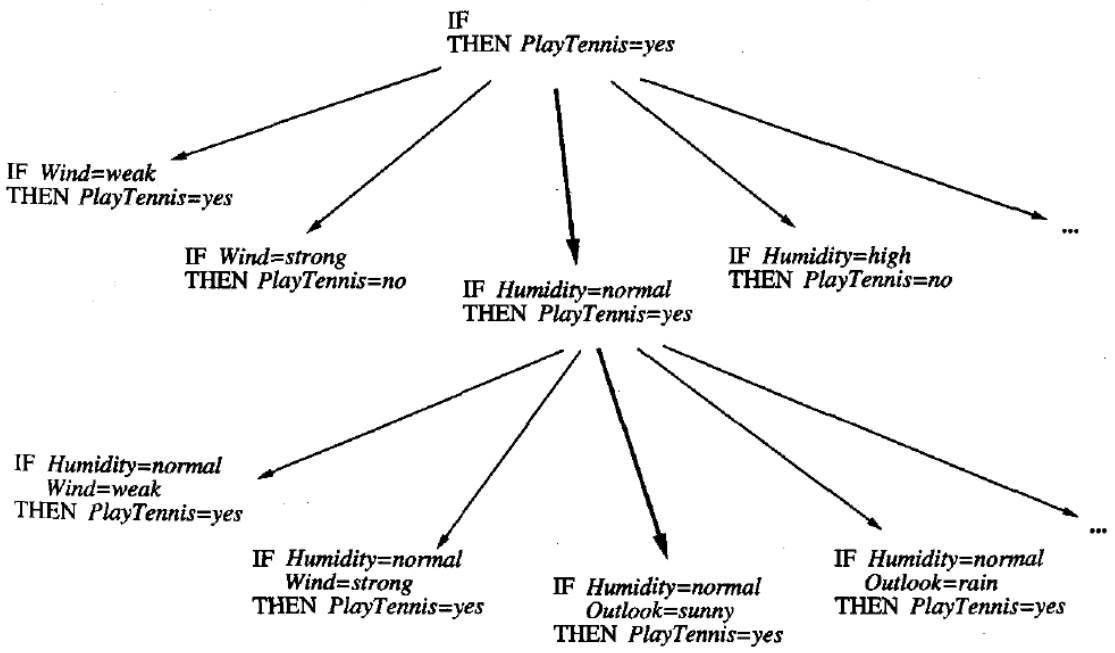
این الگوریتم ترتیبی یکی از متداول ترین روش های یادگیری دسته قوانین فصلی است. این روش مسئله ی یادگیری دسته قوانین فصلی را به سری ای از مسائل ساده تر تبدیل می کند، در هر یک از این مسائل ساده تر یک قانون عطفی یاد گرفته می شود. چون این روش، جستجویی حریصانه انجام می دهد، در یادگیری قوانین بدون مراجعه به مراحل قبلی تظمینی نیست که کوچکترین دسته قوانین یا بهترین دسته قوانینی که نمونه های آموزشی را می پوشاند را پیدا کنیم.

Learn-one-rule را چگونه باید طراحی کنیم تا نیازمان در الگوریتم ترتیبی برطرف کند. الگوریتمی می خواهیم که قانونی را ایجاد کند که دقت بالایی داشته باشد، اما لازم نیست تمام نمونه های مثبت را بپوشاند. در این بخش الگوریتم های متفاوتی را به همراه تفاوت های مهم شان بررسی خواهیم کرد. در این بخش فقط به روش های گزاره ای می پردازیم و تعمیم این روش ها به horn clause های درجه اول را به بخش بعدی موکول می کنیم.

۱۰.۲.۱ جستجوی ستونی کلی به جزئی

یکی از راههای موثر تعریف Learn-one-rule سازمان دهی فضای فرضیه ها در همان شکل کلی الگوریتم ID3 است، با این تفاوت که در اینجا ما امید وار کننده ترین شاخه ی درخت را در هر مرحله بررسی خواهیم کرد. همانطور که در درخت جستجوی شکل ۱۰.۱ نیز آمده است، جستجو قانونی شروع می شود که کلی ترین شروط را دارد (شرطی ندارد و تمامی نمونه های را می پوشاند)، سپس به صورت حریصانه با اضافه کردن ویژگی هایی که کارایی قانون را بر روی نمونه های آموزشی بیشترین افزایش می دهند ادامه پیدا خواهد کرد. این فرایند بعد از اضافه

شدن ویژگی دوم نیز به همین صورت پیش می رود، و این فرایند به همین ترتیب ادامه پیدا می کند. مشابه ID3 این فرایند نیز حریصانه با افزایش شروط ویژگی ها فرضیه را گسترش می دهد تا کارایی آن به حد آستانه ی قابل قبولی برسد. بر خلاف ID3 این تعریف Learn-one-rule فقط یک زیر شاخه در هر مرحله ی جستجو اضافه می شود، فقط یک شرط که جفتی از ویژگی و مقدار اش است، اما در ID3 یک دسته زیر شاخه اضافه می شد که تمامی مقادیر ویژگی را مورد بررسی قرار می داد.



شکل ۱۰.۱ جستجو در شروط در فرایند Learn-one-rule که از ترتیب کلی به جزئی استفاده می کند.
در هر مرحله تمامی شروطی که اضافه شدنشان ممکن است بررسی می شود. حکم قانون طوری انتخاب می شود تا نمونه هایی که در شروط قانون صدق می کنند را راضی کند. این شکل جستجوی ستونی با عمق ۱ را نشان می دهد.

جستجوی کلی به جزئی ای که در بالا آورده شد جستجویی حریصانه و با عمق یک و بدون نگاه به مراحل قبلی[1] بود. درست مثل تمامی جستجو های حریصانه، خطر بهینه سازی جزئی در هر مرحله وجود دارد. برای کم کردن این خطر، می توانیم الگوریتم را طوری تغییر دهیم تا جستجویش ستونی[2] شود؛ جستجویی که در آن به جای انتخاب بهترین گزینه لیستی k گزینه ی بهتر نگه داشته می شوند. در هر مرحله از پیشروی جستجو، خاص سازی برای تمامی این k گزینه ی مطرح ساخته و قوانین حاصل در مراحل بعدی نیز با اضافه کردن k گزینه ی بهتر خاص تر می شوند. جستجوی ستونی لیستی از امیدوار کننده ترین جایگزین های فرضیه ی فعلی (بهترین فرضیه) نگه داری می کند و میزان موفقیت آنها در هر مرحله از جستجو در نظر گرفته می شود. این الگوریتم جستجوی کلی به جزئی ستونی در برنامه ی CN2 که توسط (Clark and Niblett 1989) توصیف شد به کار رفته است. این الگوریتم به طور کامل در جدول ۱۰.۲ آورده شده است.

Learn-one-rule(Target_attribute,Attributes,Examples,k)

این زیر روال قانونی که تعدادی از نمونه های Examples را پوشش می دهد خروجی می دهد. در این جستجو زیر روال از جستجوی ستونی

---

[1] backtrack
[2] Beam search

حریصانه برای پیدا کردن بهترین قانون کمک می گیرد، این جستجو توسط معیار Performance کنترل می شود.

- Best_hypothesis را با مقدار اولیه ی Ø مقدار دهی اولیه کن.
- Candidate_hypothesis را با مقدار اولیه ی {Best_hypothesis} مقدار دهی اولیه کن.
- تا زمانی که مجموعه ی Candidate_hypotheses تهی نیست حلقه ی زیر را ادامه بده

۱. Candidate_hypotheses خاص تری بعدی را ایجاد کن

- مجموعه ی تمامی قیود به فرم (a=v) ← All_constrains
- که در آن a عضوی از مجموعه ی Attributes است و v نیز مقدار ممکنی از a است که در مجموعه ی Examples فعلی ظاهر شده
  - برای تمامی h های Candidate_hypotheses و برای تمامی c های All_constrains
    - با اضافه کردن c خاص سازی از h ایجاد کن
- هر فرضیه ی تکراری، ناسازگار یا کاملا اختصاصی را از New_candidate_hypotheses حذف کن.

۲. Best_ hypothesis را تغییر بده

- برای تمامی h های New_candidate_hypotheses حلقه ی زیر را اجرا کن
  - اگر (Performance(h,Examples,Target_attribute)> Performance(Best_hypothesis,Examples,Target_attribute))
    آنگاه Best_hypothesis←h

۳. Candidate_ hypotheses را تغییر بده

- K بهترین عضو مجموعه ی New_candidate_hypotheses را بر اساس معیار Performance
  Candidate_ hypotheses←
- قانونی به فرم زیر را خروجی بده

"اگر Best_hypothesis آنگاه prediction"

که در این رابطه prediction پر تکرار ترین مقدار Target_attribute در میان Examples هایی است که در Best_hypothesis صدق می کنند.

Performance(h,Examples,Target_attribute)

- زیر مجموعه ای از Examples که با h سازگار است ← h_Examples
- مقدار (h_Examples)Entropy– را برگردان، آنتروپی بر اساس مجموعه ی Target_attribute محاسبه می گردد.

جدول ۱۰.۲ یکی از تعریف های Learn-one-rule جستجویی کلی به جزئی و ستونی است.
شرط فرضیه ی فعلی با متغیر Candidate_hypotheses مشخص می شود. این الگوریتم مشابه الگوریتمی است که در برنامه ی CN2 به کار رفته است (Clark and Niblett 1989).

بیایید نکاتی که در مورد الگوریتم Learn-one-rule در جدول ۱۰.۲ آورده شد را با دقت بیشتری بررسی کنیم. ابتدا اینکه توجه داشته باشید که هر فرضیه که در حلقه ی اصلی در نظر گرفته می شود عطفی از شروط ویژگی مقدار است. هر یک از این فرضیه های عطفی نظیر یک

دسته شروط ممکن برای یادگیری اند که با آنتروپی نمونه هایی که می پوشاند اندازه گیری می شود. جستجو مرحله به مرحله فرضیه های خاص تر را در نظر می گیرد تا به کلی ترین فرضیه برسد که تمامی ویژگی های ممکن را داشته باشد. قانون خروجی الگوریتم قانونی است که در این جستجو بیشترین Performance را داراست، میزان خاصی قانون در این انتخاب هیچ تاثیری ندارد. حکم قانون خروجی در مرحله ی انتهایی الگوریتم مشخص می شود، بعد از این که شروط توسط متغیر Best_hypothesis مشخص شد. حکم قانون خروجی متداول ترین مقدار ویژگی هدف در نمونه های تحت پوشش آن تعیین می شود. نکته ی آخر این که توجه داشته باشید با اینکه جستجوی ستونی خطر قانون های موضعی بهینه را کمتر می کند اما هنوز خطر از بین نخواهد رفت. با این وجود، حتی زمانی که قوانین موضعی بهینه اند، الگوریتم ترتیبی می تواند دسته قوانینی را یاد بگیرد که با هم نمونه های آموزشی را بپوشانند، زیرا که الگوریتم متناوبا از زیرروال Learn-one-rule استفاده می کند.

۱۰.۲.۲ نسخه ها

الگوریتم ترتیبی، با استفاده از الگوریتم Learn-one-rule دسته قوانینی را یاد می گیرد که نمونه های آموزشی را پوشش دهند. نسخه های متفاوتی از این روش مورد بحث و بررسی قرار گرفته است. برای مثال، در بعضی شرایط ممکن است لازم باشد طوری برنامه ریزی شود که فقط قوانینی که نمونه های مثبت را می پوشاند یاد بگیریم و نمونه هایی که در هیچ یک از قوانین صدق نمی کند را به طور پیشفرض منفی دسته بندی کنیم. برای مثال، برای یادگیری تابع هدفی مثل "زنان بارداری که دوقلو باردارند" یکی از چنین شرایط است. در این حالت، نسبت نمونه های مثبت به کل جمعیت کم است، بنابراین قوانین خلاصه تر و مفهوم تر خواهند بود اگر فقط قوانین برای دسته بندی نمونه های مثبت در نظر گرفته شود و بقیه ی نمونه ها به طور پیشفرض منفی دسته بندی شوند. این روش مشابه negative-as-failure در زبان Prolog است که در آن اگر نتوان ثابت کرد که یک نمونه مثبت است به طور پیشفرض منفی فرض می شود. برای یادگیری چنین قوانینی که فقط یک مقدار ویژگی هدف را تعیین می کنند، الگوریتم Learn-one-rule را می توان طوری تغییر داد که پارامتری اضافی داشته باشد که مقدار پیشبینی مقدار هدف مورد توجه را مشخص کند. جستجوی ستونی کلی به جزئی به همان شکل قبلی دست نخورده باقی می ماند و فقط تعریف زیر روال Performance که فرضیه ها را ارزیابی می کرد تغییر می کند. توجه داشته باشید که تعریف Performance به عنوان آنتروپی در این وضعیت دیگر مناسب نیست، زیرا که این تعریف به قانون هایی که تعداد زیادی نمونه ی منفی را می پوشانند نیز مثل قوانینی که تعداد زیادی نمونه ی مثبت را می پوشانند مقدار زیادی نسبت می دهد. در چنین شرایطی استفاده از نسبت نمونه های مثبت پوشانده شده به کل نمونه های پوشانده شده ی فرضیه معیاری مناسب تر خواهد بود.

تغییرات دیگری نیز ممکن است، این نسخه های تغییریافته در خانواده ای از الگوریتم هایی با نام AQ، (Michalski , 1969, Michalski et al. 1986) قرار می گیرند، این خانواده از الگوریتم ها قبل از الگوریتم CN2 بوجود آمده و تمامی نتایج بالا بر اساس این خانواده از الگوریتم ها بدست آمده است. الگوریتم های AQ نیز فصلی از قوانین عطفی را یاد می گیرند، با این وجود، AQ و الگوریتم بیان شده در بسیاری از جهات متفاوت اند. ابتدا اینکه الگوریتم پوششی[1] ی AQ با الگوریتم ترتیبی فرق دارد، زیرا که صراحتا به دنبال قوانینی می گردد که مقدار هدف خاصی را پوشش می دهند، و برای هر مقدار هدف یک دسته قوانین فصلی را یاد می گیرد. دوم اینکه الگوریتم های AQ برای یادگیری تک قوانین از روشی متفاوت با Learn-one-rule استفاده می کنند. مشابه قبل این زیر روال نیز از جستجوی ستونی کلی به جزئی استفاده می کند با این تفاوت که برای تمرکز این جستجو از یک نمونه ی مثبت استفاده می شود. در کل، این زیر روال فقط ویژگی هایی را در نظر می گیرد که با نمونه ی مثبت سازگارند و با این فرض برای تعمیم از جستجوی کلی به جزئی استفاده می کند. در هر مرحله الگوریتم

[1] covering

قانونی سازگار با یکی از نمونه های مثبت غیر پوشش داده شده یاد می گیرد تا مشابه قبل جستجو را برای پیدا کردن فصلی از قوانین عطفی ادامه دهد.

۱۰.۳ یادگیری دسته قوانین: خلاصه

الگورتیم ترتیبی بالا و الگوریتم یادگیری درختی فصل ۳ متد های متفاوت ممکنی برای یادگیری دسته قوانین هستند. این قسمت جنبه های مختلف فضای طراحی این گونه الگوریتم ها (الگوریتم های یادگیری دسته قوانین) را بررسی خواهد کرد.

ابتدا اینکه الگوریتم های ترتیبی در هر مرحله فقط یک قانون را یاد می گیرند و نمونه هایی را که توسط آن قانون پوشانده می شوند حذف کرده و همین فرایند را با بقیه ی نمونه ها ادامه می دهند. در مقابل، الگوریتم های یادگیری درختی مثل ID3 کل دسته قوانین را با هم و در یک جستجو برای درختی قابل قبول یاد می گیرند. به همین دلیل، الگوریتم هایی چون ID3 را الگوریتم های پوشش همزمان[1] می نامند، در مقابل الگوریتم های ترتیبی مثل CN2. کدام یک از این نوع الگوریتم ها ارجهیت دارد؟ تفاوت کلیدی در قدم های ابتدایی جستجوی آنهاست. در هر مرحله ی ID3 بین تمامی ویژگی های ممکن، ویژگی ها با استفاده از تقسیم بندی ای که انجام می دهند انتخاب می شوند. در مقابل CN2 در بین جفت ویژگی مقدار ها، ویژگی ها را با استفاده از زیر مجموعه ای از داده ها که می پوشانند انتخاب می کند. یکی از راههای درک این تفاوت توجه به تعداد انتخاب های خاصی است که این دو الگوریتم انجام می دهند تا دسته قوانین مساوی ای را یادبگیرند. در یادگیری n قانون k ویژگی تست ممکن برای شروط دارد، الگوریتم های ترتیبی k.n مرحله جستجو انجام می دهند و در هر مرحله نیز انتخابی مستقل برای معلوم کردن شرط قانون انجام می دهند. در مقابل، الگوریتم های پوشش همزمان تعداد بسیار کمتری انتخاب مستقل انجام می دهند زیرا که هر انتخاب ویژگی در گره ای از درخت متناسب با معلوم کردن آن ویژگی برای تعداد زیادی از قوانین است. به عبارت دیگر، اگر گره ای از درخت ویژگی ای که m مقدار ممکن دارد را بررسی کند، انتخاب این ویژگی برای آن گره ی درخت هم ارز انتخاب این ویژگی برای m قانون نظیر آن است (تمرین ۱۰.۱). بنابراین، الگوریتم های ترتیبی مثل CN2 نسبت به الگوریتم های پوشش همزمان مثل ID3 تعداد بیشتری انتخاب مستقل انجام می دهند. با این وجود، این سوال همچنان بدون جواب باقی است: کدام روش ارجهیت دارد؟ اگر داده های موجود به اندازه ی کافی زیاد باشد تا بتوان تعداد زیاد انتخاب های مستقل الگوریتم ترتیبی را برطرف کند الگوریتم ترتیبی بهتر است، در مقابل اگر داده های موجود کم باشد بهتر است که انتخاب ویژگی ها بین شروط مشترک باشد و الگوریتم های پوشش همزمان مفید تر خواهد بود. جنبه ی با اهمیت دیگر، این است که، آیا اینکه قوانین ویژگیهای مشابهی را تست کنند برای ما مطلوب است؟ در الگوریتم های پوشش همزمان مثل یادگیری درختی چنین کاری انجام می شود. اما در الگوریتم های ترتیبی نیازی به چنین کاری نیست.

جنبه دوم تفاوت این دو نوع الگوریتم نحوه ی کنترل جستجویشان در Learn-one-rule است. در الگوریتمی که در بالا توضیح دادیم، این جستجو از فرضیه های کلی تر به جزئی تر انجام می شود. در دیگر الگوریتم های مطرح شده (مثل Find-S که در فصل ۲ بود) این جستجو از جزئی به کلی انجام می شود. یکی از مزیت های جستجوی کلی به جزئی در این است که کلی ترین فرضیه منحصر به فرد است، اما جزئی ترین فرضیه ها در اکثر فضا های فرضیه ای منحصر به فرد نیستند (به تعداد نمونه ها خاصترین فرضیه وجود دارد). با وجود تعداد زیادی خاص ترین فرضیه معلوم نیست که جستجو را باید از کدام فرضیه شروع کرد. یکی از برنامه هایی که از جستجوی جزئی به کلی استفاده می کند، Golem

[1] simultaneous covering

(Muggleton and Feng 1990) است که این مشکل را با انتخاب چندین نمونه به طور تصادفی و شروع با آنها حل می کند. سپس بهترین فرضیه در میان این فرضیه های تصادفی انتخاب می شود.

جنبه سوم تفاوت در این است که Learn-one-rule جستجویی آزمون وخطایی[1] در فرضیه های با قائده دارد یا به عبارت دیگر کنترل نمونه ای[2] است و تک نمونه های آموزشی تعمیم آنرا کنترل می کنند. الگوریتم های جستجوی متداول کنترل نمونه ای شامل Find-S، Candidate-Elimination (در فصل ۲)، AQ، و الگوریتم Cigol که پیشتر در همین فصل بررسی شد هستند. در هر یک از این الگوریتم ها تولید یا تغییر فرضیه با بررسی یک تک نمونه ی آموزشی انجام می شود، و انتظار می رود که فرضیه ی حاصل کارایی بهتری برای آن تک نمونه داشته باشد. این نوع الگوریتم ها با الگوریتم Learn-one-rule آزمون و خطایی که در جدول ۱۰.۲ آمد متفاوت اند، در این الگوریتم فرضیه های موفق فقط بر اساس زبان[3] بیان فرضیه ها بوجود می آیند و فقط زمانی به داده های آموزشی رجوع می شود که می خواهیم با استفاده از کارایی بر روی کل نمونه های آموزشی، بین این فرضیه های ممکن فرضیه ای را انتخاب کنیم. یکی از مزیت های مهم روش آزمون و خطا این است که هر انتخاب در جستجو بر اساس کارایی است که بر پایه ی تعداد زیادی نمونه تعریف می شود، بنابراین اثر داده های خطا دار مینیمم می شود، اما در مقابل در الگوریتم های کنترل نمونه ای که فرضیه ها بر اساس تک نمونه های آموزشی تغییر می کنند خطر اشتباه بر اساس یک نمونه ی خطای آموزشی بسیار زیاد است و الگوریتم در مقابل داده های خطا دار کاملا آسیب پذیر است.

تفاوت چهارم دو روش این است که آیا و چگونه فرضیه ها هرس می شوند؟ همانطور که می دانید در یادگیری درختی، احتمال این وجود دارد که دسته قوانینی پیدا کنیم که با نمونه های آموزشی خیلی خوب سازگار باشد اما روی کل نمونه ها ضعیف عمل کند. یکی از روشهای بر طرف کردن این مشکل هرس هر قانون بعد از یادگیری از نمونه های آموزشی است. در کل، شروط قوانینی که حذف شان باعث افزایش کارایی قوانین بر روی یک دسته ی هرس، که مجزا از داده های آموزشی است، می شود باید هرس شوند. بحث کامل روی این موضوع را در قسمت ۳.۷.۱.۲ انجام دادیم.

جنبه آخر تفاوت بین این دو روش، نحوه ی تعریف روال Performance که برای کنترل جستجو در Learn-one-rule استفاده می شود است. توابع ارزیابی مختلفی را می توان مورد استفاده قرار داد. چندین نمونه از متداول ترین توابع ارزیابی در زیر آورده شده اند:

- تکرار نسبی[4]. اگر n تعداد نمونه هایی باشد که با قانون تطابق دارند و n_c تعداد نمونه هایی باشد که قانون درست دسته بندی می کند، ارزیابی تکرار نسبی قانون کسر زیر خواهد بود:

$$\frac{n_c}{n}$$

  تکرار نسبی در ارزیابی قوانین در AQ مورد استفاده قرار گرفته است.

- تخمین m دقت[5]. این تخمین دقت به سمت دقتی پیشفرض برای قوانین بایاس شده است. از این روش زمانی که تعداد داده ها کم است و دقت قوانین باید بر اساس مجموعه ی نمونه های کوچکی تخمین زده شود استفاده می شود. اگر n و n_c به ترتیب تعداد

[1] generate then test
[2] example driven
[3] syntax
[4] relative frequency
[5] m-estimate of accuracy

نمونه های سازگار و درست دسته بندی شده بر اساس قانون مورد نظر باشند و p نیز احتمال اولیه ی نمونه ی تصادفی باشد و اگر m یک وزن باشد، خواهیم داشت که دقت تخمین m، میانگین وزن دار تکرار نسبی و احتمال اولیه خواهد بود.

$$\frac{n_c + mp}{n + m}$$

توجه دارید که اگر m صفر باشد این کسر همان تکرار نسبی خواهد بود. با افزایش مقدار m تعداد نمونه هایی که برای تغییر احتمال اولیه ی p لازم است افزایش خواهد یافت. معیار m میانگین توسط (Cestnik and Bratko 1991) پیشنهاد شد و در نسخه های مختلف CN2 نیز به کار می رود. همچنین این تخمین در دسته بندی کننده ساده ی بیز به کار می رود.

- آنتروپی[1]. این معیار در تعریف Performance در الگوریتم جدول ۱۰.۲ استفاده شده است. اگر S مجموعه ای از نمونه ها باشد که با شروط قانون تطابق دارد، آنتروپی یکدستی تابع هدف را در این مجموعه از نمونه ها اندازه گیری می کند. ما از منفی آنتروپی استفاده می کنیم تا قوانین بهتر امتیاز بیشتری داشته باشند.

$$-Entropy(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

در این رابطه c تعداد مقادیر ممکن تابع هدف است و p_i نیز نسبتی از S است که ویژگی هدف i امین مقدارش را می گیرد. ترکیبی از آنتروپی و یک تست آماری در الگوریتم CN2 مورد استفاده قرار گرفته است (Clark and Niblett 1989). همچنین آنتروپی هسته ی تابع بهره ی اطلاعات در بسیاری از الگوریتم های یادگیری درختی است.

۱۰.٤ یادگیری قوانین درجه اول

در قسمت قبل، الگوریتم هایی که دسته قوانین گزاره ای را یاد می گرفتند (دسته قوانینی که هیچ متغیری نداشتند) را مورد بحث قرار دادیم. در این بخش، به یادگیری قوانینی که متغیر دارند خواهیم پرداخت و یادگیری horn clause های درجه اول را در حالت کلی بررسی می کنیم. یکی از انگیزه های توجه به چنین قوانینی قدرت بیان آنها نسبت به قوانین گزاره ای است. یادگیری استقرایی قوانین یا تئوری های درجه اول گاهی برنامه نویسی استقرایی منطقی[2] (ILP) نامیده می شود زیرا که می توان به این فرایند به دید روش ایجاد خودکار برنامه های Prolog از نمونه ها نگاه کرد. Prolog هدفی کلی است، زبان برنامه نویسی معادل تورینگ که در آن برنامه ها با دسته horn clause ها مشخص می شوند.

۱۰.٤.۱ horn clause های درجه اول

برای مشاهده ی مزیت های استفاده از نمایش درجه اول به جای نمایش گزاره ای (بدون متغیر)، کار یادگیری مفهوم ساده ی Daughter(x,y) را که بر روی زوج افراد x و y تعریف می شود را در نظر بگیرید. زمانی که x دختر y است مقدار Daughter(x,y) درست است و در غیر این صورت مقدار آن غلت است. فرض کنید که برای هر فرد در داده های موجود با ویژگی های Name، Father،

[1] entropy
[2] inductive logic programming

Mother، Male و Female توصیف می شود. بنابراین، هر نمونه ی آموزشی توصیفی از دو فرد با ویژگی هایشان و مقدار ویژگی هدف Daughter خواهد بود. برای مثال، در زیر یک نمونه ی مثبت آورده شده است:

$$< Name_1 = Sharon, Mother_1 = Louise, Father_1 = Bob,$$

$$Male_1 = False, Female_1 = True, Name_2 = Bob,$$

$$Mother_2 = Nora, Father_2 = Victor, Male_2 = True,$$

$$Female_2 = Fales, Daughter_{1,2} = True >$$

در این نمونه زیر نویس هر ویژگی مشخص می کند که ویژگی کدام یک از دو فرد توصیف می شود. حال اگر تعدادی نمونه ی آموزشی برای مفهوم هدف $Daughter_{1,2}$ جمع کنیم و آنها را به یک یادگیر گزاره ای مثل CN2 یا C4.5 بدهیم قانون های خروجی مثل قانون زیر خواهند بود:

$$IF \quad (Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = True)$$

$$THEN \quad Daughter_{1,2} = True$$

با وجود اینکه این قانون درست است اما بسیار خاص است، حتی اگر در دسته بندی نمونه های جدید به کار رود کاربردش بسیار کم خواهد بود. مشکل اینجاست که نمایش گزاره ای هیچ راه کلی ای برای توصیف روابط بین مقادیر ویژگی ها ندارد. در مقابل، برنامه ای که از قوانین درجه اول استفاده می کند می تواند قانون کلی زیر را یاد بگیرد:

$$IF \quad Father(y,x) \wedge Female(y), \quad THEN \quad Daughter(x,y)$$

در این رابطه x و y متغیر هایی هستند که هر فردی می توانند باشند.

Horn clause های درجه اول همچنین می توانند متغیر هایی در شروط داشته باشند که در نمایش گزاره ای ممکن نیست. برای مثال، یک قانون برای GrandDaughter می تواند قانون زیر باشد:

$$IF \quad Father(y,z) \wedge Mother(z,x), Female(y)$$

$$THEN \quad GrandDaughter(x,y)$$

توجه داشته باشید که متغیر z در این رابطه پدر y است که در حکم قانون نیآمده است. زمانی که یک متغیر فقط در شرط یک قانون ظاهر می شود بدین معناست که قانون به وجود آن وابسته است؛ به عبارت دیگر، حکم قانون تا زمانی درست است که حداقل یک نمونه وجود داشته باشد که در شرط قانون درمکان z صدق کند.

همچنین در این نوع نمایش استفاده از خود حکم و ایجاد قوانین بازگشتی ممکن می باشد. برای مثال، دو قانونی که در ابتدای همین فصل آورده شده اند با هم ویژگی Ancestor(x,y) را بیان می کردند. متد های یادگیری ILP مثل متد هایی که در زیر توضیح خواهیم داد اثبات شده که می توانند پهنای گسترده ای از توابع بازگشتی ساده (مثل تابع Ancestor و توابعی که برای ترتیب کردن عناصر یک لیست ، پاک کردن یک عضو خاص یا ترکیب کردن دو لیست به کار می روند) را یاد بگیرند.

۱۰.٤.۲ واژگان

قبل از شروع بحث در مورد الگوریتم های یادگیری دسته قوانین horn clause، بیایید ابتدا بعضی واژگان اساسی منطق را معرفی کنیم. تمامی اصطلاحات[۱]، ترکیبی از ثابت ها[۲] (مثلا Bob و Louise)، متغیر ها[۳] (مثل x و y) ، نماد های گزاره ای[۴] (مثل Married و Greater_Than) و نماد های توابع[۵] (مثل age) هستند. تفاوت گزاره ها و توابع این است که گزاره ها یکی از دو مقدار True و False را می گیرند اما توابع می توانند هر ثابتی را به عنوان مقدار داشته باشند. ثابت ها را با حروف بزرگ و متغیر ها را با حروف کوچک نشان خواهیم داد. همچنین توابع را با حروف کوچک و گزاره ها را با حروف بزرگ نشان خواهیم داد.

با این نشانه گذاری می توان اصطلاح به فرم ذیل ساخت: یک جمله[۶] مساوی یک ثابت، یا متغیر یا هر تابعی از یک جمله باشد (مثل Bob، x، age(Bob)). یک عبارت[۷] یک گزاره یا نقیض آن است (مثل Married(Bob,Louise) یا ¬Greater_Than(age(Sue),20)). اگر یک عبارت علامت نقیض (¬) داشته باشد به آن عبارت منفی می گوییم، در غیر این صورت به آن عبارت مثبت می گوییم.

یک بند[۸] فصلی از عبارات است که در آن فرض می شود تمامی متغیر ها معلوم فرض می شوند. یک horn clause شامل حداکثر یک عبارت مثبت است،

$$H \vee \neg L_1 \vee \ldots \vee \neg L_n$$

در این رابطه H عبارت مثبت و $\neg L_1 \ldots \neg L_n$ عباراتی منفی هستند. چون داریم $(B \leftarrow A) = (B \vee \neg A)$ و $\neg(B \wedge A) = (\neg B \vee \neg A)$، پس horn clause بالا را می توان به صورت مشابه به شکل زیر نشان داد.

$$H \leftarrow (L_1 \wedge \ldots \wedge L_n)$$

که در نمادگذاری قبلی معادل با عبارت زیر است.

$$IF\, L_1 \wedge \ldots \wedge L_n\ THEN\ H$$

بدون توجه به نحوه ی نمایش horn clause شروط قانون $L_1 \wedge \ldots \wedge L_n$ بدنه ی قانون[۹] یا مقدم[۱۰] نامیده می شوند. عبارت H حکم را تشکیل می دهد که سر قانون[۱۱] یا نتیجه[۱] نامیده می شود. برای راحتی کار، این تعاریف در جدول ۱۰.۳ گرد آوری شده اما باز در موقع مواجهه با با تعاریف دیگر آنها را بیان می کنیم.

---

[1] Expression
[2] constant
[3] variable
[4] predicate symbol
[5] Function symbols
[6] term
[7] literal
[8] clause
[9] clause body
[10] antecedents
[11] clause head

هر عبارت خوش فرم[2] از ثابتها (مثل Joe, 23 ,Mary)، متغیرها (مثل x)، گزاره ها (مثل، Female در (Female(Mary) و توابع (مثل age در (age(Mary) تشکیل شده است. (گزاره=predicate)

جمله[3] هر ثابت، متغیر یا تابعی است که بر روی جمله ای دیگر اعمال شده است. (مثل Mary, age(Mary), x, age(x)).

عبارت[4] هر گزاره یا عکس گزاره ای که به مجموعه ای از جمله ها اعمال می شود است. (مثل، Female(Mary), ¬Female(x), Greater_than(age(Mary,20))

عبارت ؟ (ground literal) ---

عبارت منفی (negative literal) عبارتی است که پیشوند منفی دارد (مثل، ¬Female(Joe)).

عبارت مثبت (positive literal) عبارتی است که علامت منفی نداشته باشد (مثل، Female(Mary)).

--- (clause) فصلی از عبارات به فرم $M_1 \vee \dots M_n$ است که متغیر های آن در کل گزاره هایی هستند.

(horn clause) ها به فرم زیر بیان می شوند

$$H \leftarrow (L_1 \wedge \dots \wedge L_n)$$

در این رابطه $H, L_1, \dots, L_n$ همگی عبارات مثبتی هستند. H سر horn clause یا حکم (consequent) نامیده می شود. عطف عبارات $L_1 \wedge \dots \wedge L_n$ نیز بدنه یا شرط horn clause نامیده می شود.

برای تمامی عبارات A و B عبارت (A←B) معادل (A∧¬B) است و ¬(A∧B) نیز معادل (¬A∨¬B). بنابراین، horn clause ها را می توان به فرم معادل فصلی نوشت

$$H \vee \neg L_1 \vee \dots \vee \neg L_n$$

یک جانشینی (substitution) تابعی است که متغیر ها را با جملاتی جایگزین می کند. برای مثال، جانشینی {x/3,y/z} متغیر x را با جمله ی 3 و متغیر y را با جمله ی z جایگزین می کند.

برای عبارت L و جایگزینی θ از Lθ برای حاصل اعمال θ به L استفاده می کنیم.

جانشینی یکتا کننده[5] ی دو عبارت L_1 و L_2 جانشینی θ است که داشته باشیم $L_1\theta = L_2\theta$.

جدول ۱۰.۳ تعاریف اولیه ی منطق درجه اول (first order logic).

---

[1] consequent
[2] well-formed
[3] term
[4] literal
[5] unifying substitution

۱۰.۵ یادگیری دسته قوانین درجه اول: FOIL

الگوریتم های مختلفی برای یادگیری دسته قوانین درجه اول یا همان horn clause ها ارائه شده است. در این بخش برنامه ای به نام FOIL (Quinlar 1990) که روشی بسیار مشابه الگوریتم های ترتیبی و الگوریتم های Learn-one-rule قسمت قبل است را توضیح می دهیم. در واقع برنامه ی FOIL تعمیم طبیعی الگوریتم های قبلی به نمایش قوانین درجه اول است. رسما، فرضیه های یادگرفته شده ی FOIL دسته قوانین درجه اولی هستند که هر قانون یک horn clause است. فقط دو تفاوت وجود دارد. اول اینکه قوانین یادگرفته شده ی FOIL محدود کننده تر از قوانین کلی horn clause هستند، زیرا که عبارات آنها نمی توانند شامل توابع نمادی باشند (این باعث می شود تا پیچیدگی جستجو فضای فرضیه ای کمتر شود). دوم اینکه قوانین FOIL از قوانین horn clause شامل ترند[1] زیرا که قوانین FOIL می توانند در بدنه قانون عبارت منفی نیز داشته باشند. از FOIL برای مسائل زمینه های مختلف استفاده شده است. برای مثال، از آن برای یادگیری حالت بازگشتی الگوریتم Quicksort و یادگیری تمییز دادن چینش های قانونی و غیر قانونی صفحه ی شطرنج استفاده شده است.

الگوریتم FOIL در جدول ۱۰.۴ به طور خلاصه بیان شده است. توجه دارید که حلقه ی خارجی مشابه نسخه ای از الگوریتم های ترتیبی ای که قبلا درباره ی آن بحث کردیم است؛ این حلقه در هر اجرا یک قانون جدید یاد می گیرد و نمونه های مثبتی را که توسط قانون پوشانده می شوند حذف خواهد کرد. حلقه ی داخلی متناسب با نسخه ای از الگوریتم Learn-one-rule است که برای نمایش قوانین درجه اول تعمیم داده شده است. همچنین توجه دارید که تفاوت های بسیار کم و کوچکی میان FOIL و الگوریتم های قبلی وجود دارد. در کل، FOIL بر خلاف الگوریتم قبلی که دنبال قوانینی که مقدار تابع هدف را درست یا غلت دسته بندی می کنند تنها بدنبال قوانینی می گردد که مقدار تابع هدف را درست (True) پیشبینی می کنند. همچنین جستجوی FOIL بیشتر hillclimbing است تا جستجوی ستونی (به طور معادل، از ستونی با پهنای یک برای جستجو استفاده می کند).

FOIL(Target_predicate,Predicates,Examples)

Pos ← تمامی نمونه هایی که برایشان Target_predicate درست است.
Neg ← تمامی نمونه هایی که برایشان Target_predicate غلت است.
Learned_rules ← {}
تا زمانی که Pos تهی نیست حلقه ی زیر را ادامه بده
قانون جدید NewRule را یاد بگیر

NewRule ← قانونی که Target_predicate را بدون شرط پیشبینی می کند
Neg ← NewRuleNeg
تا زمانی که NewRuleNeg تهی نیست حلقه ی زیر را ارامه بده
عبارتی جدید برای خاص سازی به NewRule اضافه کن

Candidate_literals ← عبارات کاندید برای NewRule را با Predicates ایجاد کن.

---

[1] expressive

$$\underset{L \in Candidate_literals}{\operatorname{argmax}} \quad Foil_Gain(L, NewRule) \rightarrow Best_literal$$

Best_literal را به شروط NewRule اضافه کن.

NewRuleNeg ← زیرمجموعه ای از NewRuleNeg که شروط NewRule را راضی می کند.

Learned_rules+NewRule → Learned_rules

Pos ← {اعضایی از Pos که توسط NewRule پوشانده می شوند}-Pos .

Learned_rules را خروجی بده.

جدول ۱۰.۴ الگوریتم پایه ای FOIL.

متد خاصی برای ساخت Candidate_literals و تعریف FOIL_Gain به کاربرده می شود که در متن آورده شده است. الگوریتم پایه ای را می توان با کمی تغییر با داده های خطادار سازگار کرد، در متن به آن تغییرات اشاره شده است.

جستجوی فضای فرضیه ای FOIL با بررسی سلسله مراتبی[1] بهتر درک می شود. هر تکرار حلقه ی بیرونی FOIL یک قانون به مجموعه ی فصلی فرضیه Learned_rules می افزاید. اثر هر قانون جدید در کلی تر کردن فرضیه فصلی فعلی است (اندازه ی مجموعه ی نمونه هایی که مثبت دسته بندی می کند را افزایش می دهد). با این نگاه، روش جستجو کلی به جزئی در میان فضای فرضیه ای و با شروع از خاص ترین فرضیه فصلی و خروج از الگوریتم در زمانی که تمام نمونه ها را پوشاند خواهد بود. حلقه ی داخلی جستجویی بهتر[2] برای تعیین دقیق تعریف هر قانون جدید انجام می دهد. این حلقه ی داخلی فضای فرضیه ای دومی که فصل عبارت هاست را برای پیدا کردن فصلی که فرض های قانون را تشکیل دهد جستجو می کند. با این فضای فرضیه ای، این حلقه از روشی کلی به جزئی، hillclimbing و با شروع از کلی ترین فرضیه ی ممکن (بدون هیچ شرطی) و افزایش عبارات به آن برای خاص تر کردن قانون برای پرهیز از پوشاندن نمونه های منفی عمل می کند.

دو تفاوت اساسی میان FOIL و الگوریتمهای ترتیبی و Learn-one-rule که قبلا بررسی کردیم وجود دارد. این تفاوت ها که ناشی از قوانین درجه اولند به شرح زیرند:

در جستجوی کلی به جزئی برای یادگیری قوانین جدید، FOIL از مراحلی دیگر برای ایجاد خاص سازی های ممکن استفاده می کند.این تفاوت ناشی از آن است که شروط قانون می توانند متغیر نیز داشته باشند.

FOIL از معیار کارایی ای به نام Foil_Gain استفاده می کند در حالی که Learn-one-rule که در جدول ۱۰.۲ نیز آمده بود از معیار آنتروپی استفاده می کرد. این تفاوت ناشی از آن است که FOIL فقط به دنبال قوانینی می گردد که نمونه ها را مثبت دسته بندی کنند.

دو قسمت بعدی این تفاوت ها را به طور دقیقتر بررسی می کنند.

۱۰.۵.۱ ایجاد خاص سازی های ممکن FOIL

برای ایجاد خاص سازی ای از قانون فعلی، FOIL مجموعه ای از عبارات جدید را ایجاد می کند، هر یک از این عبارات ممکن است به تنهایی به شروط قانون اضافه شوند. به عبارت دقیقتر، فرض کنید که قانون فعلی به فرم زیر باشد،

[1] hierarchically
[2] finer-grained

$$P(x_1, x_2, \dots, x_k) \leftarrow L_1 \dots L_n$$

در این رابطه $L_1 \dots L_n$ عبارات تشکیل دهنده ی شرط قانون فعلی و $P(x_1, x_2, \dots, x_k)$ نیز عبارت سر قانون یا حکم[1] است. FOIL خاص سازی های ممکن این قانون را با در نظر گرفتن عبارتی جدید مثل L_{n+1} که به یکی از فرمهای زیر است:

$Q(v_1, \dots, v_r)$، که در آن Q گزاره ی دلخواهی از Predicates و v_i ها نیز متغیر های جدید یا موجود در قانون هستند. حداقل یکی از v_i ها در ایجاد عبارت باید در قانون موجود باشد.

$Equal(x_j, x_k)$، که در آن x_j و x_k متغیر های موجود در قانون هستند.

قرینه ی یکی از دو عبارت بالا.

برای تصور، یادگیری قوانین را برای عبارت هدف GrandDaughter(x,y) بر اساس گزاره های Father (predicator) و Female در نظر بگیرید. جستجوی کلی به جزئی FOIL با کلی ترین قانون ممکن شروع می شود،

$$\text{GrandDaughter}(x, y) \leftarrow$$

که بدین معناست که GrandDaughter(x,y) برای هر x و y مقدار درست دارد. برای خاص سازی این قانون اولیه، فرایند بالا عبارات زیر را به عنوان خاص تر سازی های ممکن شرط قانون در نظر می گیرد: Equal(x,y)، Female(x)، Female(y)، Father(y,x)، Father(x,y)، Father(x,z)، Father(y,z)، Father(z,x)، Father(z,y) و نقیض عبارات مذکور (مثل $\neg\text{Equal}(x, y)$). توجه دارید که z متغیری جدید است در حالی که x و y در قانون فعلی موجود اند.

حال فرض کنید که بین عبارات مذکور، FOIL حریصانه عبارت Father(y,z) که ما را به خاص ترین عبارت ممکن می برد را انتخاب کند،

$$\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z)$$

در کلی سازی عبارات ممکن برای خاص تر کردن این قانون، FOIL تمامی عبارات ذکر شده در مرحله ی قبل به اضافه ی عبارات Female(z)، Equal(z,x)، Father(z,w)، Father(w,z) و نقایضشان را در نظر خواهد گرفت. این عبارات جدید بخاطر اضافه شدن متغیر z در مرحله ی قبل به مجموعه ی عبارات ممکن اضافه می شوند. بهمین خاطر متغیر جدید w نیز در نظر گرفته خواهد شد.

اگر FOIL در این مرحله عبارت Father(z,x) و در مرحله ی بعدی Female(y) را انتخاب کند نتیجه قانون زیر خواهد بود، که فقط نمونه های مثبت را می پوشاند و متاقبا جستجو را متوقف می کند.

$$\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z) \wedge Father(z, x) \wedge Female(y)$$

در این لحظه، FOIL تمامی نمونه های مثبت پوشانده شده توسط این قانون را حذف خواهد کرد. اگر نمونه ی مثبت باقی بماند، جستجوی کلی به جزئی برای یافتن قانونی دیگر از سر گرفته خواهد شد.

[1] postcondition

۱۰.۵.۲ کنترل جستجو در FOIL

برای انتخاب بهترین عبارت میان عبارات ممکن تولیدی در هر مرحله FOIL از معیاری از کارایی قانون بر روی داده های آموزشی استفاده می کند. در این کار، تمامی ترکیب های ممکن متغیر ها ی قانون فعلی در نظر گرفته می شود. برای تصور این فرایند، دوباره مثالی یادگیری GrandDaughter(x,y) را در نظر بگیرید. فرض کنید که داده های آموزشی مجموعه ی ساده ی زیر باشد و از قرارداد[1] P(x,y) استفاده می کنیم (بخوانید "P ی x، y است").

GrandDaughter(Victor,Sharon) Fahter(Sharon,Bob) Father(Tom,Bob)

Female(Sharon) Father(Bob,Victor)

بیایید در اینجا برای سادگی کار فرض کنیم که برای تخمین متغیر GrandDaughter فقط از متغیر های GrandDaughter، Father و Female بر روی ثابتهای Victor، Sharon، Bob و Tom استفاده می کنیم، متغیر هایی که در بالا نیآمده اند غلت فرض شده اند (بدین معنا که انگار عبارات ¬GrandDaughter(Tom,Bob)، ¬GrandDaughter(Victor,Victor) و ... در بالا آمده اند.).

برای انتخاب بهترین خاص کننده ی قانون فعلی، FOIL تمامی ترکیب های ممکن ثابت های داده های آموزشی را در نظر می گیرد. برای مثال، در مرحله ی اول قانون به شکل زیر خواهد بود،

$$\text{GrandDaughter}(x, y) \leftarrow$$

در این قانون متغیرهای x و y لازم نیست هیچ شرط خاصی داشته باشند و ممکن است هر ترکیب چهار ثابت Victor، Sharon، Bob و Tom باشند. در اینجا از نمایش {x/Bob, y/Sharon} برای نمایش یک ترکیب (binding) خاص استفاده می کنیم، که به هر متغیر یک مقدار ثابت نسبت می دهد. چون ۴ ثابت وجود دارند بنابراین برای مقدار اولیه ی قانون ۱۶ حالت ترکیب ممکن است. ترکیب {x/Victor, y/Sharon} یک نمونه ی مثبت است زیرا که داده های آموزشی GrandDaughter(Victor,Shron) را شامل می شود. ۱۵ ترکیب دیگر ممکن (مثل {x/Bob, y/Tom}) مدارک منفی (negative evidence) در قانون مثال فعلی هستند زیرا که هیچ ادعایی (assertion) برای درستی آنها در داده های آموزشی موجود نیست.

در هر مرحله قانون بر اساس این مجموعه ی ترکیب های[2] مثبت و منفی ارزیابی می شود، با این فرض که قوانینی که ترکیب های مثبت بیشتر و ترکیب های منفی کمتری را بپوشانند ارجهیت بیشتری دارند. با اضافه شدن عبارات بیشتر به قانون، مجموعه ی ترکیب ها نیز تغییر خواهد کرد. توجه داشته باشید که اگر عبارتی که متغیر جدیدی را تعریف می کند به قانون اضافه شود، تعداد ترکیب های قانون در طول افزایش خواهد یافت (برای مثال، اگر Father(y,z) به قانون بالا اضافه شود، ترکیب اولیه ی {x/Victor, y/Sharon} به {x/Victor, y/Sharon,

[1] convention
[2] binding

{z/Bob} که طول بیشتری دارد تبدیل خواهد شد. همچنین توجه دارید که اگر امکان داشته باشد که متغیر جدید با ثابت های مختلفی جفت[1] شود، تعداد ترکیب های قانون جدید بیشتر از تعداد ترکیبهای قانون قبلی خواهد بود.

تابع ارزیابی FOIL برای تخمین کارایی اضافه کردن عبارت جدید بر اساس ترکیب های مثبت و منفی پوشش داده شده قبل و بعد از اضافه کردن عبارت جدید عمل می کند. به عبارت دقیقتر، قانونی مثل R و عبارت ممکن L که به بدنه ی آن اضافه می شود را در نظر بگیرید، اگر R′ قانون جدید حاصل از اضافه کردن L به R باشد، مقدار Foil_Gain(L,R) که برای اضافه کردن L به R تعریف می شود به صورت زیر تعریف می شود،

$$Foil_Gain(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right) \qquad (10.1)$$

در این رابطه p_0 و n_0 به ترتیب تعداد ترکیب های مثبت و منفی قانون R و p_1 و n_1 به ترتیب تعداد ترکیب های مثبت و منفی قانون R′ هستند. t نیز تعداد نمونه های مثبتی است که توسط قانون R پوشانده می شد و با اضافه کردن عبارت L هنوز پوشانده می شود. هنگامی که متغیری جدید با اضافه کردن L به R اضافه می شود، ترکیب های قبلی پوشانده شده، زمانی پوشانده شده در باقی می مانند که ترکیبی نظیر آنها در R′ صدق کند.

تابع Foil_Gain تفسیری مستقیم از تئوری اطلاعات است. بنابه تئوری اطلاعات، $-\log_2 \frac{p_0}{p_0+n_0}$ مینیمم تعداد بیتهای لازم برای کد سازی دسته بندی یک ترکیب دلخواه از ترکیب های پوشانده شده توسط قانون R است. به طور مشابه، $-\log_2 \frac{p_1}{p_1+n_1}$ نیز تعداد بیت های لازم برای کد سازی یک ترکیب دلخواه از ترکیب های قانون R′ است. از آنجایی که t تعداد ترکیب های پوشانده شده ی مشترک R و R′ است، Foil_Gain(L,R) را می توان به عنوان میزان کاهش تعداد بیتهای لازم برای کدسازی تمامی دسته بندی های ممکن ترکیب های R دانست.

۱۰.۵.۳ یادگیری دسته قوانین بازگشتی[2]

در بحث بالا از احتمال اینکه عبارت های اضافه شده به بدنه ی قانون خود گزاره ی (predicate) هدف باشند صرف نظر کردیم، مثل عبارت سر قانون. با این وجود، اگر متغیر هدف را به مجموعه ی گزاره ها اضافه کنیم، FOIL با توجه به آن عبارات ممکن را تشکیل می دهد. همین خاصیت به FOIL اجازه می دهد تا توانایی یادگیری قوانین بازگشتی را داشته باشد، قوانینی که از گزاره های سر و بدنه ی قانون با هم استفاده می کنند. برای مثال، قانون زیر را که تعریفی بازگشتی از رابطه ی Ancestor است را در نظر بگیرید.

IF Parent(x,y)	THEN	Ancestor(x,y)
IF Parent(x,z)∧Ancestor(z,y)	THEN	Ancestor(x,y)

با داشتن مجموعه ای از نمونه های آموزشی را می توان با روشی مشابه یادگیری GrandDaughter یاد گرفت. توجه دارید که قانون دوم از جمله قوانینی است که به ذات درون جستجوی FOIL محسوب می شود، Ancestor در مجموعه ی گزاره ها[1] که از آن عبارات جدید قانون

[1] bind
[2] recursive

ایجاد می شود در نظر گرفته خواهد شد. البته اینکه این قانون یادگرفته می شود یا خیر کاملا به عبارت های دیگر ممکن و اینکه آیا مـی تواننـد در جسـتجوی حریصـانه ی FOIL امتیـاز بیشـتری داشـته باشـند بسـتگی دارد. (Cameron-Jones and Quinlar 1993) در مـورد نمونه های بسیاری از کاربرد های موفق FOIL در یادگیری قوانین بازگشتی بحث کرده اند. آنها همچنین در مورد مباحـث دیگـر FOIL چـون چگونگی اجتناب از ایجاد حلقه های بینهایت در قوانین بحث هایی انجام داده اند.

۱۰.۵.٤ خلاصه ی FOIL

به طور خلاصه، FOIL الگوریتم های ترتیبی CN2 را برای کنترل یادگیری دسته قوانین درجه اول مشابه horn clause ها تامیم می دهـد. برای یادگیری هر قانون و در هر مرحله ی اضافه کردن عبارت به شروط قانون، FOIL جستجویی کلی به جزئی انجام می دهد. عبـارات جدیـد اضافه شده به قانون ممکن است در شروط قانون یا حکم قانون موجود باشند، و یا حتی ممکن است متغیر های جدیدی به قانون اضافه کنند. در هر مرحله، از تابع Foil_Gain که در رابطه ی ۱۰.۱ آمده برای انتخاب بین عبارات جدید ممکن استفاده می شود. اگـر عبـارات جدیـد بتواننـد گزاره ی هدف را در بربگیرند، FOIL می تواند، اصولا دسته قوانین بازگشتی را نیز یاد بگیرد. تا جایی که پیچیدگی مانع از ایجاد قوانین با حلقـه بینهایت شود اثبات شده FOIL می تواند با موفقیت دسته قوانین بازگشتی را یاد بگیرد.

اگر داده های آموزشی بدون خطا باشند، FOIL ممکن است آنقدر اضافه کردن قوانین را ادامه دهد تا دیگر هیچ نمونه ی منفی پوشانده شده ای وجود نداشته باشد. برای کنترل داده های خطا دار، جستجو تا زمانی که یک شرط بین دقت، پوشانندگی و پیچیدگی برقرار شود ادامه پیدا خواهد کرد. FOIL برای جلوگیری از رشد بیش از اندازه ی قوانین از روش کوتاهترین توضیح استفاده می کند، روشی کـه در آن عبـارات جدیـد فقـط زمانی اضافه می شوند که طول توضیح آنها از طول توضیح داده های آموزشی ای که توجیه می کنند کمتـر باشـد. جزئیـات ایـن اسـتراتژی در (Quinlar 1990) آمده است. علاوه بر ان، FOIL قوانینی که یاد می گیرد را نیز بعد از یادگیری هرس می کنـد، ایـن روش هـرس کـردن مشابه هرس کردن درخت های تصمیم گیری است (فصل ۳).

۱۰.٦ استقرا به عنوان استنتاج وارونه

روش دوم و کاملا متفاوت برای برنامه نویسی استقرایی بر اساس مشاهدات ساده ای است که نشان می دهد استقرا (induction) فقط وارون استنتاج (deduction) است! در کل، یادگیری ماشین ساختن تئوری هایی است که برای داده های مشاهده شده توضیح می آورند. با داشـتن مجموعه ی داده های D و دانش قبلی (background knowledge) B، یادگیری را می توان پیدا کردن فرضیه هـایی مثـل h تعریـف کرد که h و B بتوانند با هم دلیلی برای D بیاورند. به عبارت دقیقتر، مثل همیشه فرض کنیـد کـه داده هـای آموزشـی D کـه مجموعـه ای از نمونه های آموزشی به فرم $< x_i, f(x_i) >$ است در اختیار است. در اینجا x_i نشان دهنده ی i امـین نمونـه ی آموزشـی و $f(x_i)$ مقـدار هدف آن است. یادگیری، مسئله ی پیدا کردن فرضیه ای مثل h است که بتوان دسته بندی $f(x_i)$ برای x_i را از h و مشخصات x_i و دانش قبلی B نتیجه گیری کرد.

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i) \qquad (10.2)$$

---

[1] Predicates

عبارت $X \vdash Y$ را بخوانید "Y را می توان از X نتیجه گرفت (follows deductively)" یا "X موجب Y (entails) می شود". رابطه ی ۱۰.۲ محدودیت هایی که لازم است که h داشته باشد تا بتوان از B و h و x_i، $f(x_i)$ را نتیجه گرفت توصیف می کند.

برای مثال، مسئله ای را در نظر بگیرید که در آن مفهوم هدف "جفت افرادی مثل <u,v> است که در آن v فرزند u است"، و آنرا با Child(u,v) نشان می دهیم. فرض کنید که فقط یک نمونه ی مثبت به ما داده می شود Child(Bob, Sharon)، و در آن نمونه ها به صورت Male(Bob)، Female(Sharon) و Father(Sharon,Bob) توصیف شده اند. علاوه بر آن فرض کنید که دانش قبلی داریم که $Parent(u,v) \leftarrow Father(u,v)$. می توان وضعیت عبارات رابطه ی ۱۰.۲ را به صورت زیر مشخص کرد:

$$x_i: Male(Bob), Female(Sharon), Father(Sharon, Bob)$$

$$f(x_i): \qquad Child(Bob, Sharon)$$

$$B: \qquad Parent(u,v) \leftarrow Father(u,v)$$

دو فرضیه از فرضیه های ممکنی که در رابطه ی $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می کنند در زیر آورده شده اند،

$$h_1: Child(u,v) \leftarrow Father(v,u)$$
$$h_2: Child(u,v) \leftarrow Parent(v,u)$$

توجه دارید که عبارت هدف Child(Bob, Sharon) بدون استفاده از B و تنها از $h_1 \wedge x_i$ نتیجه گرفته می شود. اما در مورد فرضیه ی h_2، اوضاع متفاوت است، عبارت هدف Child(Bob, Sharon) از $B \wedge h_2 \wedge x_i$ نتیجه گیری خواهد شد و نمی توان آنرا از $h_2 \wedge x_i$ به تنهایی نتیجه گرفت. این مثال، نقش دانش قبلی در گسترش مجموعه ی فرضیه های قابل قبول برای یک مجموعه ی معلوم از داده های آموزشی را مشخص می کند. همچنین، نشان می دهد که چگونه متغیر های جدید (مثل Parent) را می توان به فرضیه ها (مثل h_2) حتی زمانی که متغیر در توصیف نمونه ی x_i دخیل نیست معرفی کرد. این فرایند افزودن (augmenting) مجموعه ای از متغیر ها بر اساس دانش قبلی استقرای ساختاری (constructive induction) نامیده می شود.

این اهمیت رابطه ی (۱۰.۲) است که مسئله ی یادگیری را در محیط استنتاج استقرایی و منطق بیان می کند. در بحث ما این منطق، منطق گزاره ای و منطق درجه اول خواهد بود و الگوریتم های راحت الدرک برای اتوماتیک کردن استنتاج خواهد بود. جالب است که ایجاد عکس این فرایند ها برای اتوماتیک کردن فرایند تعمیم استقرایی ممکن است. به نظر می رسد، این دید که استقرا را می توان با عکس کردن استنتاج بدست آورد را اولین بار در قرن نوزدهم W. S. Jevons مشاهده کرد، وی می نویسد:

استقرا در حقیقت عکس عمل استنتاج است و نمی توان بدون این ارتباط وجودش را تصور کرد، بنابراین سوال اهمیت پیش نخواهد آمد. چه کسی فکر خواهد کرد که جمع یا تفریق در ریاضیات مهم تر است؟ اما در سهولت تفاوت زیادی بین یک عمل و عکسش وجود دارد؛ ... باید در نظر گرفته شود که انجام عمل استقرا بسیار سخت تر و پیچیده تر از عمل استنتاج است ... (Jevons 1874)

در ادامه ی این فصل به استقرا به دید عکس استنتاج نگاه خواهیم کرد. در اینجا مسئله ی اصلی و مورد توجه طراحی عکس عملگر های نتیجه گیری (entailment operators) است. یکی از عملگر های عکس نتیجه گیری، O(B, D) است که نمونه های آموزشی $D = \{<x_i, f(x_i)>\}$ و دانش قبلی B را دریافت کرده و فرضیه ای خروجی می دهد که در رابطه ی ۱۰.۲ صدق کند.

$$O(B,D) = h \text{ such that } (\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

البته در کل، ممکن است فرضیه های مختلفی برای h وجود داشته باشد که $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$. یکی از روش های ILP برای انتخاب میان چنین فرضیه هایی استفاده از قانون کوتاهترین توضیح است (برای اطلاعات بیشتر به بخش ۶.۶ مراجعه کنید).

روشهای جذاب دیگر برای فرمولی کردن کار یادگیری به عنوان پیدا کردن فرضیه ای مثل h که بتواند در رابطه ی $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق کند وجود دارد.

این فرمولی کردن تعریف های معمول یادگیری مفاهیم را به عنوان پیدا کردن مفهومی کلی که با مجموعه ای از نمونه های آموزشی مطابقت داشته باشد را شامل می شود (این تعریف معمول مشابه حالتی است که هیچ دانش قبلی ای نداشته باشیم.).

یکی کردن نماد دانش قبلی B، با این فرمول تعریف غنی تری از تناسب یک فرضیه به ما می دهد. تا به حال، تناسب فرضیه (مثل شبکه ای عصبی) را فقط وابسته به مشخصات فرضیه و داده ها و مستقل از محیط مطالعه فرض می کردیم. در مقابل، این فرمول اجازه می دهد تا اطلاعات فضای عمل که با دانش قبلی B مشخص می شود را جزو تعریف "تناسب" (fit) قرار دهیم. در کل، h با نمونه ای مثل $< x_i, f(x_i) >$ متناسب است هر گاه بتوان $f(x_i)$ را از $B \wedge h \wedge x_i$ نتیجه گرفت.

با تاثیر دادن دانش قبلی B، این فرمول متد های یادگیری ای را می طلبد که به جای جستجوی کورکورانه ی فضای فرضیه ای از دانش قبلی برای هدایت جستجوی h استفاده کنند. فرایند دقت عکس (inverse resolution) که در قسمت های بعدی توضیح داده می شود از دانش قبلی برای این کار استفاده خواهد کرد.

به طور همزمان، تحقیقات بر روی استقرای برنامه نویسی منطقی نشان می دهد که این فرمولی کردن با چندین مشکل کاربردی روبروست:

الزامات لازم برای $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ به طور طبیعی با داده های آموزشی خطا دار سازگار نیست. مشکل اینجاست که این عبارت اجازه ی وجود خطای احتمالی در داده های مشاهده شده نمونه های x_i یا مقدار تابع هدفشان $f(x_i)$ را به ما نمی دهد. وجود چنین خطاهایی ممکن است محدودیت های متناقض در h ایجاد کند. متاسفانه، اکثر محیط های منطقی رسمی با دادن مجموعه ای متناقض از ادعاها (assertion) کارایی خود را در تمیز دادن مقدار واقعی و غیر واقعی از دست می دهند.

زبان منطقی درجه اول آنقدر شامل است که تعداد فرضیه هایی که در عبارت $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می کنند بسیار زیاد است که جستجو میان این فضای فرضیه ها در حالت کلی رام نشدنی (intractable) است.

اکثر تحقیقات فعلی در فرم های محدود قوانین درجه اول بررسی می شوند یا از دانش قبلی درجه دوم نیز استفاده می گردد تا بتوان فضای فرضیه ای را برای جستجو ساده تر کرد.

بر خلاف این تصور که دانش قبلی باید به جستجو برای یک فرضیه کمک کند، در اکثر سیستم های ILP (شامل تمامی سیستم های بحث شده در این فصل) پیچیدگی جستجو فضای فرضیه ای با افزایش دانش قبلی افزایش می یابد. (با این وجود، برای الگوریتم هایی که دانش قبلی پیچیدگی جستجوی فضای فرضیه ای را کم می کنند به فصل ۱۱ و ۱۲ مراجعه کنید)

در قسمت بعدی، یکی از روشهای کلی عکس عملگر نتیجه گیری را که با عکس کردن استنتاج فرضیه هایی ایجاد می کند را بررسی خواهیم کرد.

۱۰.۷ دقت عکس

متدی کلی ای برای اتوماتیک کردن استنتاج قانون دقت (resolution rule) است که توسط (Robinson 1965) پیشنهاد شد. قانون دقت قانونی کامل برای استنتاج در منطق درجه اول است. بنابراین، جای این سوال هست که بپرسیم آیا می توان قانون دقت را وارون کرد تا یک عکس عملگر نتیجه گیری بدست آید؟ جواب آری است، وارون این قانون عملگری است که پایه ی برنامه ی Cigol را تشکیل می دهد که توسط (Muggleton and Buntine 1988) ارائه شد.

بهتر است که قانون دقت را در فرم گزاره ای معرفی کنیم تا برای تعمیم بر روی نمایش درجه اول آماده باشد. اگر L یک عبارت گزاره ای دلخواه باشد و P و R نیز دو حکم گزاره ای دلخواه باشند قانون دقت به صورت زیر خواهد بود،

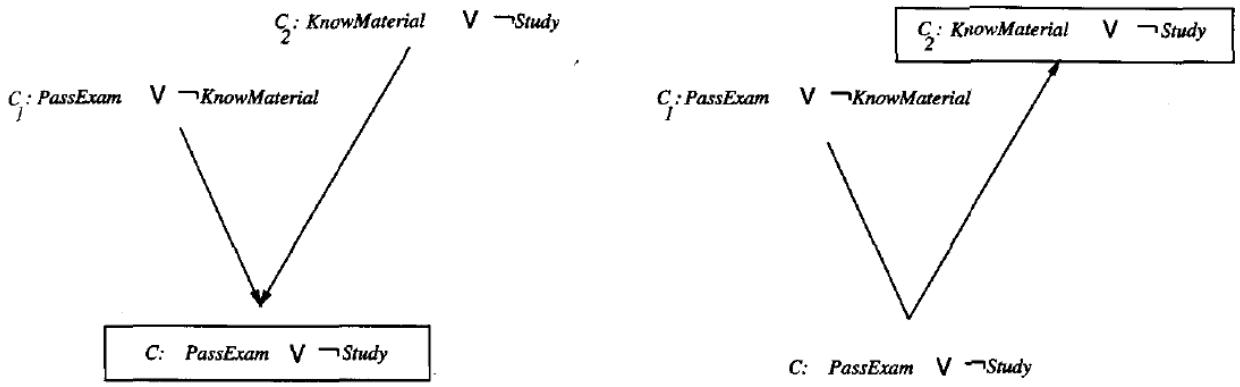
$$P \vee L$$

$$\neg L \vee R$$

$$P \vee R$$

آنرا باید به صورت روبرو خواند: با داشتن دو حکم (clause) بالایی، حکم پایین نتیجه گیری می شود. با داشتن دو ادعای $P \vee L$ و $\neg L \vee R$ واضح است که یکی از دو عبارت L یا ¬L غلت است، بنابراین حکم $P \vee R$ قانون دقت درست خواهد بود.

فرم کلی گزاره ای عملگر دقت در جدول ۱۰.۵ آورده شده است. با داشتن دو عبارت C_1 و C_2 عملگر دقت ابتدا عبارتی مثل L را مشخص می کند که عبارتی مثبت در یکی از عبارت و عبارتی منفی در عبارت دیگر است. سپس نتایج فرمول بالا را می کشد. برای مثال، عملکرد عملگر دقت را در شکل سمت راست شکل ۱۰.۲ در نظر بگیرید. با داشتن C_1 و C_2، مرحله ی اول فرایند عبارت $L = \neg KnowMaterial$ را مشخص می کند که در C_1 آمده و عکسش $\neg(\neg KnowMaterial) = KnowMaterial$ در C_2 آمده است. بنابراین نتیجه حکمی است که از ترکیب این دو عبارت بوجود می آید $C_1 - \{L\} = PassExam$ و $C_2 - \{\neg L\} = \neg Study$ است. به عنوان مثالی دیگر، نتیجه ی اعمال قانون دقت را به دو عبارت $C_1 = A \vee B \vee C \vee \neg D$ و $C_2 = \neg B \vee E \vee F$ را که $A \vee C \vee \neg D \vee E \vee F$ است.



شکل ۱۰.۲ در سمت چپ کاربردی از (استنتاج) قانون دقت برای بدست آوردن C از C_1 و C_2 آمده است.
در سمت راست کاربردی از (استقرا) عکس آن عمل و بدست آوردن C_2 از C و C_1 آورده شده است.

با داشتن عبارات C_1 و C_2 عبارتی مثل L از حکم C_1 پیدا کن که $\neg L$ در C_2 آمده باشد.
C را با استفاده از تمامی عبارات C_1 و C_2 به غیر از L و $\neg L$ تشکیل بده. بـه عبـارت دقیقتـر، مجموعـه عبـارات C را می توان از رابطه ی زیر بدست اورد،

$$C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

در اینجا U نشان دهنده ی اجتماع مجموعه ها و "-" نشان دهنده ی اختلاف مجموعه هاست.

جدول ۱۰.۵ عملگر دقت (فرم گزاره ای).
با داشتن دو حکم $\boldsymbol{C_1}$ *و* $\boldsymbol{C_2}$ *می توان با استفاده از عملگر دقت Cای را پیدا کرد که* $\boldsymbol{C_1 \wedge C_2 \vdash C}$.

وارون کردن عملگر دقت برای ساخت یک عکس عملگر نتیجه گیری $O(C, C_1)$ که اعمال استقرایی انجام می دهد بسیار ساده است. در کل، عکس عملگر نتیجه گیری باید بتواند یکی از حکم های اولیه C_2 را با بازگشایی (resolve) C و دیگر حکم C_1 بدست آورد. مثالی را فـرض کنید که در آن حکم نهایی $C = A \vee B$ و حکم اولیه $C_1 = B \vee D$. چگونه می توان حکمی مثل C_2 را پیدا کرد که $C_1 \wedge C_2 \vdash C$؟ اول اینکه باید توجه داشت که طبق تعریف عملگر دقت هر عبارتی که در C هست ولی در C_1 نیست حتما در C_2 موجود خواهد بـود. در مثـال مـا، این نشان می دهد که C_2 حتما عبارت A را خواهد داشت. دوم اینکه عبارتی که در C_1 وجود دارد اما در C وجود ندارد توسط قانون دقت حذف شده است، بنابراین عکس آن باید در C_2 حضور داشته باشد. در مثال ما، این نشان می دهـد کـه C_2 بایـد حتمـا عبـارت ¬D را داشـته باشـد. بنابراین خواهیم داشت که $C_2 = A \vee \neg D$. واضح است که از ترکیب دو قانون C_1 و C_2 بدست آمده C نتیجه گرفته خواهد شد.

با معلوم بودن حکم های C و C_1 عبارتی مثل L را پیدا کن که در C_1 موجود باشد اما در C موجود نباشد.
حکم دوم مثل C_2 را که عبارات زیر را شامل می شود را تشکیل بده

$$C_2 = \big(C - (C_1 - \{L\})\big) \cup \{\neg L\}$$

جدول ۱۰.۶ عکس عملگر دقت (فرم گزاره ای).
با داشتن دو حکم C و $\boldsymbol{C_1}$ *می توان حکمی مثل* $\boldsymbol{C_2}$ *را پیدا کرد که* $\boldsymbol{C_1 \wedge C_2 \vdash C}$.

توجه دارید که در مثال بالا جواب دیگری برای C_2 وجود دارد. در کل، C_2 را می توان به صورت خاص تر $A \vee \neg D \vee B$ دانست. تفاوت میان این جواب و جواب اول در این است که در اینجا عبارتی که در C_1 ظاهر شده بود را به C_2 اضافه کرده ایم. نکته ی کلی اینجا ایـن اسـت کـه قانون عکس دقت قطعی نیست، در کل ممکن است بیش از یک C_2 وجود داشته باشد که بتوان از C_1 و C_2، C را نتیجه گرفـت. روش مـا در انتخاب بین جواب های مختلف ارجه دانستن حکم هایی است که طول توضیح کمتری دارند، یا به طور مشابه، فرض می کنیم C_2 هیچ عبارتی مشترک با C_1 ندارد. اگر این بایاس به سمت حکم های کوتاهتر را با قانون عکس دقت ترکیب کنیم الگوریتم جدول ۱۰.۶ بدست خواهد آمد.

حال می توانیم بر اساس عکس عملگر نتیجه گیری الگوریتم های یادگیری قـانونی مثـل دقـت معکـوس را طراحـی کنـیم. در کـل، الگـوریتم یادگیری می تواند از عکس نتیجه گیری برای ساختن فرضیه هایی که بتوانند به همراه دانش قبلی داده های آموزشی را نتیجـه دهنـد تشـکیل دهند. یکی از استراتژی های ممکن استفاده از الگوریتم های ترتیبی برای یادگیری حلقه ای دسته ای از horn clause با این روش است. در هر حلقه، الگوریتم نمونه ی آموزشی ای مثل $< x_i, f(x_i) >$ را که هنوز پوشانده نشده است انتخاب می کند. سپس از قانون عکـس دقـت برای ایجاد فرضیه ای ممکن مثل h_i که در رابطه ی $(B \wedge h_i \wedge x_i) \vdash f(x_i)$ صدق کند استفاده می شـود، در اینجـا B دانـش قبلـی بـه علاوه ی تمامی قوانین قبلی یادگرفته شده است. توجه دارید که این جستجویی بر پایه نمونـه هاسـت (example-driven)، زیـرا کـه هـر فرضیه ممکن برای پوشاندن یک نمونه ی خاص ایجاد شده است. البته اگر چندین فرضیه موجود باشد (که یک نمونـه را بپوشـانند) فرضـیه ای

ارجه تر خواهد بود که دقت بهتری بر روی دیگر نمونه های پوشانده شده اش داشته باشد. برنامه ی Cigol از عکس دقت به علاوه ی نوعی الگوریتم ترتیبی به همراه تعامل با کاربر برای بدست آوردن نمونه های آموزشی بیشتر برای بدست آوردن راهنمایی در جستجوی میان فضای وسیع مراحل استقرا استفاده می کند. با این وجود Cigol بیشتر از قوانین درجه اول به جای نمایش گزاره ای استفاده می کند. در زیر تعمیم قانون دقت لازم برای سازگاری با نمایش درجه اول را توصیف خواهیم کرد.

۱۰.۷.۱ دقت در نمایش درجه اول

قانون دقت به راحتی به قوانین درجه اول تعمیم پیدا می کند. مشابه حالت گزاره ای، این فرایند دو حکم به برای ورودی دریافت کرده و یک حکم خروجی می دهد. تفاوت کلیدی با حالت گزاره ای در این است که این فرایند در نمایش درجه اول بر اساس یکتاکردن (unifying) جانشینی ها (substitution) انجام می شود.

هر نگاشت (mapping) از متغیر ها به جملات را جانشینی می نامیم. برای مثال، جانشینی $\theta = \{x/Bob, y/z\}$ در متغیر x مقدار عبارت Bob و در متغیر y مقدار عبارت z را قرار می دهد. از نماد $W\theta$ برای نمایش نتیجه ی اعمال جانشینی θ در عبارت W استفاده می کنیم. برای مثال، اگر L عبارت Father(x, Bill) باشد و θ همان جانشینی مثال قبلی باشد داریم، Lθ=Father(x, Bill).

زمانی که می گوییم θ جانشینی ای یکتاست که برای دو عبارت L_1 و L_2 داشته باشیم $L_1\theta = L_2\theta$. برای مثال، اگر $L_1 = Father(x, y)$ و $L_2 = Father(Bill, z)$ باشد و $\theta = \{x/Bob, y/z\}$، θ یک جانشینی یکتا برای L_1 و L_2 است زیرا که $L_1\theta = L_2\theta = Father(Bill, y)$. اهمیت جانشینی یکتا در این است که در فرم گزاره ای دقت، می توان با پیدا کردن عبارتی مثل L که L در C_1 و $\neg L$ نیز در C_2 باشد نتیجه ی دو حکم C_1 و C_2 را تعیین کرد. در دسته قوانین درجه اول، این تعمیم پیدا کردن عبارتی مثل L_1 از C_1 و L_2 از C_2 است که بتوان جانشینی ای مثل θ پیدا کرد که برای L_1 و $\neg L_2$ یکتا باشد ($L_1\theta = \neg L_2\theta$).

قانون دقت سپس حکم C را از رابطه ی زیر تشکیل می دهد.

$$C = (C_1 - \{L_1\}\theta \cup (C_2 - \{L_2\})\theta \qquad (10.3)$$

حالت کلی عبارت دقت در جدول ۱۰.۷ آمده است. برای تصور، فرض کنید که $C_1 = White(x) \leftarrow Swan(x)$ و با فرض اینکه $C_2 = Swan(Fred)$. عبارت $C_1 = White(x) \vee \neg Swan(x)$ بدست خواهد امد. حال می توان قانون دقت را اعمال کرد. بنابراین، نتیجه ی C از ترکیب دو عبارت $(C_1 - \{L_1\})\theta = White(Fred)$ و $(C_2 - \{L_2\})\theta = \emptyset$ یا C=White(Fred).

عبارتی مثل L_1 از C_1 و L_2 از C_2 و جانشینی θ را پیدا کن که $L_1\theta = \neg L_2\theta$.

نتیجه ی C را با اجتماع تمامی عبارات $C_1\theta$ و $C_2\theta$ به جز $L_1\theta$ و $\neg L_2\theta$ را تشکیل بده. به عبارت دقیقتر مجموعه ی عباراتی که در C خواهند بود به صورت زیرند،

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

جدول ۱۰.۷ قانون دقت (فرم درجه اول)

۱۰.۷.۲ وارون کردن دقت: حالت درجه اول

به صورت تحلیلی می توان مشتق عکس قانون دقت را با دستکاری رابطه ی ۱۰.۳ که تعریف قانون دقت است بدست آورد. ابتدا توجه داشته باشید که θ در رابطه ی ۱۰.۳ را می توان به صورت یکتا به θ_1 و θ_2 تجزیه کرد که $\theta = \theta_1\theta_2$، در این رابطه θ_1 شامل تمامی جانشینی های مربوط به متغیر های C_1 و θ_2 شامل تمامی جانشینی های مربوط به متغیر های C_2 می شود. این تجزیه برای این عملی است، که C_1 و C_2 همیشه با متغیر های خالص (distinct) شروع می شوند (زیرا که این دو به صورت جهانی جملات مستقل اند--). با این تجزیه θ می توان رابطه ی ۱۰.۳ را به صورت زیر بازنویسی کرد،

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$$

همیشه توجه داشته باشید که علامت "-" در اینجا به معنای اختلاف مجموعه هاست. حال اگر عکس عملگر دقت فقط حکم های C_2 ی را که اشتراکی با C_1 ندارند خروجی دهد (طبق قانون کوتاهترین طول توضیح) می توان عبارت بالا را به صورت زیر باز نویسی کرد،

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

حال بلاخره از این حقیقت که طبق تعریف داریم $L_2 = \neg L_1\theta_1\theta_2^{-1}$ استفاده کرده و C_2 را برای بدست می آوریم،

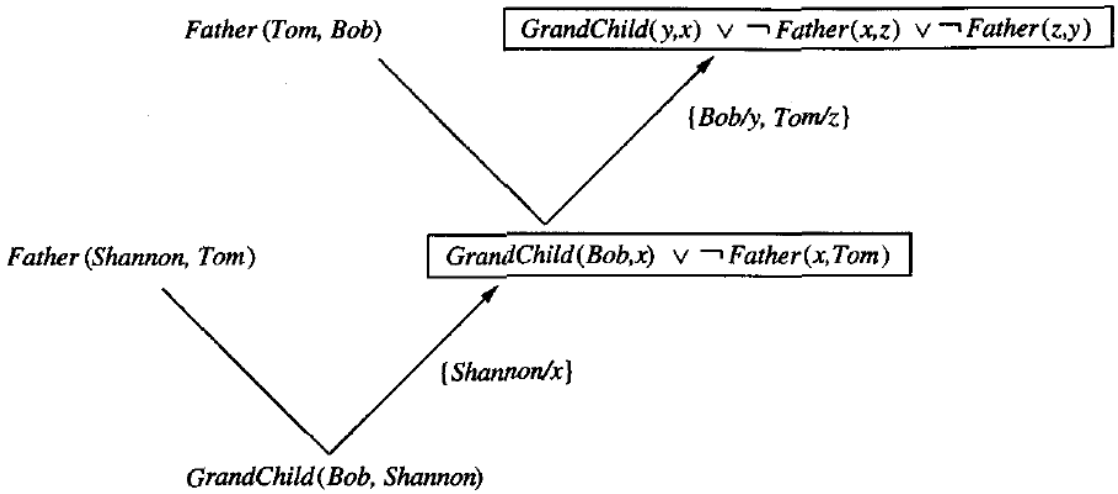
عکس دقت:

$$C_2 = (C - (C_1 - \{L_1\})\Theta_1)\Theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\} \qquad (10.4)$$

رابطه ی ۱۰.۴ عکس قانون دقت را برای منطق درجه اول نشان می دهد. مشابه فرم گزاره ای، عملگر عکس نتیجه گیری غیر قطعی (nondeterministic) است. در کل، در اعمال این عملگر باید در حالت کلی انتخاب های مختلف برای C_1 برای حل و جانشینی یکتا ی Θ_1 و Θ_2 وجود دارد. هر یک از این انتخاب ها ممکن است به جوابی خاص برای C_2 ختم می گردد.

شکل ۱۰.۳ اعمال چند مرحله ای عکس قانون دقت را برای یک مثال ساده نشان می دهد. در این شکل، هدف یادگیری قوانینی برای GrandChild(y,x) با داشتن داده های آموزشی D=GrandChild(Bob,Shannon) و دانش قبلی B={Father(Shannon,Tom), Father(Tom,Bob)} است. به بالاترین مرحله ی درخت عکس قانون دقت در شکل ۱۰.۳ توجه کنید. در اینجا، حکم C را برای نمونه ی آموزشی GrandChild(Bob, Shannon) را بررسی کرده و حکم $C_1 = Father(Shannon, Tom)$ را از دانش قبلی بر می گزیند. برای اعمال عکس عملگر دقت فقط یک انتخاب، Father(Shannon,Tom)، برای L_1 وجود دارد. فرض کنید که عکس جانشینی ها را به صورت $\Theta_1^{-1} = \{\}$ و $\Theta_2^{-1} = \{Shannon/x\}$. در این حالت، حکم نتیجه ی C_2 از ترکیب حکم $(C - (C_1 - \{L_1\})\Theta_1)\Theta_2^{-1} = (C\Theta_1)\Theta_2^{-1} =$ $GrandChild(Bob, x)$ و حکم $\{\neg L_1\theta_1\theta_2^{-1}\} = \neg Father(x, Tom)$. بنابراین نتیجه ی حکم GrandChild(Bob,Shannon)∨¬Father(x,Tom) یا به طور مترادف (GrandChild(Bob,x) ←Father(x,Tom)) خواهد بود. توجه داشته باشید که این قانون کلی به اضافه ی C_1 نمونه ی آموزشی GrandChild(Bob,Shannon) را نتیجه خواهد داد.

Telegram: @Farabar_BI
www.farabar.net



شکل ۱۰.۳ یک فرایند عکس دقت چند مرحله ای.

در این مثال، حکم های درون مستطیلها حاصل از مراحل استناجی هستند. برای هر مرحله، C حکمی است که در بالا نشان داده شده، C_1 حکم سمت چپ و C_2 حکم داخل مستطیل در راست است. در هر دو مرحله ی استنتاجی، θ_1 جانشینی تهی {} است، و θ_2^{-1} نیز جانشینی نشان داده شده در زیر C_2 است. توجه دارید که نتیجه نهایی (مستطیل گوشه ی بالا و راست) فرم جایگزین GrandChild(y,x)←Father(x,y)∧Father(z,y) است.

به طور مشابه، این حکم نتیجه گیری شده را می توان به عنوان نتیجه ی C برای مرحله ی دوم عکس دقت به کار برد، مشابه شکل ۱۰.۳. در هر مرحله، توجه دارید که چندین حالت نتیجه ی ممکن وجود دارد، این نتایج به انتخاب جانشینی وابسته اند (تمرین ۱۰.۷). در مثال شکل ۱۰.۳، مجموعه ی خاصی از انتخاب ها به حکم راضی کننده ی GrandChild(y,x)←Father(x,y)∧Father(z,y) می رسد.

۱۰.۷.۳ خلاصه ی وارون کردن دقت

به طور خلاصه، عکس دقت روشی کلی برای ایجاد اتوماتیک فرضیه ی h است که در رابطه ی $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق کند. این کار با عکس قانون کلی دقت (که در رابطه ی ۱۰.۳ آمده) صورت می گیرد. با شروع از بازگشایی قانون و حل آن برای حکم C_2، عکس قانون دقت در رابطه ی ۱۰.۴ بدست می آید.

با داشتن مجموعه ای از حکم ها، ممکن است با بکار بردن عکس قانون دقت چندین فرضیه بدست آید. توجه دارید که عکس قانون دقت این مزیت را دارد که فقط فرضیه هایی که در $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می کنند را ایجاد می کند. در مقابل جستجوی آزمون و خطایی FOIL در هر مرحله از جستجو تعداد زیادی فرضیه که بعضیشان این شرط را ارضا نمی کنند ایجاد می کند. سپس FOIL از داده های D برای انتخاب بین این فرضیه ها استفاده خواهد کرد. با دانستن این تفاوت، انتظار می رود که جستجو ی مبتنی بر عکس قانون دقت متمرکز تر و کارا تر باشد. با این وجود، این نتیجه گیری همیشه برقرار نیست. یکی از دلایل این است که عکس عملگر دقت فقط کسر کوچکی از داده های موجود را در هر مرحله از ایجاد فرضیه اش مد نظر قرار می دهد، در حالی که، FOIL تمامی داده های ممکن را برای انتخاب میان فرضیه های نحوی (syntactical) ایجاد شده مد نظر قرار می دهد. تفاوت های بین استراتژی هایی که از عکس استنتاج استفاده می کنند و استراتژی هایی که از جستجوی آزمون و خطایی استفاده می کنند همچنان موضوعی تحقیقاتی است. (Srinivasan et al. 1995) آزمایشی را که در آن این دو روش مقایسه می شوند مطرح می کند.

www.farabar.net

۱۰.۷.٤ تعمیم، نتیجه گیری Θ و نتیجه گیری

قسمت قبلی به ارتباط بین استقرا و عکس نتیجه گیری پرداخت. با درنظر داشتن اسرار قبلی در استفاده از ترتیب کلی تری برای سازماندهی جستجو ی فضای فرضیه ای، بد نیست که رابطه ی بین روابط کلی تری و عکس نتیجه گیری را مشاهده کنیم. برای درک این رابطه، تعاریف زیر را در نظر بگیرید.

رابطه ی کلی تری. در فصل ۲ رابطه ی کلی تر یا مساوی بودن ($\geq_g$) را به فرم زیر تعریف کردیم: برای دو تابع منطقی مقدار $h_j(x)$ و $h_k(x)$ زمانی می گوییم $h_j(x) \geq_g h_k(x)$ اگر و تنها اگر داشته باشیم که $(\forall x)h_k(x) \rightarrow h_j(x)$. این رابطه ی $\geq_g$ در بسیاری از الگوریتم های یادگیری برای کنترل جستجوی فضای فرضیه ای به کار می رود.

نتیجه گیری Θ (Θ-subsumption).دو حکم C_j و C_k را در نظر بگیرید که هر دو به فرم $HVL_1 \vee \ldots \vee L_n$ هستند و H نیز عبارتی مثبت و L_i نیز عبارات دلخواهی هستند. اگر و فقط اگر جانشینی ای مثل Θ یافت شود که $C_i\Theta \subseteq C_k$، آنگاه حکم C_k زمانی نتیجه گیری Θ ی C_j است. این تعریف را (Plotkin 1970) انجام داده است.

نتیجه گیری (Entailment). دو حکم C_j و C_k را در نظر بگیرید. اگر و فقط اگر C_k را از C_j استنتاج کرد (می نویسیم $C_j \vdash C_k$)، آنگاه C_j از C_k نتیجه گیری می شود.

رابطه ی بین سه تعریف بالا چیست؟ ابتدا بیایید تعریف $\geq_g$ را با نماد گذاری درجه اول مشابه دو تعریف دیگر بازنویسی کنیم. اگر h(x) فرضیه ی منطقی مقدار برای مفهوم هدف c(x)، که در آن h(x) با عطفی از عبارات بیان شده، آنگاه می توان فرضیه را با حکم زیر بازنویسی کرد،

$$c(x) \leftarrow h(x)$$

در اینجا نیز از تفسیر Prolog مبنی بر اینکه اگر نتوان اثبات کرد که x نمونه ای مثبت است، منفی دسته بندی خواهد شد استفاده می کنیم. بنابراین، می توان مشاهده کرد که تعریف قبلی مان از $\geq_g$ به شروط یا بدنه ی Horn clause ها اعمال می شود. حکم ضمنی Horn clause میز مفهوم هدف c(x) است.

رابطه ی این تعریف $\geq_g$ با تعریف نتیجه گیری Θ چیست؟ توجه دارید که اگر $h_1 \geq_g h_2$، آنگاه حکم $C_1: c(x) \leftarrow h_1(x)$ از $C_2: c(x) \leftarrow h_2(x)$ نتیجه گیری Θ می شود. علاوه بر این، نتیجه گیری Θ حتی هنگامی که سر حکم ها متفاوت اند درست است. برای مثال، حکم A در مثال زیر حکم B را نتیجه Θ می دهد:

$$A: Mother(x, y) \leftarrow Father(x, z) \wedge Spouse(z, y)$$

$$B: Mother(x, Louise) \leftarrow Father(x, Bob) \wedge Spouse(Bob, y) \wedge Female(x)$$

زیرا که $A\Theta \subseteq B$ اگر Θ={y/Louise,z/Bob}. تفاوت کلیدی در این است که $\geq_g$ به طور ضمنی فرض می کند که سر دو حکم یکی هستند، در حالی که نتیجه گیری Θ برای حکم هایی که سر های متفاوتی دارند نیز درست است.

بلاخره اینکه، نتیجه گیری Θ حالت خاصی از نتیجه گیری است. بدین معنا که اگر حکم A از حکم B نتیجه گیری Θ شود خواهیم داشت که A⊢B. با این وجود، می توان حکم های A و B را پیدا کرد که A⊢B اما B از A نتیجه گیری Θ نشود. برای مثال زوج مرتب حکم های زیر را در نظر بگیرید:

$$A: Elephant(father_of(x)) \leftarrow Elephant(x)$$

$$B: Elephant(father_of(father_of(y))) \leftarrow Elephant(y)$$

در اینجا تابع father_of(x) تابعی است که بر روی افراد تعریف شده و پدر x را بر می گرداند. توجه دارید که B را می توان از A اثبات کرد اما جانشینی Θ وجود ندارد که نتیجه گیری Θ ی A، B شود.

همانطور که در این مثالها نیز نشان داده شد، نمادگذاری قبلی کلی تر بودن حالت خاصی از نتیجه گیری Θ است که خود نیز حالت خاصی از نتیجه گیری است. بنابراین جستجو ی فضای فرضیه ای با کلی تر یا خاص تر کردن فرضیه ها محدود تر از جستجو با عمگر های عکس نتیجه گیری است. متاسفانه، نمادگذاری متوسط نتیجه گیری Θ، نمادگذاری راحتی را ایجاد می کند که در بین نمادگذاری قبلی مان در رابطه ی کلی تری و نماد گذاری نتیجه گیری است.

۱۰.۷.۵ Prolog

با وجود اینکه عکس دقت متدی فریبنده برای ایجاد فرضیه های ممکن است، در عمل می تواند به راحتی به انفجاری از فرضیه های ممکن تبدیل شود. روش جایگزین دیگر استفاده از عکس نتیجه گیری برای ایجاد تک خاصترین فرضیه است که با دانش قبلی داده های آموزشی را نتیجه بدهند. این خاصترین فرضیه را می توان برای محدود کردن جستجو ی کلی تری در فضای فرضیه ای مشابه فضای فرضیه ای FOIL به کاربرد، با این شرط اضافه که فقط فرضیه های کلی تر از این مرز در نظر گرفته خواهند شد. این روش در سیستم Prolog مورد استفاده قرار گرفته است، الگوریتم Prolog به طور خلاصه در زیر آورده شده:

کاربر با استفاده از زبان عبارات درجه اول محدود شده فضای فرضیه ای H را مشخص می کند. محدودیت های با "نحوه ی تعریف" ایجاد می شود که به کاربر امکان مشخص کردن پیشبینی و نماد های تابع و نوع و فرم آرگومانهای هر کدام را می دهد ---.

Prolog از الگوریتمی ترتیبی برای یادگیری دسته عباراتی از H که داده ها را می پوشانند کمک می گیرد. برای هر نمونه ی $< x_i, f(x_i) >$H که هنوز توسط عبارات یادگرفته شده پوشانده نشده، این سیستم ابتدا به دنبال خاصترین فرضیه ی h_i درون H می گردد که داشته باشیم $(B \wedge h_i \wedge x_i) \vdash f(x_i)$. به عبارت دقیقتر، این سیستم فرضیه را با محاسبه ی خاصترین فرضیه ها در میان فرضیه هایی که از آنها $f(x_i)$ با k بار استفاده از قانون دقت نتیجه گرفته می شود تخمین می زند (k پارامتری است که توسط کاربر تعیین می شود).

Prolog از جستجویی کلی به جزئی در فضای فرضیه ای محدود بین کلی ترین فرضیه ی ممکن و مرز خاصترین h_i، که در مرحله ی ۲ محاسبه شد، استفاده می کند. در این مجموعه ی فرضیه ها، این سیستم به دنبال فرضیه ای است که کمترین طول توضیح (که با تعداد عبارات سنجیده می شود) را داشته باشد. این بخش از جستجو با روش ابتکاری A*-like که هرس بدون ریسک حذف خاصترین فرضیه را ممکن می سازد انجام می گیرد.

جزئیات الگوریتم Prolog در (Muggleton 1992,1995) آورده شده است.

۱۰.۸ خلاصه و منابع برای مطالعه ی بیشتر

نکات اصلی این فصل شامل موارد زیر می شود:

الگوریتم های پوشش ترتیبی فصلی از قوانین را با یادگیری یک قانون دقیق و سپس حذف نمونه های مثبت پوشش داده شده توسط این قانون و تکرار این فرایند تا اینکه تمامی نمونه های مثبت پوشش داده شوند یاد می گیرند. این الگوریتم کارا و حریص برای یادگیری دسته قوانین گزینه ای در مقابل یادگیری درختی بالا به پایین مثل الگوریتم ID3 است که می توان آنرا به صورت پوشش همزمان (در مقابل پوشش ترتیبی) دانست.

در الگوریتم های پوشش ترتیبی متدهای مختلفی را برای یادگیری یک قانون ارائه شده است. این متد ها در استراتژی جستجو برای بررسی فضای شروط ممکن قانون متفاوت اند. یکی از روشهای متداول، که در برنامه ی CN2 نیز به کار رفته، استفاده از جستجوی ستونی و کلی به جزئی است. در این روش قوانین خواص تر ایجاد و مورد بررسی قرار می گیرند تا قانونی به اندازه ی کافی دقیق پیدا گردد. روش های جستجوی فرضیه ای جزئی به کلی دیگر از جستجویی مبتنی بر نمونه ها به جای آزمون و خطا کمک می گیرند و از معیارهای آماری مختلف برای دقت قانون در کنترل جستجو کمک می گیرند.

مجموعه قوانین درجه اول (قوانینی که متغیر نیز دارند) نمایش شاملی دارند. برای مثل، زبان برنامه نویسی Prolog برنامه ها را در حالت کلی با استفاده از مجموعه ای از horn clause های درجه اول نشان می دهد. برای همین گاهی به مسئله ی یادگیری horn clause های درجه اول مسئله ی برنامه نویسی منطقی استقرایی می گویند.

یکی از روشهای یادگیری دسته قوانین درجه اول تعمیم الگوریتم پوشش ترتیبی CN2 از نمایش گزاره ای به نمایش درجه اول است. این روش در برنامه ی FOIL به کار رفته است، این برنامه دسته قوانین درجه اول، شامل جمله قوانین بازگشتی، را یاد می گیرد.

روش دیگری برای یادگیری قوانین درجه اول بر اساس این مشاهده که استقرا عکس استنتاج است می باشد. به عبارت دیگر، مسئله ی استقرا پیدا کردن فرضیه مثل h است که

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

در این رابطه B دانش قبلی کلی است، $x_1 \ldots x_n$ توصیف نمونه های درون داده های آموزشی D هستند و $f(x_1) \ldots f(x_n)$ نیز مقادیر هدف نمونه های آموزشی هستند.

با دنبال کردن دیدگاه استقرا به عنوان عکس استنتاج، بعضی برنامه ها جستجویی برای فرضیه ها با استفاده از عملگری که عملگر معروف استنتاج را عکس می کند انجام می دهند. برای مثال Cigol از عکس دقت، عملگری که عکس عملگر استنتاج دقت متداول در اثبات قضایای مکانیکی است، استفاده می کند. Progol نیز استراتژی عکس استنتاج را با استراتژی جستجوی فضای فرضیه ای کلی به جزئی ترکیب می کند.

کارهای اولیه بر روی یادگیری نسبی توضیحات شامل برنامه ی معروف Winston (1970) برای یادگیری توضیحات به فرم شبکه برای مفاهیمی چون رئیس ("arch")، کار Banerji (1964,1969) و سری کارهای Michalski بر روی برنامه های AQ (Michalski 1960; Michalski et al. 1986) می شود. کارهای Michalski از جمله اولیه کارهایی است که استفاده از نمایش منطقی را در یادگیری بررسی کرد. تعریف Plotkin (1970) از جایگزینی θ رابطه ی اولیه ی بین استقرا و استنتاج را ثابت کرد. همچنین Vere (1975) یادگیری نمایش منطقی را مورد بررسی قرار داد، برنامه ی META-DENDRAL ی Buchanan توضیحات نسبی مربوطه ی طیف جرمی ساختارهای ملکولی --- را یادگرفته. این برنامه به طور موفقیت آمیز در کشف قوانین مفید ای که متاقبا در کتب شیمی منتشر شده است. الگوریتم Candidate-Elimination vertion space ی Mitchell در رابطه های مشابه ساختارهای شیمی به کار رفته است.

با رواج زبان Prolog در اواسط دهه ی 1980، تحقیقات به سمت یادگیری توضیحات نسبی بیان شده با دسته Horn clause ها رفت. کارهای اولیه بر روی horn clause ها شامل برنامه ی MIS از Shapiro (1983) و MARVIN از Sammut and Banerji (1986) می شود. الگوریتم FOIL از Quinlan (1990) که در فصل نیز مورد بررسی قرار گرفت، به سرعت با الگوریتم های جستجوی کلی به جزئی برای قوانین درجه اول شامل MFOIL از Dzeroski (1991)، FOCL از Pazzani et al. (1991)، CLAUDIEN از De Raedt and Bruynooghe (1993) و MARKUS از Grobelnik (1992). الگورتیم FOCL در فصل ۱۲ مورد بررسی قرار گرفته است.

خط دیگر تحقیقی از یادگیری horn clause با عکس عمل استنتاج توسط Muggleton and Buntine (1988) پیشنهاد شد که بر اساس ایده های مشابه Sammut and Banerji (1986) و Muggleton (1987) ایجاد شده بود. کارهای اخیر در این حوضه بر دیگر استراتژی های جستجو و متدهای تمرکز فضای فرضیه برای ایجاد یادگیری مهار شده است. برای مثال، Kietz and Wrobel (1992) از قانون schemata در برنامه ی RDT خود که فرم بیان در نظر گرفته شده در طول یادگیری را محدود می کند استفاده کرده اند، Muggleton and Feng (1992) نیز محدودیت نمایش درجه اول را به ij-determinate را بررسی کرده اند. Cohen (1994) برنامه ی کلی GRENDEL را که مجموعه ای از توضیحات محض را درباره ی زبان توصیف بدنه ی قانون دریافت کرده و به کاربر اجازه می دهد تا فضای فرضیه ای را به طور دلخواه محدود کند را مورد بحث قرار داده است.

Lavrac and Dzeroski (1994) کتابی ساده ای را درباره ی برنامه نویسی استقرایی منطقی ارائه می کنند. دیگر کتب می توان به Bergadano and Gunetti (1995)، Morik et al. (1993)، Muggleton (1992,1995b) اشاره کرد. فصل مربوطه ی Wrobel (1996) نیز دید خوبی در این زمینه ارائه می کند. Bratko and Muggleton (1995) خلاصه ی تعدادی از کاربرد های اخیر ILP در مسائل کاربردی مهم را مطرح می کنند. مجموعه ای از کارگاههای ILP سالانه منبع خوبی از تحقیقات اخیر در این زمینه ارائه می کنند (به De Raedt (1996) رجوع کنید).

تمرینات

۱۰.۱ الگوریتمی پوشش ترتیبی مشابه CN2 و الگوریتمی پوشش همزمان مثل ID3 را در نظر بگیرید. هر دو الگوریتم برای یادگیری مفهوم هدف تعریف شده روی نمونه های توصیفی با عصف n متغیر منطقی به کار می روند. اگر ID3 درخت تصمیم متقارنی با عمق d را یاد بگیرد این درخت $2^d - 1$ گره تصمیم مستقل خواهد داشت، و بنابراین $2^d - 1$ انتخاب در هنگام ساخت فرضیه خروجی انجام می دهد. چه تعداد قانون برای نمایش چنین درختی به عنوان فصل دسته قوانین لازم است؟ هر یک از این قوانین چندین شرط خواهند داشت؟ یک الگوریتم پوشش ترتیبی چه تعداد انتخاب مستقل باید انجام دهد تا همین دسته قوانین را ایجاد کند؟ فکر می کنید کدام سیستم با داده های آموزشی یکی، بیشتر تمایل به overfit خواهد داشت؟

۱۰.۲ الگوریتم learn-one-rule در جدول ۱۰.۲ را چنان تغییر دهدی که بتواند قوانینی که شروطشان مقایسه ی ویژگی ها با اعداد حقیقی است را نیز یاد بگیرد (مثل temperature>42). الگوریتم خود را با تغییراتی که باید به جدول ۱۰.۲ اعمال شود مشخص کنید. راهنمایی: به همین تعمیم در یادگیری درختی توجه کنید.

۱۰.۳ الگوریتم learn-one-rule در جدول ۱۰.۲ را چنان تغییر دهدی که بتواند قوانینی که شروطشان عضویت در مجموعه ی معلومی است را نیز یاد بگیرد (مثل $\text{nationality} \in \{Canadian, Brazilian\}$). الگورتیم تغییر یافته ی شما باید تمامی فرضیه های ممکن شامل چنین زیرمجموعه هایی را جستجو کند. الگوریتم خود را با تغییراتی که باید به جدول ۱۰.۲ اعمال شود مشخص کنید.

۱۰.۴ استفاده از learn-one-rule در ایجاد استراتژی جستجوی فضای فرضیه ای را در نظر بگیرید. در کل، ویژگی های جستجوی زیر را در نظر بگیرید:

(a) آزمون و خطا در مقابل کنترل شده با داده ها

(b) کلی به جزئی در مقابل جزئی به کلی

(c) پوشش ترتیبی در مقابل پوشش همزمان

در مورد مزیت های انتخابهای الگوریتم جدول ۱۰.۱ و ۱۰.۲ بحث کنید. برای هر یک از این سه ویژگی استراتژی جستجو، تاثیر مثبت یا منفی آن را بحث کنید.

۱۰.۵ از قانون عکس دقت در فرم گزاره ای بر روی گزاره های $C = A \vee B$ و $C_1 = A \vee B \vee G$ اعمال کنید. حداقل دو جواب برای C_2 ارائه کنید.

۱۰.۶ از قانون عکس دقت در فرم گزاره ای بر روی گزاره های $C = R(B,x) \vee P(x,A)$ و $C_1 = S(B,y) \vee R(z,x)$ اعمال کنید. حداقل چهار جواب برای C_2 ارائه کنید. A و B ثابت و x و y متغیر هستند.

۱۰.۷ اولین مرحله ی عکس قانون دقت را در شکل ۱۰.۳ را در نظر بگیرید. حداقل دو حاصل مختلف -----

۱۰.۸ روابط تعریفی مسئله ی استقرا در این فصل را در نظر بگیرید:

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

حال تعریف قبلی مان از بایاس استقرایی در فصل ۲ را نیز در نظر بگیرید، رابطه ی ۲.۱. در آنجا B_{bias} را با رابطه ی زیر تعریف کردیم:

$$(\forall x_i \in X)(B_{bias} \wedge D \wedge x_i) \vdash L(x_i, D)$$

در این رابطه $L(x_i, D)$ دسته بندی ای است که یادگیر برای نمونه ی x_i بعد از مشاهده ی داده های آزمایشی D انجام می دهد و X کل فضای نمونه ای است. توجه دارید که رابطه ی اول قصد دارد که فرضیه ای را که یادگیر علاقه به پیدا کردنش را دارد توصیف کند در حالی که رابطه ی دوم قصد دارد خط مشی یادگیر را برای تعمیم فرای داده های آموزشی را تعیین کند. یادگیری ایجاد کنید که بایاس استقرایی اش دقیقا مشابه دانش قبلی موجود B باشد.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

horn clause درجه اول	first-order horn clase

inductive logic programming (ILG)	برنامه نویسی منطقی
inductive logic programming	برنامه نویسی استقرایی منطقی
simultaneous covering	پوشش همزمان
Implementing	تعریف
relative frequency	تکرار نسبی
mechanical theorem provers	ثابت کننده ی تئوری
beam search	جستجوی ستونی
Clause, postcondition	حکم
sets of rules	دسته قوانین
Resolution	دقت
propositional representation	روش های گزاره ای
Syntax	زبان
Subroutine	زیر روال
Precondition	شرط
Literal	عبارت
deductive operator	عملگر های نتیجه گیری
Expressive	قابلیت بیان
Expressive	قوی تر در بیان
Performance	کارایی
example driven	کنترل نمونه ای
covering	الگوریتم پوششی
sequential covering algorithms	الگوریتم های ترتیبی
candidate	ممکن
first order logic	منطق درجه اول
backtrack	نگاه به مرحله های قبلی
predicate symbol	نماد های گزاره ای
propositional representation	نمایش های گزاره ای

Telegram: @Farabar_BI

www.farabar.net

فصل یازدهم: یادگیری تحلیلی

متد های یادگیری استقرایی مثل شبکه های عصبی و یادگیری درختی برای رسیدن به حدی از دقت در تعمیم نیاز به تعداد قابل توجهی نمونه آموزشی دارند، این نیاز در مرز های تئوری و نتایج آزمایشگاهی تاثیر خواهد گذاشت. یادگیری تحلیلی از دانش قبلی و استدلال استنتاجی برای جمع آوری اطلاعات نمونه های آموزشی استفاده می کند، بنابراین به محدودیت تعداد نمونه های آموزشی وابسته نیست. در این فصل به متد یادگیری تحلیلی ای به نام یادگیری توضیحی[1] (EBL) می پردازیم. در یادگیری توضیحی دانش قبلی برای بررسی و یا توضیح هر یک از نمونه های آموزشی مشاهده شده به کار می رود. سپس از توضیحات داده شده برای تشخیص ویژگی های مرتبط[2] از ویژگی های غیر مرتبط[3] نمونه های آموزشی استفاده می شود، سپس می توان تعمیم را بر اساس روشهای منطقی[4] به جای روشهای آماری پیش برد. یادگیری توضیحی برای یادگیری قوانین کنترل جستجو و کارهای برنامه ریزی و انجام[5] با موفقیت به کار گرفته شده است. در این فصل با فرض اینکه دانش قبلی همه جانبه[6] و درست[7] است به یادگیری توضیحی می پردازیم. و در فصل بعدی به ترکیب یادگیری تحلیلی و استقرایی برای مسائلی که دانش قبلی فقط تا حدی درست است می پردازیم.

۱۱.۱ معرفی

در فصول گذشته انواع مختلف متد های یادگیری استقرایی، متد هایی که تابع را از طریق نمونه های آموزشی و پیدا کردن خواصی که یک نمونه را مثبت یا منفی می کند می آموزند، را بررسی کردیم. متد های یادگیری مثل درخت تصمیم گیری، شبکه های عصبی، برنامه نویسی

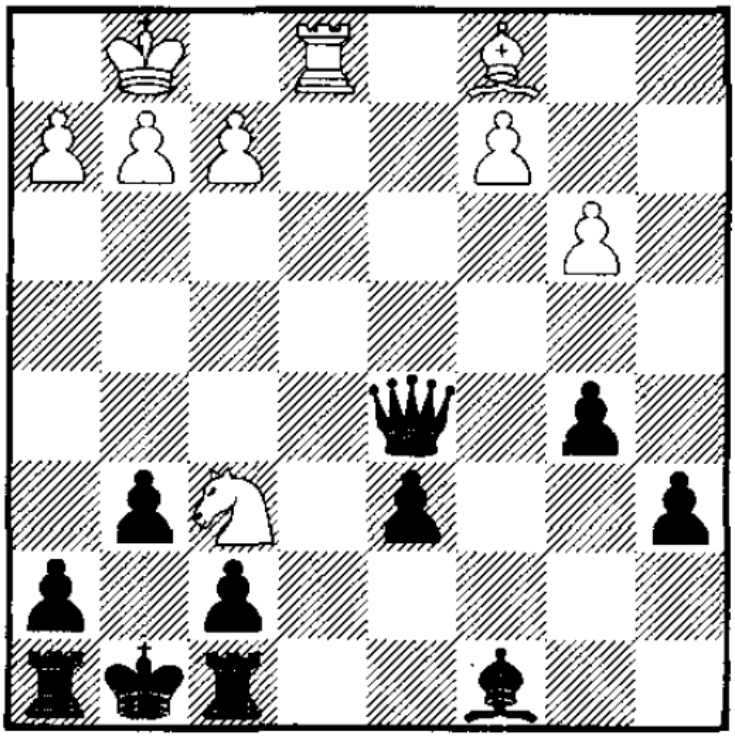
[1] explanation-based learning
[2] relevant
[3] irrevelant
[4] logical
[5] planning and schedualing
[6] complete
[7] correct

www.farabar.net

منطقی استقرایی و الگوریتم های ژنتیک همگی نمونه های متد های استقرایی هستند. ضعف کلیدی متد های استقرایی در نیاز بـه تعـداد قابـل توجهی نمونه های آموزشی است. در واقع همانطور که در فصل ۷ نیز گفته شد، بررسـی هـای تئـوری نشـان مـی دهـد کـه محـدودیت هـای پایه ای ای در دقت این نوع متد ها هنگامی که تعدادی نمونه آموزشی محدود است وجود دارد.

آیا می تواند متد های یادگیری ای ایجاد کرد که این محدودیت های پایه ای را نداشته باشد و با استفاده از داده های آموزشی موجود کـار کنـد؟ جواب مثبت است به شرط اینکه در خود تعریف مسئله ی یادگیری تجدید نظر کنیم. یکی از راههای ممکن ایجاد الگـوریتم هـای یـادگیری ای است که دانشی قبلی را علاوه بر نمونه های آموزشی دریافت کنند. یادگیری توضیحی چنین متدی است. این متد از دانش قبلی برای بررسـی و توضیح هر یک از نمونه های آموزشی استفاده می کند تا بفهمد که کدام ویژگی های نمونه های آموزشی مربوط و کدام ویژگی ها نـامربوط بـه تابع هدفند. چنین اطلاعاتی باعث می شود تا این الگوریتم بتواند بهتر از سیستم های استقرایی که فقط مبتنی بر نمونه هـای آموزشـی هسـتند، نمونه ها را بررسی کند. همانطور که در فصول گذشته نیز دیدیم، سیستم های برنامه نویسی منطقی استقرایی مثل Cigol دانشـی اضـافی نیـز برای جهت دهی به یادگیری دارند. با این وجود، چنین سیستم هایی از دانش قبلی برای تغییر توصیفات نمونه های ورودی استفاده مـی کننـد و پیچیدگی فضای فرضیه ای جستجو را افزایش می دهند. در مقابل، یادگیری توضیحی از این دانش قبلی کـاهش پیچیـدگی فضـای فرضـیه ای مورد جستجو کمک می گیرد، بنابراین، پیچیدگی نمونه ای کاهش یافته و دقت تعمیم یادگیر نیز افزایش خواهد یافت.

برای شهود لازم برای یادگیری توضیحی، یادگیری بازی شطرنج را در نظر بگیرید. در کل، فرض کنید که می خواهیم برنامه ی ما مجموعـه ی پوزیسیون های مهمی را یادبگیرد، مثل مفهوم هدف "پوزیسیون هایی که سیاه در دو حرکت وزیـر خـود را از دسـت خواهـد داد". شـکل ۱۱.۱ پوزیسیونی از این مفهوم هدف را نشان می دهد. البته می توان از یادگیری استقرایی برای یادگیری این تابع هدف استفاده کرد. با این حال، چون صفحه ی تقریبا پیچیده است (۳۲ مهره و ۶۴ خانه) و الگو هایی که به این مفهوم ختم می شوند تقریبا پیچیده اند (به مکـان نسـبی بسـیاری از مهره ها در صفحه بستگی دارد)، باید هزاران نمونه ی آموزشی به یک سیستم استقرایی بدهیم تا بتوانیم انتظار داشته باشیم که آن نمونـه هـای جدید را درست دسته بندی کند.



شکل ۱۱.۱ نمونه ای مثبت از مفهوم هدف "پوزیسیون هایی که سیاه در دو حرکت وزیر خود را از دست خواهد داد".
توجه داشته باشید که اسب همزمان به شاه و وزیر حمله کرده، پس سیاه باید شاه خود را حرکت دهد و در حرکت بعدی سفید وزیر سیاه را خواهد گرفت.

یکی از نکات جالب در مورد یادگیری شطرنج این است که انسانها فقط با دیدن چند مثال چنین توابع هدفی را یاد می گیرند! در واقع، بعد دیدن همین شکل ۱۱.۱ اکثر افراد فرضیه ی مناسب را پیدا می کنند "پوزیسیونهایی که در آن شاه و وزیر هم زمان زیر حمله قرار می گیرند". هیچ کس فرضیه ی دیگری مثل "پوزیسیونهایی که در آن چهار سرباز سفید در خانه ی اولیه ی خود باشند" را پیشنهاد نمی دهد. چگونه آدمی می تواند فقط با دیدن یک مثال تعمیم موفقی انجام دهد؟

جواب در این است که یادگیری آدمی متکی بر تفسیر نمونه ها با دانش قبلی است، در اینجا این دانش قبلی قوانین شطرنج در مورد نحوه ی حرکت مهره هاست. اگر از یادگیر انسانی بپرسید که چرا شکل ۱۱.۱ یک نمونه مثبت از مفهوم "پوزیسیون هایی که سیاه در دو حرکت وزیر خود را از دست خواهد داد" است اکثر افراد جوابی مشابه خواهند داد: "زیرا که اسب سفید همزمان به شاه و وزیر حمله کرده است، سیاه باید از کیش خارج شود، پس در حرکت بعدی اسب وزیر را خواهد گرفت". اهمیت چنین توضیحاتی به این است که اطلاعات لازم برای تعمیم موفق از جزئیات نمونه های آموزشی در پی بردن به تابع هدف را در خود دارند. ویژگی های نمونه ی آموزشی که در توضیح آمد (مثل مکان اسب سفید، شاه سیاه و وزیر سیاه) مرتبط به تابع هدفند و باید در توضیح فرضیه ی تعمیم استفاده شوند. در مقابل، ویژگی هایی که در توضیح نیامده (مثل اینکه که سیاه ۴ سوار دارد) نباید توجه شود.

دانش لازم برای یادگیر ساختن چنین توضیحی برای این مثال شطرنجی چیست؟ فقط کافی است حرکت های قانونی مهره های شطرنج را بدانیم: بدانیم که اسب و بقیه ی مهره ها چگونه در صفحه حرکت می کنند، بدانیم که دو بازیکن نوبت به نوبت مهره های خود را حرکت می دهند، و بدانیم که اگر بازیکنی شاه حریف را بگیرد برنده می شود. توجه داشت باشید که فقط با این دانش قبلی در کل از نظر تئوری می توان حرکت بهینه را برای هر پوزیسیون پیدا کرد. با این وجود، در عمل این محاسبات می تواند بسیار پیچیده باشد، با وجود اینکه آدمی علم

زیادی درباره ی شطرنج دارد اما با این حال هنوز نمی تواند آنرا به طرز بهینه بازی کند. پس اکثر یادگیری انسانی در شطرنج (و دیگر بازی هایی که نیاز به نقشه کشیدن دارد) نیاز به فرایندی بلند برای پیدا کردن نتایج دانش قبلی، و نمونه های آموزشی دارد.

در این فصل به توصیف الگوریتم های یادگیری ای می پردازیم که به طور خودکار چنین توضیحاتی را ساخته و بر اساس آنها مفاهیمی یاد خواهند گرفت. در ادامه ی این قسمت مسئله ی یادگیری تحلیلی را به صورت دقیقتر تعریف خواهیم کرد. در قسمت بعد، به الگوریتم توضیحی خاصی به نام Prolog-EBG خواهیم پرداخت. و در قسمت های بعدی نیز خواص کلی این الگوریتم و رابطه ی آنرا با الگوریتم های یادگیری استقرایی، را که در دیگر فصل ها آورده شده اند، بررسی خواهیم کرد. در آخر نیز به کاربرد یادگیری توضیحی برای بهبود کارایی در مسائلی با فضای حالت بزرگ[1] خواهیم پرداخت. در سراسر این فصل فرض می کنیم که توضیحات از دانش قبلی کاملا درست (مثل دانش انسانها) ناشی می شوند، مثل مثال شطرنج بالا. در فصل ۱۲ به حالتی کلی تر که در آن دانش قبلی تقریبا درست است خواهیم پرداخت.

۱۱.۱.۱ مسائل یادگیری استقرایی و یادگیری تحلیلی

تفاوت های اصلی بین متد های یادگیری استقرایی و متد های تحلیلی این است که آنها دو تصور متفاوت از مسئله دارند:

- در یادگیری استقرایی، به یادگیر فضای فرضیه های H و دسته مثالهای $D = \{< x_1, f(x_1) >, \dots, < x_n, f(x_n) >\}$ داده می شود تا فرضیه ای از H مثل h را که سازگار با نمونه های آموزشی D است پیدا کند.
- در یادگیری تحلیلی، ورودی یادگیر شامل همان فضای فرضیه ای H و مثالهای D است. علاوه بر این ورودی ها، ورودی دیگری نیز به شکل روبرو به یادگیر داده می شود: یک تئوری قلمرو[2] مثل B که دانش قبلی ای به یادگیر می دهد تا بتواند نمونه های آموزشی را تجزیه و تحلیل کند. خروجی یادگیر فرضیه ای مثل h از H خواهد بود که با نمونه های آموزشی D و تئوری قلمرو B سازگار باشد.

برای تصور، در مثال ذکر شده x_i ها پوزیسیون ها هستند و $f(x_i)$ زمانی که پوزیسیون نمونه ی مثبتی از "پوزیسیونهایی که در آن شاه و وزیر هم زمان زیر حمله قرار می گیرند" درست و در غیر این صورت غلت است. فضای فرضیه ای H را می توان دسته گزاره های تعریف شده در فصل ۱۰ (قانون های if-then) که در آن شروط جای نسبی مهره ها در صفحه ی شطرنج است در نظر گرفت. تئوری قلمرو B را نیز می توان قوانین شطرنج در نظر گرفت. این قوانین شامل نحوه ی حرکت مهره ها، تغییر نوبت حرکت، و این حقیقت است که زمانی که کسی شاهش را از دست بدهد می بازد می شود.

توجه داشته باشید که در یادگیری تحلیلی، یادگیر باید فرضیه ای را به عنوان خروجی ارائه کند که هم با نمونه های آموزشی و هم با تئوری قلمرو سازگار باشد. زمانی که می گوییم فرضیه ی h با تئوری قلمرو ی B سازگار است که تئوری قلمروی B فرضیه ی h را رد نکند ($B \nvdash \neg h$). این شرط آخر باعث می شود ابهام H در زمانهایی که یادگیر با نمونه های آموزشی ای مواجه است که H را به اندازه ی کافی محدود نمی کنند کم شود. اثر کلی که توسط تئوری قلمرو ایجاد می شود، دقت درستی فرضیه ی خروجی را افزایش می دهد.

بیایید مسئله ی دیگری از یادگیری تحلیلی را معرفی کنیم، مسئله ای که در ادامه ی فصل برای توضیحات به کار می رود. فضای نمونه های X را در نظر بگیرید که هر نمونه در این فضا جفتی از اشیاء است. هر یک از این دو شی با خواص رنگ، حجم، صاحب، جنس، نوع و چگالی و

---

[1] large state-space search problems
[2] domain theory

رابطه ی بین دو شی نیز با خاصیت On مشخص شده است. با معلوم بودن فضای نمونه ها می خواهیم مفهوم هدف "جفت اجسامی که یکی را می توان با اطمینان به روی دیگری گذاشت" را که با نماد SafeToStack(x,y) یاد بگیریم. یادگیری چنین مفهومی می تواند بسیار سودمند باشد، برای مثال، برای یک ربات که می خواهد اجسام را در فضای محدودی انبار کند این مفهوم بسیار مهم است. تعریف کامل این مسئله در جدول ۱۱.۱ آمده است.

ورودی های:

- فضای نمونه های X: هر نمونه جفتی از اجسام است که با خواص Type، Color، Volume، Owner، Material، Density و On مشخص می شوند.
- فضای فرضیه های H: هر فرضیه دسته ای از قوانین Horn clause است. سر هر Horn clause یک عبارت شامل SafeToStack است. در بدنه ی هر horn clause رابطه ای عطفی از عملگر هایی است که بر پایه ی همان خواص نمونه ها عمل می کنند. این عملگر ها شامل توابع LessThan، Equal، و GreaterThan و توابعی مثل plus، minus و times می شود. برای مثال:

SafeToStack(x, y) ←Volume(x, vx) ∧ Volurne(y, vy) ∧ LessThan(vx, vy)

- مفهوم هدف: SafeToStack(x,y)
- نمونه های آموزشی: نمونه هایی مثل نمونه ی آموزشی زیر SafeToStack(Obj1,Obj2):

On(Obj1,Obj2)	Owner(0bj1 , Fred)
Type(0bj1 , Box)	Owner(Obj2, Louise)
Type(Obj2, Endtable)	Density(0bj1,0.3)
Color(Obj1, Red)	Material(Obj1, Cardboard)
Color(Obj2, Blue)	Material (Obj2, Wood)
Volume(Obj1,2)	

Domain Theory B:

SafeToStack(x, y) ←¬Fragile(y)

SafeToStack(x, y) ←Lighter(x, y)

Lighter(x, y) ← Weight(x, wx)∧ Weight(y, wy)∧ LessThan(wx, wy)

Weight(x, w) ← Volume(x, v) ∧ Density(x,d) ∧ Equal(w, times(v, d))

Weight(x,5) ← Type(x, Endtable)

Telegram: @Farabar_BI
www.farabar.net

Fragile(x) ← Material (x, Glass)

...

- خروجی:

فرضیه ای از H را پیدا کن که هم با نمونه های آموزشی و هم با تئوری قلمرو سازگار باشد.

جدول ۱۱.۱ مثالی از مسئله های یادگیری تحلیلی

همانطور که در جدول ۱۱.۱ نیز نشان داده شده است فضای فرضیه های H تمامی قوانین درجه اول if-then یا همان horn clause در نظر گرفته شده است (در سراسر این فصل ازهمان نمادگذاری و اصطلاحات تعریف شده در جدول ۱۰.۳ استفاده می کنیم). برای مثال، نمونه ی horn clause ی که در جدول ۱۱.۱ آمده است، بیان می کند که جسم x را می توان روی جسم y گذاشت اگر که حجم x کمتر از (LessThan) حجم y باشد (در این نمایش Vx و Vy به ترتیب حجم جسم x و حجم جسم y در نظر گرفته شده اند). توجه داشته باشید که در بیان horn clause ها می توانند از تمامی ویژگی های نمونه ها و تمامی توابع معرفی شده استفاده کرد. یک نمونه مثبت SafeToStack(Obj1,Obj2) نیز در جدول ۱۱.۱ آمده است.

برای فرمولی کردن این یادگیری تحلیلی باید ابتدا تئوری قلمرویی را پیدا کنیم که توضیح دهد که چرا نمونه ی مثبت، نمونه ی مثبتی از مفهوم هدف مذکور است. در مثال پوزیسیون شطرنجی که در اول فصل بیان کردیم، توضیحی از اینکه چرا یک نمونه، نمونه ی مثبت است آورده شده بود. در این مثال نیز تئوری قلمرو باید توضیح دهد که چرا نمونه ی مذکور، نمونه ای مثبت است، به عبارت دیگر باید توضیح دهد که چرا اجسامی با چنین ویژگی هایی را می توان روی هم قرار داد. تئوری قلمروی بیان شده در جدول تعریف هایی همچون "جسم x را می توان روی جسم y قرار داد هر گاه که y شکننده (fragile) نباشد" و "جسم x زمانی شکننده است که جنس (material) آن شیشه (glass) باشد" ارائه شده است. مشابه فرضیه های یادگیر، تئوری قلمرو نیز با استفاده از دسته ای horn clause تعریف شده است تا یادگیر بتواند فرضیه ها را با تئوری قلمرو مقایسه کند. توجه داشته باشید که تئوری قلمرو ویژگی های جدیدی مثل Lighter و Fragile را نیز مورد استفاده قرار داده که جزو ویژگی های نمونه ها نیستند اما آنها را نیز با استفاده از ویژگی های Material، Volume و Density تعریف کرده است. در آخر توجه داشته باشید که تئوری قلمروی نشان داده شده در جدول می تواند اثبات کند که نمونه ی مثبت یک نمونه ی مثبت است (توضیح لازم را ارائه می کند).

۱۱.۲ یادگیری با تئوری قلمرو های کامل: Prolog-EBG

همانطور که پیشتر نیز گفتیم، در این فصل به یادگیری توضیحی ای که بر اساس تئوری قلمرو های کامل اند[1] می پردازیم. یعنی اینکه تئوری قلمروهای مورد استفاده هم درست[2] هم همه جانبه اند زمانی می گوییم یک تئوری قلمرو درست است که تمامی رابطه هایش در جهان واقعی قابل اطمینان باشد و زمانی می گوییم یک تئوری قلمرو همه جانبه است که با توجه به فضای نمونه ها و تابع هدف تمامی نمونه های مثبت را پوشش دهد. به عبارت دیگر، اگر یک تئوری قلمرو تمامی نمونه های که تابع هدف را راضی می کند را توضیح دهد همه جانبه است. توجه داشته باشید که تعریف ما از همه جانبه بودن حرفی در مورد اینکه توضیحی برای نمونه های منفی داشته باشد نمی زند. با این وجود اگر قرار داد

[1] perfect
[2] correct

www.farabar.net

prolog را قبول کنیم، باید نمونه هایی که توضیحی برای مثبت بودن ندارند را منفی دسته بندی کنیم، بنابراین تئوری قلمرو تمامی نمونه ها را پوشش خواهند داد.

ممکن است این سوال را بپرسید که آیا منطقی است که فرض کنیم که چنین تئوری قلمروی کاملی در اختیار یادگیر است؟ به هر حال، با وجود چنین تئوری قلمروی کاملی چرا باید به فکر یادگیری باشیم؟ دو جواب برای چنین سوالی وجود دارد:

- اول اینکه حالاتی وجود دارد که می توان چنین تئوری قلمرو ی کاملی را پیدا کرد. مثال شطرنجی که در ابتدای فصل زدیم یکی از این حالات است که در آن نحوه ی حرکت مهره ها تئوری قلمرو کامل را تشکیل می دهد که با استفاده از آن می توان بازی کردن بهینه را آموخت. علاوه بر آن، با وجود اینکه می توان به راحتی نحوه ی حرکت مهره ها را روی کاغذ نوشت، تئوری قلمروی کامل، اما نمی توان به سادگی نحوه ی بهینه شطرنج بازی کردن را روی کاغذ نوشت. در چنین حالاتی، ما ترجیح می دهیم که تئوری قلمرو را برای یادگیر پیدا کنیم و فرموله کردن مفهوم هدف را به یادگیر واگذار کنیم تا توضیحی قابل استفاده (مثل "پوزیسیونهایی که در حرکت بعدی وزیرم را از دست خواهم داد") از تابع هدف بدهد و با بررسی و تعمیم از این نمونه های آموزشی تخمین خوبی از تابع هدف بدهد. در قسمت ۱۱.۴ کاربرد های موفقیت آمیز یادگیری توضیحی با تئوری قلمرو های کامل را در افزایش کارایی در مسائل متمرکز بر جستجوی برنامه ریزی و بهینه سازی آورده ایم.
- دوم اینکه، در بسیاری از حالات فرض اینکه یک تئوری قلمروی کامل وجود دارد بی دلیل است. حتی نوشتن یک تئوری درست و همه جانبه در مورد مسئله ی ساده ی SafeToStack سخت است. فرض واقع بینانه این است که فرض کنیم توضیحاتی قابل قبول[1] از تئوری قلمرو ی غیر کامل داریم نه اینکه خود تئوری قلمرو را در دسترس داشته باشیم. با این وجود، با درک ایده ی تئوری قلمرو های کامل نقش توضیحات در یادگیری را خواهیم فهمید. در فصل ۱۲ یادگیری با تئوری قلمرو های غیر کامل را بررسی خواهیم کرد.

در این قسمت الگوریتم Prolog-EBG (Kedar-Cabelli and McCarty 1987) را ارائه خواهیم داد که پایه ی بسیاری از الگوریتم های یادگیری توضیحی است. Prolog-EBG الگوریتمی ترتیبی است (رجوع کنید به فصل ۱۰). به عبارت دیگر این الگوریتم با یک قانون horn clause شروع می کند و سپس نمونه های پوشانده شده ی قانون را حذف کرده و دوباره این فرایند را برای نمونه های مثبت باقیمانده ادامه می دهد تا تک تک نمونه های مثبت پوشش داده شوند. زمانی که تئوری قلمرو درست و همه جانبه باشد، Prolog-EBG تظمین می کند که فرضیه ای (دسته قوانین) را خروجی دهد که هم درست باشند و هم نمونه های مثبت آموزشی را پوشش دهد. برای هر دسته از نمونه های آموزشی فرضیه ی خروجی Prolog-EBG دسته ای از عبارات منطقی کافی را بر اساس تئوری قلمرو ایجاد می کند. Prolog-EBG تجدید نظری در EBG است که توسط (Mitchell 1986) معرفی شده و شباهت زیادی به الگوریتم EGGS (DeJong and Mooney 1986) دارد. الگوریتم Prolog-EBG در جدول ۱۱.۲ آورده شده است.

۱۱.۲.۱ یک مثال

برای تصور دوباره نمونه های آموزشی و تئوری قلمروی جدول ۱۱.۱ را در نظر بگیرید، الگوریتم Prolog-EBG الگوریتمی ترتیبی است که مرحله به مرحله تعداد بیشتری از نمونه های آموزشی را پوشش می دهد. برای هر نمونه ی مثبت آموزشی که هنوز توسط horn clause ها پوشش نداده شده است، horn clause جدیدی تشکیل می شود: (۱) این horn clause باید توضیحی برای نمونه ی جدید داشته باشد،

[1] plausible explanations

(۲) این توضیح باید بررسی شود تا تعمیم لازم انجام شود و (۳) horn clause جدید به فرضیه ی فعلی اضافه می شود تا فرضیه نمونه ی آموزشی جدید را علاوه بر نمونه های قبلی پوشش دهد. در زیر هر کدام از این مراحل را بررسی خواهیم کرد.

۱۱.۲.۱.۱ توضیح دادن نمونه های آموزشی

مرحله ی اول هر یادگیری نمونه ی آموزشی پیدا کردن توضیحی برای آن با استفاده از تئوری قلمرو است که توضیح دهد که چرا این نمونه ی مثبت مفهوم هدف را راضی می کند. زمانی تئوری قلمرو درست و همه جانبه است پس باید اثباتی برای اینکه چرا نمونه مفهوم هدف را راضی می کند داشته باشد. اما زمانی که تئوری قلمرو کامل نیست، نمایش توضیحات باید طوری تغییر کنند که قابل قبول باشد و مقادیر را نیز تخمین بزند (فقط به جای اثبات محض نباشد).

Prolog-EBG(TargetConcept, TrainingExamples, DomainTheory)

- {} → LearnedRules
- Pos → نمونه های مثبت از مجموعه ی TrainingExamples
- برای هر نمونه ی مثبت از Pos که توسط LearnedRules پوشانده نمی شوند:

۱. توضیح:

- Explanation → توضیحی را که چرا TargetConcept PossitiveExample را راضی می کند.

۲. بررسی:

- SufficientConditions → کلی ترین دسته ویژگی های PossitiveExample که برای راضی کردن TargetConcept بر اساس Explanation لازم است.

۳. بازنگری:

- NewHornClause + LearnedRules → LearnedRules ، در این رابطه NewHornClause به صورت زیر است:

"SufficientConditions → TargenConcept"

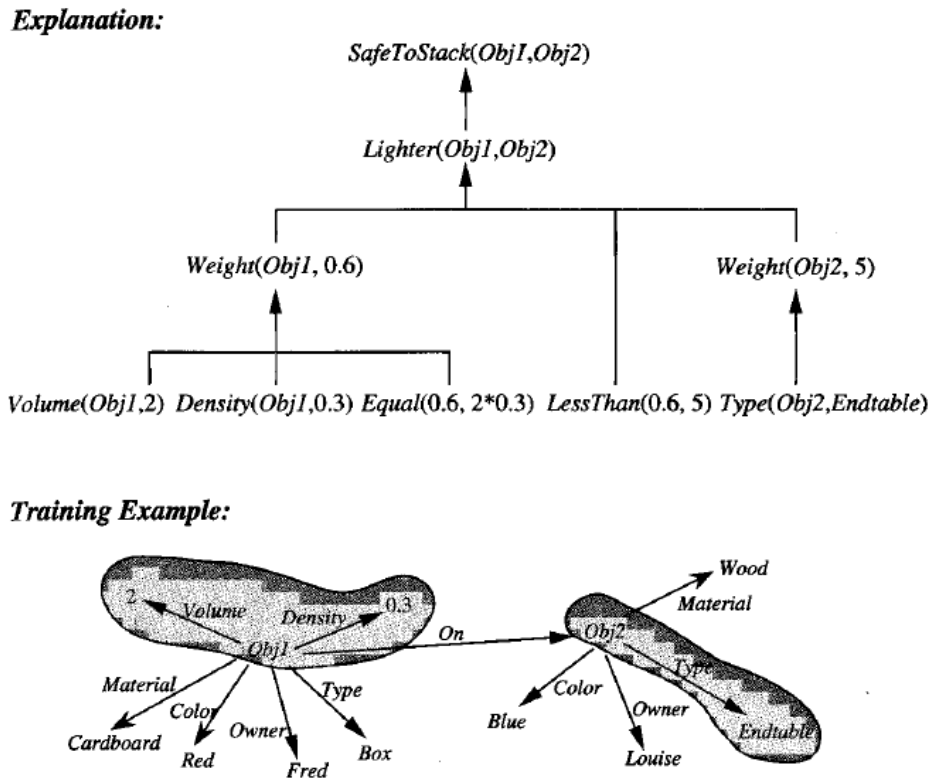
- مجموعه ی LearnedRules را خروجی بده.

جدول ۱۱.۲ الگوریتم Prolog-EBG.

در این الگوریتم برای هر نمونه ی مثبت که هنوز با دسته قوانین یادگرفته شده (LearnedRules) سازگار نیستند، یک horn clause جدید ایجاد می شود. این horn clause جدید در سه مرحله ایجاد می شود. (۱) توضیح نمونه با استفاده از تئوری قلمرو (۲) بررسی توضیحات تا معلوم گردد کدام ویژگی های نمونه در مفهوم هدف دخیل اند و (۳) ساخت یک horn clause جدید که در اینگونه نمونه ها با تابع هدف سازگار باشد.

توضیحات مربوط به نمونه ی ارائه شده در شکل ۱۱.۲ آمده است، توجه داشته باشید که شکل نشان داده شده در وسط نمود تصویری همان نمونه ای است که در جدول ۱۱.۱ آمده بود. بالای شکل توضیحات ساخته شده برای این نمونه را نشان می دهد. توجه داشته باشید که این توضیحات (یا همان اثبات مثبت بودن نمونه) برای گذاشتن Obj1 بر روی Obj2 آمده زیرا که Obj1 سبکتر از Obj2 است. این سبکتری Obj1 از Obj2 از وزن Obj1، که از چگالی (Density) و حجم (Volume) آن بدست آمده و وزن Obj2 که از وزن پیشفرض Endtable بدست آمده، ناشی شده است. Horn clause ی که از این توضیحات بدست می آید در جدول ۱۱.۱ آمده است. توجه داشته باشید که توضیح فقط از تعدادی از ویژگی های Obj1 و Obj2 استفاده کرده (ویژگی هایی که در شکل هاشور زده شده اند.)

با وجود اینکه در مثال ما فقط یک توضیح برای نمونه ممکن بود اما در کل ممکن است که از یک تئوری قلمرو چندین توضیح برای یک نمونه وجود داشته باشد. در چنین شرایطی چند تا یا کل توضیحات مورد استفاده قرار می گیرند. با وجود اینکه این توضیحات ممکن است horn clause را به تعمیم های مختلفی بکشانند اما با این حال همه در جهت تئوری قلمرو اند. در Prolog-EBG، مثل Prolog توضیحات با یک جستجوی زنجیر وار معکوس[1] انجام می شود. Prolog-EBG نیز مثل Prolog در اولین توضیح پیدا شده متوقف می شود و جستجو را ادامه نمی دهد.



شکل ۱۱.۲ توضیح یک نمونه ی آموزشی.

شبکه ی پایین شکل توضیح نمونه ی آموزشی جدول ۱۱.۱ را برای مفهوم SafeToStack(Obj1,Obj2) ارائه می دهد. قسمت بالا یی شکل چگونگی راضی شدن مفهوم SafeToStack توسط این نمونه ی آموزشی را نشان می دهد. قسمت های هاشور خورده در شکل ویژگی های استفاده شده در توضیح را مشخص می کنند. ویژگی های دیگر تاثیری بر تعمیم فرضیه در مرحله ی بررسی ندارند.

۱۱.۲.۱.۲ بررسی توضیحات

نکته ی کلیدی در تعمیم نمونه های آموزشی جواب سوال "از تمامی ویژگی های نمونه های مثبت کدام ویژگی ها ویژگی های تاثیر گذارند؟" است. توضیح ساخته شده ی یادگیر جواب مناسبی به این سوال می دهد: دقیقا ویژگی هایی که در توضیح مورد استفاده قرار گرفته اند. برای مثال توضیحی که در شکل ۱۱.۲ آمده از چگالی (density) Obj1 استفاده کرده در حالی که از صاحب (owner) آن استفاده ای نکرده. بنابراین فرضیه ای که برای SafeToStack(x,y) ارائه می شود باید از ویژگی Density(x,0.3) استفاده کند و کاری با ویژگی

[1] backward chaining search

Owner(x,Fred) نداشته باشد. با جمع آوری اطلاعات مشخص شده در برگ های درخت شکل ۱۱.۲ و جایگزین کردن x و y به جای Obj1 و Obj2 می توان قانون کلی زیر را از تئوری قلمرو بدست آورد:

SafeToStack(x, y) ←Volume(x, ۲) ∧ density(y, 0.3) ∧ Type(y,EndTable)

قانون بالا تمامی موارد ذکر شده در برگ های شکل ۱۱.۲ را جز دو عبارت "Equal(0.6,times(2,0.3))" و "LessThan(0.6,5)" را در خود دارد. این دو برگ حذف شده اند زیرا که جدا از ویژگی های دو جسم x وy همیشه مقدار درست دارند.

با داشتن این قانون، برنامه می تواند توجیه[1] متناسب را هم پیدا کند: توضیح نمونه ی آموزشی اثباتی برای درستی این قانون است. با وجود اینکه این توضیح برای پوشش نمونه ی آموزشی ساخته شده بود، اما توضیحی برای تمامی نمونه هایی که در آن صدق کنند نیز دارد.

قانون بالا تعمیمی قابل توجه روی نمونه ی آموزشی داده است، زیرا که بسیاری از ویژگی های دیگر نمونه را که مهم نبودند حذف کرده (ویژگی های مثل رنگ (color) دو شی). با این وجود با بررسی دقیقتر توضیحات، قوانین کلی تری را می توان بدست آورد. الگوریتم Prolog-EBG کلی ترین قانون را که توضیحات را توجیه کند بدست می آورد، این کار با استفاده از پیدا کردن ضعیفترین پیشنویس[2] توضیحات انجام می شود:

تعریف: ضعیفترین پیشنویس یک نتیجه گیری مثل C با توجه به اثبات P، کلی ترین دسته فرضهای اولیه ای مثل A است که بر اساس A با فرض درست بودن P بتوان C را نتیجه گرفت.

برای مثال مذکور ضعیفترین پیشنویس مفهوم هدف SafeToStack(x,y)، با توجه به نمونه ی آموزشی جدول ۱۱.۱ به شکل قانون زیر بیان می شود. این کلی ترین قانونی است که می توان با توضیح ۱۱.۲ نوشت.

SafeToStack(x,y)←Volume(x,vx)∧ Density(x,dx)∧ Equal(wx,times(vx,dx))∧

LessThan(wx,5)∧ Type(y,EndTable)

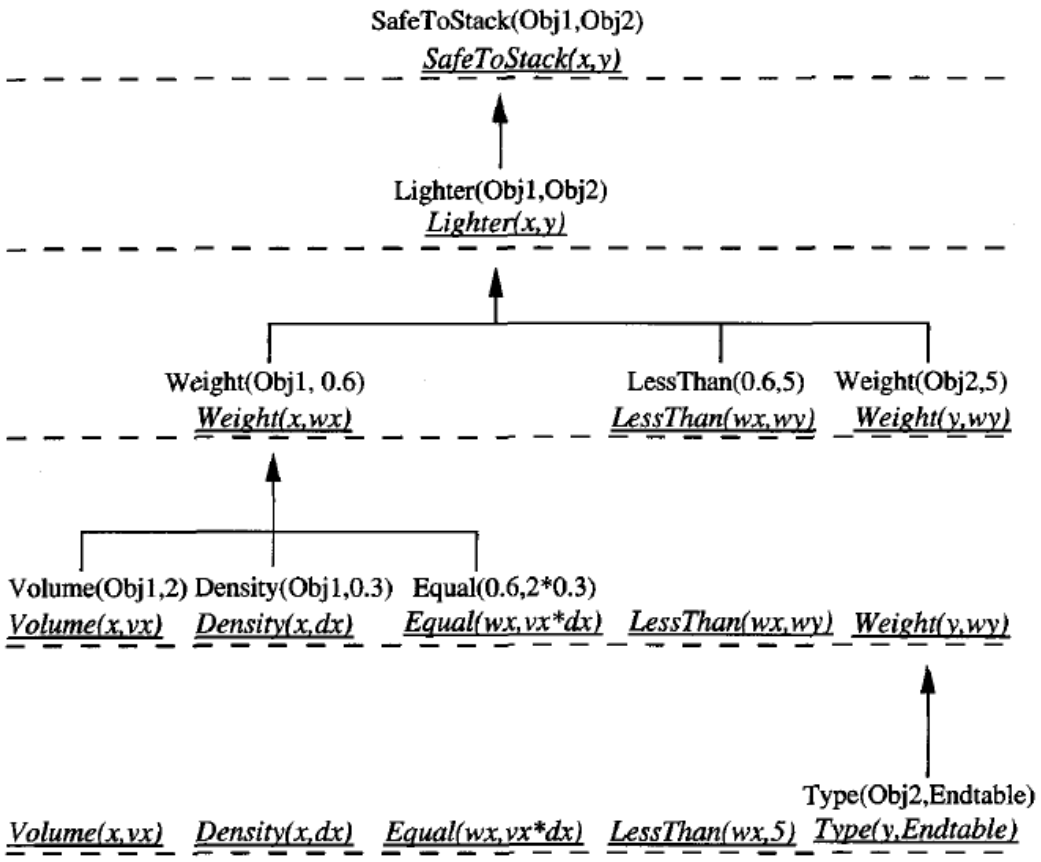
توجه دارید که این قانون کلی دو مقدار حجم (volume) و (density) که در قانون اول آمده بود را نیاز ندارد. و به جای آن شرایطی کلی تر برای مقادیر این ویژگی ها می گذارد.

الگوریتم Prolog-EBG ضعیفترین پیشفرض تابع مفهوم را بر اساس توضیحات و فرایندی به نام regression (Waldinger 1977) محاسبه می کند. فرایند regression بر روی تئوری قلمروهایی که بر اساس horn clause ها تعریف شده اند عمل می کند. این فرایند پله به پله توضیحات را بر عکس مورد بررسی قرار می دهد، ابتدا ضعیفترین پیشفرض مفهوم هدف را بر اساس مرحله ی نهایی اثبات توضیحات محاسبه می کند، سپس ضعیفترین پیشفرض عبارات حاصل را با توجه به قدم قبلی محاسبه می کند، و به همین ترتیب ادامه می دهد. این فرایند زمانی پایان می یابد که تمامی قسمت های توضیح بررسی شده باشند، یعنی ضعیفترین پیشفرض مفهوم هدف با توجه به تمامی برگ های توضیح بدست آمده باشد.

---

[1] justification
[2] weakest preimage

مراحل طی شده ی فرایند regression در شکل ۱۱.۳ آمده است، در این شکل قسمت توضیح شکل ۱۱.۲ با خط غیرکج[1] دوباره آورده شده است. مرز تعیین شده در هر مرحله ی فرایند regression با خط کج و زیرخط دار[2] نوشته شده است. فرایند از ریشه ی درخت شروع به کار می کند و مرزی را برای مفهوم هدف کلی SafeToStack(x,y) بیان می کند. در مرحله ی اول، ضعیفترین پیشفرض ممکن با توجه به عبارات و قسمت انتهایی درخت محاسبه می شود. در این مرحله قانون بدست آمده SafeToStack(x,y) ← Lighter(x,y) خواهد بود، پس ضعیفترین پیشفرض ممکن Lighter(x,y) است. حال فرایند با مفهوم جدید {Lighter(x,y)} ادامه پیدا می کند. در مرحله ی بعدی و بررسی horn clause بعدی توضیح با فرایند به ضعیفترین پیشنویس {Weight(x,wx),LessThan(wx,wy),Weight(y,wy)} می رسیم، یعنی تمامی x و y هایی که در آنها wx (وزن x) از wy (وزن y) کمتر است. این فرایند به همین صورت پله به پله ادامه پیدا می کند تا بلاخره به دسته قوانینی که در برگهای درخت نوشته شده اند می رسد. مجموعه ی این قوانین که در انتهای شکل ۱۱.۳ نیز آمده قانون معرف ضعیفترین پیشفرض خواهد بود.



شکل ۱۱.۳ محاسبه ی ضعیفترین پیشفرض برای مفهوم SafeToStack(Obj1,Obj2) با توجه به توضیح موجود.
مفهوم هدف از ریشه (نتیجه) تا برگهای درخت بررسی می شود. در هر مرحله (که با خطچین جدا شده) دسته قوانین مرزی (با خط کج و زیرخط دار) یک مرحله در توضیحات عقب می رود. زمانی که فرایند پایان می پذیرد، عطف شرایط بدست آمده ضعیفترین پیشفرض موجود برای توضیح و مفهوم هدف موجود خواهد بود. ضعیفترین پیشفرض بدست آمده در انتها با خط کج و زیرخط دار نشان داده شده است.

[1] nonitalic
[2] italic underlined

قلب فرایند regression الگوریتم دیگری به نام regress است که در هر مرحله مرز ها[1] فعلی را با استفاده از Horn clause ی از تئوری قلمرو پس می راند[2]. این الگوریتم در جدول ۱۱.۳ آورده شده است. مثال آمده در این جدول مرحله ی اول فرایند regression را در شکل ۱۱.۳ نشان می دهد. همانطور که در جدول هم آورده شده است، الگوریتم regress با پیدا کردن جانشین ای برای سر horn clause که با عبارت مرز یکی است و جایگزینی مرز با بدنه ی قانون، آنرا پیش می برد.

قانون خروجی الگوریتم Prolog-EBG با فرمول زیر بیان می شود: بدنه ی قانون ضعیف ترین پیشفرض پیدا شده از طریق روش بالا خواهد بود و سر قانون نیز همان مفهوم هدف است که برای هر هر مرحله ی فرایند regression جانشینی ای (مثلا جانشینی Θ_{hl} در جدول ۱۱.۳) به آن اعمال شده است. این جانشینی از این جهت ضروری است که ثبات اسامی متغیر ها در سر و بدنه ی حکم ایجاد شده را حفظ کرده و سر قانون را در مواقعی که توضیحات استفاده شده فقط یک حالت خاص از تابع هدف اند مشخص کند. همانطور که پیشتر نیز اشاره شد، برای مثال حاضر قانون نهایی به شکل زیر خواهد بود،

SafeToStack(x,y)← Volume(x,vx)∧Density(x,dx)∧

Equal(wx,times(vx,dx))∧LessThan(wx,5)∧Type(y,Endtable)

Regress(Frontier,Rule,Literal, Θ_{hi})

Frontier: دسته ای از قوانین که باید با Rule، پسروی کنند (regress)

Rule: یک horn clause

Literal: عبارتی در Frontier که توسط Rule در توضیحات استنتاج می شود

Θ_{hi}: جانشینی ای که سر قانون Rule را با توجه به عبارت توضیحات یکتا می کند

این فرایند مجموعه ای از عبارات را که با هم ضعیفترین پیشفرض Frontier را با توجه به Rule تشکیل می دهند را بر می گرداند

head→ سر Rule

body→ بدنه ی Rule

Θ_{hi}→ کلی ترین یکتا کننده ی head با Literal که جانشینی ای چون Θ_{hl} وجود داشته باشد که

$$\Theta_{hl}\left(\Theta_{hi}(head)\right) = \Theta_{hi}(head)$$

$\Theta_{hi}(Frontier - head + body)$ را برگردان

مثال (اولین مرحله ی regression در شکل ۱۱.۳)

Frontier = {Volume(x, us), Density(x, dx), Equal(wx, times(vx,dx)), LessThan(wx, wy),Weight(y,

---

[1] Frontier
[2] regress

wy))

Rule = Weight(z,5) ← Type(z, Endtable)

Literal = Weight(y, wy)

Θ_{hi} = {z/Obj2}

head ← Weight (z, 5)

body ← Type(z, Endtable)

Θ_{hl} ← {z/y, wy/5}, where Θ_{li}= (y/Obj2)

Return {Volume(x, us), Density(x, dx), Equal (wx, times(vx, dx)), LessThan(wx, 5),

Type(y, Endtable)}

جدول ۱۱.۳ الگوریتم regression برای مجموعه ای از عبارات با یک Horn clause.
مجموعه ی عبارات Frontier با Rule پس رانده می شوند. Literal عضوی از Frontier است که توسط Rule در توضیحات توجیه شده است. جانشینی $\boldsymbol{\Theta_{hi}}$ جانشینی از متغیر هاست که سر Rule را به عبارت مربوطه ی آمده در توضیحات مربوط می کند. الگوریتم ابتدا جانشینی $\boldsymbol{\Theta_{hi}}$ را که سر Rule و Literal را به صورتی که با جانشینی $\boldsymbol{\Theta_{hl}}$ سازگار باشد یکتا می کند محاسبه می کند. سپس جانشینی $\boldsymbol{\Theta_{hi}}$ را با توجه به Rule برای ایجاد پیشفرض Frontier اعمال می کند. علامت "+" و "-" در الگوریتم نشان دهنده ی اجتماع و اختلاف مجموعه ها هستند. نماد گذاری {z/y} نیز نشان دهنده ی جایگزینی y به جای z است. یک نمونه از چگونگی عملکرد الگوریتم آورده شده.

۱۱.۲.۱.۳ تجدید نظر در فرضیه های موجود

فرضیه ی فعلی در هر مرحله مجموعه ای از horn clause هاست که تا آن مرحله یاد گرفته شده اند. در هر مرحله، الگوریتم ترتیبی نمونه ی مثبت جدیدی را که هنوز پوشانده نشده انتخاب کرده و آنرا توضیح می دهد و قانونی جدید بر اساس فرایندی که در بالا توضیح داده شد ایجاد می کند. توجه داشته باشید که طبق تعریفی که کردیم در این الگوریتم فقط نمونه های مثبت پوشانده خواهند شد و مجموعه ی horn clause های پوشانده شده فقط نمونه های مثبت را پیش بینی می کنند. اگر یک نمونه توسط قوانین فعلی یادگرفته شده پیشبینی نشود آن نمونه منفی دسته بندی خواهد شد. این مطابق با روش منفی در زمان شکست (negation-as-failure) است که در سیستم Prolog توضیح داده شد.

۱۱.۳ نکاتی در مورد یادگیری توضیحی

همانطور که در بالا دیدیم، Prolog-EBG بررسی ای دقیق از تک نمونه های آموزشی برای تعیین بهترین راه تعمیم آن به یک فرضیه horn clause انجام می دهد. در زیر خواص کلیدی این الگوریتم آورده شده است،

بر خلاف روشهای استقرایی، Prolog-EBG فرضیه های کلی توجیه شده توسط دانش قبلی را برای بررسی تک نمونه ها ارائه می دهد.

توضیح اینکه چگونه یک نمونه مفهوم هدف را راضی می کند مشخص می کند که کدام ویژگی های نمونه مربوط و کدام ویژگی ها نامربوط به تابع هدفند: ویژگی های استفاده شده در توضیح ویژگی های مربوط هستند.

با بررسی بیشتر توضیح، پس راندن مفهوم هدف برای تعیین ضعیف ترین پیشفرض با توجه به توضیحات به ما اجازه می دهد تا قید های (constraint) کلی دیگری درباره ی ویژگی های مربوط پیدا کنیم.

هر horn clause یادگرفته شده متناسب با شرط کافی برای راضی کردن تابع هدف است. مجموعه ی horn clause های یادگرفته شده نمونه های آموزشی مثبتی را می پوشانند که یادگیر در طول یادگیری با آنها مواجه شده است، اما دیگر نمونه هایی که این شرایط کافی را داشته باشند نیز پوشانده خواهند شد.

میزان تعمیم horn clause های یادگرفته شده به فرمول تئوری قلمرو و ترتیب مشاهده ی نمونه های آموزشی وابسته است.

Prolog-EBG کاملا فرض می کند که تئوری قلمرو درست و کامل است. اگر تئوری قلمرو درست یا کامل نباشد، مفهوم یادگرفته شده نیز نا درست خواهد بود.

جنبه های مربوطه ی بسیاری در یادگیری توضیحی وجود دارد که به درک قابلیت ها و محدودیت های آن کمک می کند:

EBL به عنوان تعمیم نمونه ها با هدایت تئوری. EBL از تئوری قلمرویش برای تعمیم نسبی نمونه ها با مشخص کردن ویژگی های مربوط و نامربوط نمونه ها استفاده می کند. با این روش این الگوریتم از مرز های پیچیدگی نمونه ای که کاملا در یادگیری استقرایی ایجاد می شود اجتناب می کند. این جنبه ای است که به طور ضمنی در توضیحات بالا در باره ی الگوریتم Prolog-EBG آمده است.

EBL به عنوان بازنویسی تئوری ها بر اساس نمونه ها. به الگوریتم Prolog-EBG می توان به صورت متدی برای بازنویسی تئوری قلمرو به فرمی کاربردی تر نگاه کرد. در کل، تئوری قلمرو اصلی با ایجاد قوانینی که (a) که از تئوری قلمرو نتیجه گیری خواهد شد و (b) نمونه های آموزشی مشاهده شده را تنها در یک مرحله استنباطی دسته بندی می کند، بازنویسی می شود. بنابراین، قوانین یادگرفته شده را می توان بازنویسی تئوری قلمرو به صورت دسته ای از قوانین حالت خاص که می توانند نمونه های مفهوم هدف با یک استنتاج دسته بندی کنند دانست.

EBL به "فقط" عنوان بازنویس آنچه یادگیر "می داند". از نظری، یادگیر مثال SafeToStack با دانشی کامل از مفهوم SafeToStack شروع می کند. بدین معنا که، اگر تئوری قلمروی اولیه برای توضیح تمامی نمونه های آموزشی کافی است، پس برای پیشبینی دسته بندی ها نیز کافی خواهد بود. بنابراین، پس از چه نظر، این یادگیر ویژگی یادگیری دارد؟ یکی از جواب ها این است که تفاوت بین اینکه چیزی را به طور کلی بدانیم و اینکه چیزی را به طور بهینه محاسبه کنیم ممکن است بسیار زیاد باشد، و در چنین شرایطی این نوع "بازنویسی دانش" می تواند فرم مهمی از یادگیری باشد. برای مثال، در بازی شطرنج، قوانین بازی تئوری قلمرویی کامل هستند، که در کل برای بازی بهینه ی شطرنج کافی اند. با این وجود، افراد برای یادگیری خوب بازی کردن نیاز به تجربه دارند. این دقیقا وضعیتی است که تئوری قلمرویی همه جانبه و کامل برای یادگیر (بازیکن) معلوم است و یادگیری فقط بازنویسی این دانش به فرمی است که بتوان آنرا به طور کارامد برای تعیین حرکت مناسب به کار برد. یک مسیر در حال ایجاد در فیزیک نیوتونی همین خاصیت را نشان می دهد، قوانین ساده ی فیزیک به راحتی نوشته می شوند اما دانشجویان قسمت عمده ای از ترم را بر روی نتایج آن کار می کنند در حالی که آنها این دانش را در فرم کاربردی تر دارند و نیازی به بدست آوردن راه حل مسئله از قوانین اولیه در امتحان نهایی نخواهند داشت. Prolog-EBG این بازنویسی دانش را با نگاشت مستقیم دسته قوانین

یادگرفته اش روی ویژگی های نمونه های مشاهده شده و دسته بندی متناسب با مفهوم هدف به طوری که سازگار با تئوری قلمروی مربوطه باشد انجام می دهد---. از آنجایی که دسته بندی نمونه ی دلخواه به مراحل متعددی استنتاج و جستجوی قابل توجهی در تئوری قلمرو دارد، قوانین یادگرفته شده نمونه های مشاهده شده را در یک مرحله استنتاج دسته بندی می کنند.

بنابراین، فرم خالص EBL، بازنویسی تئوری قلمرو به صورت دسته قوانینی که نمونه های آموزشی را در یک استنباط دسته بندی می کنند است. این نوع بازنویسی دانش گاهی گرداوری دانش[1] نامیده می شود، بدین معنا که این تبدیل کارایی دانش را Same درستی دانش افزایش می دهد.

۱۱.۳.۱ پیدا کردن خواص جدید

یکی از قابلیت های جالب Prolog-EBG توانایی آن در فرمولی کردن خواص جدیدی که به صورت صریح در توضیح نمونه های آموزشی نیآمده اند اما برای توصیف قانون کلی نمونه های آموزشی لازمند است. این قابلیت در بررسی عملکرد الگوریتم و قانون یادگفته شده ی قسمت قبل نشان داده شده است. در اینجا، قانون یادگرفته شده نشان می دهد که قید روی Volume و Density ی x این است که ضربشان باید کمتر از ۵ باشد. در واقع، نمونه های آموزشی توصیفی از چنین حاصلضربی یا مقداری که باید داشته باشد ندارند، در مقابل این قید توسط یادگیر به صورت اتوماتیک ایجاد می شود.

توجه دارید که این "ویژگی"[2] مشابه ویژگی های واحد های لایه پنهان شبکه های عصبی است؛ از این جهت که، این ویژگی یکی از خواص بسیار زیاد قابل محاسبه روی ویژگی های نمونه هاست. مشابه الگوریتم backpropation، Prolog-EBG نیز به طور خودکار چنین ویژگی هایی را حین تلاش برای تناسب با داده های آموزشی پیدا می کند. با این وجود، برخلاف فرایند آماری که ویژگی های واحدهای پنهان شبکه های عصبی را پیدا می کرد، Prolog-EBG از فرایندی تحلیلی برای ایجاد ویژگی جدید بر اساس تحلیل نمونه ی آموزشی استفاده می کند. در بالا، Prolog-EBG فرم تحلیلی ویژگی Density·Volume>5 را از نمونه سازی خاص تئوری قلمرو برای توضیح یک تک نمونه ی آموزشی ایجاد کرد. برای مثال، نماد گذاری حاصلضرب Density و Volume از آن جهت اهمیت دارد که از یکی از قوانین تئوری قلمرو که Weight را تعریف می کند ایجاد می شود. این تفکر که این حاصلضرب باید کمتر از ۵ باشد از دو قانون تئوری قلمرو که ادعا می کنند Obj1 باید Lighter و Endtable باشد و اینکه وزن Endtable (Weight)، ۵ است ناشی می شود. بنابراین، این ترکیب خاص و نمونه برداری و عبارات اولیه ی تئوری قلمرو است که به تعریف این خواص جدید کمک می کند.

مبحث یادگیری خودکار خواص مفید برای تغییر نمایش روی نمونه ها مبحث مهمی در یادگیری ماشین است. مشتقل تحلیلی این خواص جدید در یادگیری توضیحی و مشتقل استقرایی این خواص در لایه ی پنهان شبکه های عصبی دو روش مجزا را ارائه می کنند. زیرا که آنها به منابع مختلف اطلاعات (نظم های آماری روی نمونه های زیاد در مقابل تحلیل یک نمونه با استفاده از تئوری قلمرو) بررسی روشهایی که از ترکیب این دو منبع استفاده می کنند مفید خواهد بود.

---

[1] knowledge compilation
[2] feature

۱۱.۳.۲ یادگیری استنتاجی

در فرم خالص، Prolog-EBG بیشتر شبیه روشی استنتاجی است تا روشی استقرایی. بدین معنا کـه بـا محاسـبه ی ضـعیف تـرین پیشـفرض توضیحات فرضیه مثل h ایجاد می شود که از تئوری قلمروی B که نمونه های آموزشی D را پوشش می دهد نتیجـه گیـری خواهـد شـد. بـه عبارت دقیقتر، Prolog-EBG فرضیه ای مثل h را خروجی خواهد داد که در دو شرط زیر صدق می کند:

$$(\forall < x_i, f(x_i) > \in D)(h \wedge x_i) \vdash f(x_i) \qquad (11.1)$$

$$D \wedge B \vdash h \qquad (11.2)$$

در این روابط داده های آموزشی D شامل مجموعه ای از نمونه های آموزشی است که در آن x_i، i امـین نمونـه ی آموزشـی و $f(x_i)$ مقـدار هدف آن است (f تابع هدف است). توجه داشته باشید که شرط اول فقط تعبیر ریاضی شرط کلی یادگیری ماشین است، اینکه فرضیه ی h بایـد مقدار هدف $f(x_i)$ را برای هر نمونه ی x_i در D به درستی پیشبینی کند (البته در اینجا ---). البته در کل، فرضیه هـای جـایگزین بسـیاری وجود دارند که در شرط اول صدق می کنند. شرط دوم تاثیر تئوری قلمرو در Prolog-EBG را نشان می دهد: فرضیه ی خروجی مجبور اسـت از تئوری قلمرو و داده های آموزشی نتیجه گرفته شده باشد. این شرط دوم ابهام یادگیر در انتخاب فرضیه را کـاهش مـی دهـد. بنـابراین، تـاثیر تئوری قلمرو کاهش موثر اندازه ی فضای فرضیه و بنابراین کاهش پیچیدگی نمونه ها در یادگیری است.

با نمایشی دیگر می توان نوع دانشی که Prolog-EBG به عنوان تئوری قلمرو نیاز دارد را معلوم کرد. در کل، Prolog-EBG فرض می کند که تئوری قلمروی B را می توان از دسته بندی نمونه های داده های آموزشی نتیجه گرفت:

$$(\forall < x_i, f(x_i) > \in D)(B \wedge x_i) \vdash f(x_i) \qquad (11.3)$$

این شرط برای تئوری قلمرو ی B معادل این است که فرض کنیم که B بتواند برای هر نمونه مثبت توضیحی ارائه دهد.

مقایسه ی تعریف مسئله ی یادگیری Prolog-EBG و برنامه نویسی استقرایی منطقی (ILP) که در فصل ۱۰ بحث شد جالب است. در فصـل ۱۰ فرض کردیم که تعمیمی از یادگیری استقرا معمول انجام می دهیم با این تفاوت که دانش قبلی ای مثل B′ در اختیار یادگیر است. (از B′ به جای B برای دانش قبلی ILP استفاده کردیم تا نشان دهیم که B′ در رابطه ی ۱۱.۳ صدق نمی کند). ILP یـک سیسـتم اسـتقرایی اسـت امـا Prolog-EBG یک سیستم استنتاجی است. ILP از دانش قبلی B′ برای افزایش اندازه ی مجموعه ی فرضیه های ممکن استفاده می کند در حالی که Prolog-EBG از دانش قبلی B برای کاهش مجموعه ی فرضیه های قابل قبول استفاده می کند. همانطور کـه در رابطـه ی 10.2 نیز آمد، ILP فرضیه ای مثل h خروجی می دهد که که در شرط زیر صدق می کند،

$$(\forall < x_i, f(x_i) > \in D)(B' \wedge h \wedge x_i) \vdash f(x_i)$$

به رابطه ی بین این شرط و شروط h در Prolog-EBG (که در روابط ۱۱.۱ و ۱۱.۲ آمده) توجه کنید. این شرط ILP برای h حالت ضعیفتری از شرط رابطه ی ۱۱.۱ است، در ILP فقط لازم است که $(B' \wedge h \wedge x_i) \vdash f(x_i)$ در حالی که در Prolog-EBG شرط بسیار محکـم تـر است، $(h \wedge x_i) \vdash f(x_i)$. همچنین توجه داشته باشید که ILP شرطی در مقابل شرط رابطه ی ۱۱.۲ Prolog-EBG ندارد.

۱۱.۳.۳ بایاس استقرایی یادگیری توضیحی

با توجه به آنچه در فصل ۲ گفته شد، بایاس استقرایی یک الگوریتم یادگیری مجموعه ای از فرض هاست که با آن به همراه نمونه های آموزشی می توان پیشبینی های یادگیر را نتیجه گیری کرد. اهمیت بایاس استقرایی در این است که چگونگی تعمیم یادگیر بر روی نمونه های آموزشی مشاهده شده را مشخص می کند.

اما بایاس استقرایی Prolog-EBG چیست؟ همانطور که در رابطه ی ۱۱.۲ نیز گفته شده، در Prolog-EBG فرضیه ی خروجی h را می توان از $B \wedge D$ نتیجه گرفت. بنابراین، تئوری قلمرو B مجموعه ای از فرض هاست که به همراه نمونه های آموزشی می توان از آن فرضیه ی خروجی را نتیجه گیری کرد. با دانستن اینکه پیشبینی های یادگیر فقط از این فرضیه ی h ناشی می شود، به نظر می رسد که بایاس استقرایی Prolog-EBG فقط تئوری قلمرو ی B است. در واقع این گفته است اما باید یک توضیح کوچک را در نظر گرفت که: مجموعه ی بزرگی از horn clause ها را می توان از تئوری قلمرو نتیجه گیری کرد. پس عناصر باقیمانده ی بایاس استقرایی ناشی از بایاس در انتخاب Prolog-EBG در میان horn clause های ممکن است. همانطور که در بالا دیده می شود، Prolog-EBG از الگوریتمی ترتیبی که تا اتمام تمامی نمونه های مثبت قانون تولید می کند استفاده می کند. علاوه بر آن، هر horn clause کلی ترین حکم ممکن (ضعیفترین پیشفرض) بر اساس توضیحات نمونه ی آموزشی فعلی است. بنابراین، در بین این horn clause ها که از تئوری قلمرو نتیجه گیری می شوند، می توانیم بایاس استقرایی Prolog-EBG را ارجهیت مجموعه ی کوچکی از کلی ترین horn clause ها بدانیم. در واقع، الگوریتم حریص Prolog-EBG فقط یک تقریب برای الگوریتم جستجوی همه جانبه است که واقعا کوتاهترین مجموعه از کلی ترین horn clause ها را پیدا می کند. با این وجود، بایاس استقرایی Prolog-EBG را می توان تقریبا آنچه گفته شد در نظر گرفت.

تقریب بایاس استقرایی Prolog-EBG: تئوری قلمروی B به علاوه ی ارجهیت مجموعه های کوچکتر horn clause های کلی تر.

مهم ترین نکته در اینجا این است که بایاس استقرایی Prolog-EBG (خط مشیی که برای تعمیم روی داده های آموزشی پیش می گیرد) به شدت به تئوری قلمروی ورودی وابسته است. این مخالف عملکرد تمامی دیگر الگوریتم های یادگیری (مثل شبکه های عصبی و درخت یادگیری) است، در دیگر الگوریتم های یادگیری بایاس استقرایی خاصیتی ثابت از الگوریتم یادگیری بود که معمولا به نمایش فرضیه ها وابسته بود. چرا وابستگی بایاس استقرایی Prolog-EBG به ورودی و ثابت نبودن آن اهمیت دارد؟ زیرا که، همانطور که در فصل ۲ و دیگر فصول گفته شد، بایاسی وجود ندارد که در همه ی مسائل کارایی لازم را داشته باشد، همچنین یادگیری بدون بایاس بیهوده است. بنابراین، هر تلاش برای ساخت متد کلی یادگیری حداقل لازم است که بایاس استقرایی متغیری داشته باشد تا بتواند متناسب با مسئله تغییر کند. در سطح کاربردی تر، در بسیاری از کارها طبیعی است که دانشی مربوط به قلمرو مسئله (مثل، دانش درباره ی Weight در مسئله ی SafeToStack) که در چگونگی تعمیم یادگیر بر روی داده های آموزشی تاثیر گذار است در دسترس می باشد. در مقابل، بایاس کردن فرم فرضیه ها (مثل ترجیح درخت های کوتاهتر در یادگیری درختی) طبیعی نیست. در آخر، اگر مشکل بزرگ چگونگی اینکه عامل خودکار[1] توانایی هایش در یادگیری را در طول زمان بهبود می بخشد را در نظر بگیریم، بنابراین جذاب خواهد بود که الگوریتم یادگیری ای داشته باشیم که قابلیت های تعمیمش با جمع آوردی دانش بیشتر از قلمرو بهبود یابد.

[1] autonomous

۱۱.۳.٤ یادگیری مرتبه ی دانش[1]

همانطور که در رابطه ی ۱۱.۲ اشاره شد، فرضیه ی خروجی h از الگورتیم Prolog-EGB از تئوری قلمروی B و نمونه های آموزشی D نتیجه گیری می شود. در واقع، با بررسی الگوریتم Prolog-EBG فهمیدن اینکه h به طور مستقیم از B نتیجه گیری می شود و مستقل از D است بسیار ساده خواهد بود. یکی از راه های پی بردن به این حقیقت بررسی الگوریتم Lemma-Enumerator است. الگوریتم Lemma-Enumerator، خیلی ساده، تمامی درخت های اثبات[2] شامل مفهوم هدف بر اساس فرض ها در تئوری قلمرو ی B را می شمارد. برای هر درخت اثبات، Lemma-Enumerator ضعیفترین پیشفرض را محاسبه می کند و horn clause ی بر اساس آنچه در Prolog-EBG گفته شد تشکیل می دهد. تنها تفاوت Lemma-Enumerator با Prolog-EBG این است که Lemma-Enumerator کاری با داده های آموزشی ندارد و فقط تمامی درخت های اثبات را می شمارد.

توجه دارید که Lemma-Enumerator مجموعه توانی ای (superset) از horn clause ها را که خروجی Prolog-EBG هستند را خروجی می دهد. با دانستن این حقیقت سوالهای بسیاری مطرح می گردد. ابتدا اینکه، اگر فرضیه های خروجی Prolog-EBG مستقیما و تنها از تئوری قلمرو نتیجه گیری می شوند، پس نقش داده های آموزشی در Prolog-EBG چیست؟ جواب در این است که نمونه های آموزشی الگوریتم Prolog-EBG را بر ایجاد قوانینی که توزیع نمونه ها در عمل را پوشش می دهد متمرکز می کند. برای مثال، در مثال شطرنج ما، مجموعه ی تمامی (lemma) ها بسیار بزرگ است، در حالی که مجموعه ی پوزیسیون هایی که در بازی واقعی شطرنج رخ می دهد فقط قسمت کوچکی از آن مجموعه ی بسیار بزرگ است. بنابراین، با تمرکز بر نمونه های آموزشی تولیدی در تمرین ها، برنامه با مجموعه ی کوچکتری از پوزیسیون ها سر و کار خواهد داشت.

سوال دوم مطرح این است که چگونه Prolog-EBG می تواند فرضیه ای را یادبگیرد که به صورت ضمنی در تئوری قلمرو آورده شده است را یاد بگیرد. به عبارت دیگر، آیا این الگوریتم (با این فرض که الگوریتم در میزان محاسبات هیچ محدودیتی نداشته باشد) می تواند دسته بندی نمونه ای را که نمی توان به تنهایی با تئوری قلمرو دسته بندی کرد را یاد خواهد گرفت؟ متاسفانه، الگوریتم نمی تواند. اگر $B \vdash h$، پس هر دسته بندی ای که از h نتیجه گیری خواهد شد را می توان نتیجه گیری ای مستقیم از B دانست. آیا این محدودیت در یادگیری توضیحی یا استنتاجی ذاتی است؟ خیر، این محدودیت ذاتی نیست، در زیر مثالی در این باره آورده شده است.

برای ایجاد نمونه ای از یادگیری استنتاجی که در آن فرضیه ی یادگرفته شده ی h را مستقیما نتوان از B نتیجه گیری کرد، باید حالتی ایجاد کنیم که $B \nvdash h$ اما داشته باشیم $B \wedge D \vdash h$ (که در رابطه ی ۱۱.۲ نیز آمده بود). یکی از حالات جالب حالتی است که B شامل فرض هایی چون فرض "اگر x تابع هدف را راضی کند، پس g(x) نیز تابع هدف را راضی خواهد کرد" است. این فرض به تنهایی دسته بندی هیچ یک از نمونه ها را مشخص نخواهد کرد، با این وجود، زمانی که یک نمونه ی مثبت مشاهده می شود، تعمیم استنتاجی روی نمونه های مشاهده نشده ممکن خواهد شد. برای مثال، یادگیری مفهوم هدف PlayTennis را در نظر بگیرید. فرض کنید که هر روز فقط با ویژگی Humidity توصیف شود و تئوری قلمرو فرض "اگر مفهوم بازی تنیس در x Humidity درست باشد، این مفهوم در Humidity های کمتر نیز درست خواهد بود" است. به فرم رسمی تر می توان این تئوری قلمرو را به صورت زیر نوشت،

$$(\forall x)\ \mathrm{IF}\ \big((\mathrm{PlayTennis} = \mathrm{Yes}) \leftarrow (\mathrm{Humidity} = x)\big)$$

[1] knowledge level learning
[2] proof tree

$$\text{THEN } ((\text{PlayTennis} = \text{Yes}) \leftarrow (\text{Humidity} \leq \text{x}))$$

توجه دارید که از روی این تئوری قلمرو نمی توان هیچ دسته بندی را در مورد مثبت یا منفی بودن یک نمونه ی PlayTennis را نتیجه گرفت. با این وجود، زمانی که یک یادگیر نمونه ی مثبتی با Humidity=.3 را مشاهده می کند، با توجه به تئوری قلمرو، فرضیه کلی زیر را نتیجه می گیرد،

$$(\text{PlayTennis} = \text{Yes}) \leftarrow (\text{Humidity} \leq .3)$$

به طور خلاصه اینکه، این مثال تصوری حالتی از $B \nvdash h$ را نشان می دهد که در آن داریم $B \wedge D \vdash h$. فرضیه یاد گیرفته شده در این حالت پیشبینی هایی را می تواند نتیجه گیری کند که مستقیما از دانش قبلی نتیجه گیری نمی شوند. برای این نوع یادگیری، یادگیری ای که در آن پیشبینی ها فقط به تئوری قلمرو وابسته نیستند، "یادگیری مرتبه دانش" می گویند. مجموعه ی تمامی پیشبینی های نتیجه گیری شده از فرض Y را بستگی ستنتاجی[1] Y می نامند. برتری کلیدی در اینجا این است که یادگیری سطح دانش بستگی استنتاجی B در اینجا زیر مجموعه ای از بستگی استنتاجی B+h است.

نمونه ی دوم یادگیری تحلیلی سطح دانش با نوعی از فرض که تعیین[2] نامیده می شود ایجاد می شود. این نوع یادگیری سطح دانش با جزئیات در (Russel 1989) آمده است. تعیین، فرض می کند که یکی از ویژگی های نمونه کاملا به دیگر ویژگی هایش وابسته است اما نوع وابستگی را مشخص نمی کند. برای مثال، یادگیری مفهوم هدف "افرادی که پرتقالی حرف می زنند" را در نظر بگیرید و همچنین فرض کنید که تئوری قلمرو فرض تعیین "زبانی که افراد به آن حرف می زنند وابسته به ملیت آنهاست" است. این تئوری قلمرو به تنهایی هیچ دسته بندی ای بر روی نمونه ها انجام نمی دهد، با این وجود، اگر با نمونه ی مثبت "جو، ۲۳ ساله، چپ دست، برزیلی، پرتقالی حرف می زند" برخورد کنیم، از روی این نمونه و تئوری قلمرو می توان گفت که "تمامی برزیلی ها پرتقالی حرف می زنند".

هر دو مثال آورده شده نشان می دهند که چگونه با روش استنتاجی می توان فرضیه ای را خروجی گرفت که فقط نتیجه گیری شده از تئوری قلمرو نباشد. در هر دو حالت فرضیه خروجی h در رابطه ی $B \wedge D \vdash h$ صدق کرده اما در $B \vdash h$ صدق نمی کند. در هر دوحالت یادگیر فرضیه ای توجیه شده که نمی توان آنرا به تنهایی از تئوری قلمرو و یا داده های آموزشی بدست آورد استنتاج[3] می کند.

۱۱.٤ یادگیری توضیحی ی دانش کنترل جستجو

همانطور که در بالا گفته شد، محدودیت کاربرد عملی الگوریتم Prolog-EBG نیاز آن به تئوری قلمروی درست و همه جانبه است. مجموعه ای از مسائل یادگیری که در آنها این نیاز به سادگی بر طرف می شود مسائل یادگیری افزایش سرعت جستجو ی برنامه های پیچیده تعیین[4] است. در واقع، بزرگترین مسئله ای که در آن از یادگیری توضیحی استفاده می شود، مسئله ی یادگیری کنترل جستجو است، به این مسئله گاهی مسئله ی افزایش سرعت[5] نیز گفته می شود. برای مثال، بازی کردن بازی هایی چون شطرنج نیاز به جستجوی میان فضایی بزرگ

---

[1] deductive closure
[2] determination
[3] deducts
[4] speed up complex search programs
[5] speedup

از حرکات ممکن و پوزیسون های مختلف برای پیدا کردن بهترین حرکت دارد. بسیاری از مسائل عملی برنامه ریزی و بهینه سازی[1] را می توان به راحتی به صورت مسائل جستجو برای پیدا کردن حرکتی به سوی وضعیت هدف بیان کرد. در چنین مسائلی، تعریف عملگر هـای جسـتجوی قانونی به همراه تعریف هدف جستجو تئوری قلمرویی کامل و درست برای یادگیری کنترل جستجو ایجاد خواهد کرد.

اما دقیقا چگونه باید مسئله ی یادگیری کنترل جستجو را بیان کنیم تا بتوان یادگیری توضیحی را در آن به کار برد؟ مسئله ای کلی از یـادگیری کنترل جستجو یی را در نظر بگیرید که در آن S مجموعه ی تمامی وضعیت های ممکن و O مجموعه ی تمـامی عملگـر هـای ممکـن بـرای تبدیل یک وضعیت به وضعیتی دیگر و G نیز عبارتی تعریف شده روی S است که مشخص می کند کدام وضعیت هـا وضـعیت هـدف هسـتند. مسئله در کل پیدا کردن ترتیبی از عملگر هاست که وضعیت اولیه ی دلخواه s_i را به وضعیت آخر s_f ببرد که s_f عبارت G را راضی می کنـد. یکی از راههای بیان این مسئله تغییر سیستم یادگیری برای یادگیری تابع هدفی مستقل برای هر یک از عملگر های O است. در کل، برای هـر عملگر مثل o در O ممکن است مفهوم هدفی به شکل "مجموعه ی وضعیت هایی که o به یک وضعیت هدف ختم می شـود" تعریـف شـود. البته برای انتخاب دقیق اینکه کدام مفهوم هدف را یاد بگیریم به ساختار داخلی حلال مسئله[2] بستگی دارد که از دانش یادگیرفته شـده اسـتفاده خواهد کرد. برای مثال، اگر حلال مسئله سیستمی با هدفهای میانی[3] است که مسئله را با ایجاد و حل زیر هدف ها[4] حل می کند، ممکـن اسـت بخواهیم در عوض مفهوم "مجموعه وضعیت هایی که باید در آنها زیر هدف A زود تر از زیر هدف B عملی شود" را یاد بگیریم.

یکی از سیستم هایی که از یادگیری توضیحی برای بهبود جستجویش استفاده می کند Prodigy (Carbonell 1990) اسـت. Prodigy سیستم تصمیم گیری ای[5] مستقل از قلمرو[6] است که تعریف قلمروی مسئله را با وضعیت های S و عملگـر هـای O دریافـت مـی کنـد. سـپس مسئله را در فرم "ترتیبی از عملگر ها بیابید که وضعیت اولیه ی s_i را به وضعیتی که در G صدق می کند برساند" حل می کنـد. *Prodigy* از برنامه ریز[7] اهداف میانی که مسئله را به زیر هدف های تجزیه می کند و آنها را حل می کند کمک می گیرد و در آخر نیز این روشـهای حـل را ترکیب کرده و راه حلی برای کل مسئله می یابد. بنابراین، در طی این جستجو برای حل مسئله *Prodigy* مرتبا با سوالاتی نظیر "به کدام زیـر هدف را باید در قدم بعدی رسید؟" و "کدام عملگر برای حل این زیر هدف باید در نظر گرفته شود؟" مواجه است. *(Mintor 1988)* مجتمع سازی یادگیری توضیحی در *Prodigy* را با تعریف مجموعه ای از مفاهیم هدف متناسب با این نوع کنترل تصمیم که مرتبا بـا آن مـواجهیم را توضیح می دهد. برای مثال، یکی از مفاهیم هدف "مجموعه ی وضعیت هایی است که زیر هدف *A* باید قبل از زیر هدف *B* حل شود". مثـالی از قانون یادگرفته شده توسط *Prodigy* برای این مفهوم هدف در قلمرو ی مثال ساده ی روی هم گذاشتن مکعب ها به صورت زیر است،

IF One subgoal to be solved is On(x,y) and

One subgoad to be solved is On(y,z)

THEN Solve the subgoal On(y,z) before On(x,y)

---

[1] scheduling and optimization
[2] problem solver
[3] means-ends
[4] subgoal
[5] planning
[6] domain-independent
[7] *planner*

برای درک این قانون دوباره مثال روی هم گذاشتن مکعب ها را که در شکل ۹.۳ توضیح داده شده بود را در نظر بگیرید. در این مثال، هدف چیدن مکعب ها به صورتی است که کلمه ی "universal" را نمایش دهد. Prodigy این هدف را به زیر هدف هایی تقسیم می کند، زیر هدف هایی نظیر On(U,N) و On(N,I) و ... . توجه دارید که قانون بالا بدین معناست که زیر هدف On(U,N) باید قبل از زیر هدف On(N,I) حل شود. توجیه این قانون (و توضیح آن توسط Prodigy برای یادگیری این قانون) این است که اگر زیر هدف ها را در ترتیب عکس حل کنیم، به تضاد خواهیم رسید و باید حل On(U,N) را برای رسیدن به هدف On(N,I) ضایع کنیم. Prodigy در ابتدا با چنین تضادی برخورد می کند سپس این تضاد را توضیح می دهد و قانونی مثل بالا برای آن ایجاد می کند. اثر برآیند این است که Prodigy از دانش مستقل از قلمرواش برای تشخیص زیرهدف های ناسازگار و از دانش مخصوص قلمرواش از عملگر های قلمرو (برای مثال، این حقیقت که ربات در هر بار فقط یک جعبه را می تواند بلند کند) برای یادگیری قوانین لازمه در این قلمرو برای حل مشابه بالا استفاده می کند---.

استفاده از یادگیری توضیحی برای بدست آوردن دانش کنترل برای Prodigy در بسیاری از قلمروهای مسائل شامل مسئله ی چیدن مکعبهای بالا و مسائل پیچیده تر برنامه ریزی و اجرا موفق ظاهر شده اند. (Minton 1988) آزمایشهایی در سه قلمروی مسئله، که قوانین یادگرفته شده به حل مسئله با ضریب ۲ تا ۴ بهبود بخشیده اند را معرفی می کند. علاوه بر این، کارایی قوانین یادگرفته شده نیز با قوانین دستنویس این تئوری های قلمرو قابل مقایسه بوده است. Minton همچنین تعدادی فرایند گسترش یافته برای یادگیری ساده ی توضیحی ارائه می کند که کارایی یادگیری کنترل جستجو را بهبود می بخشد. این فرایند ها شامل متد های ساده سازی قوانین یادگرفته شده و حذف قوانین یادگیری ای که سودشان کمتر از هزینشان است می شود.

مثال دوم ساختار حل مسئله ی کلی که از یادگیری توضیحی کمک می گیرد سیستم Soar (Laird et al. 1986; Newell 1990) است. Soar دامنه ی وسیعی از استراتژی های حل مسئله شامل روش هدف های میانی Prodigy را پشتیبانی می کند. مشابه Prodigy، با وجود اینکه، Soar با توضیح وضعیت ها یاد می گیرد---. زمانی که این سیستم با جستجویی که جوابی قطعی برایش ندارد مواجه می شود، (مثل این سوال که از کدام عملگرد در گام بعدی استفاده کند)، سیستم --- با استفاده از متد های ضعیف نظیر (generate-and-test) برای تعیین مسیر درست عملیات استفاده می کند. ---

Prodigy و Soar نشان می دهند که متد های مبتنی بر یادگیری توضیحی را می توان برای یادگیری دانش کنترل جستجو در قلمروی مسائل مختلف به کاربرد. با این وجود، تعداد زیادی یا اکثر برنامه های جستجو همچنان از توابع ارزیابی عددی، مشابه آنچه در فصل ۱ توصیف شد، به جای قوانین بدست آمده از یادگیری توضیحی استفاده می کنند. اما دلیل چیست؟ در واقع مسائل کاربردی مهمی در استفاده از EBL برای یادگیری کنترل جستجو وجود دارد. اول اینکه در بسیاری از موارد تعداد قوانین کنترل به باید یادگرفته شوند بسیار زیادند (مثلا در حد هزاران قانون). همینطور که سیستم قوانین کنترل بیشتری را برای بهبود جستجویش یاد می گیرد باید هزینه ی بیشتر و بیشتری در هر مرحله برای جفت کردن[1] این مجموعه از قوانین و وضعیت فعلی بپردازد. توجه دارید که این مشکل به یادگیری توضیحی منوط نمی شود؛ این مشکل برای تمامی سیستم هایی که دانش یادگرفته اشان را با دسته ای از قوانین که تعدادشان افزایش می یابد وجود دارد. الگوریتم های جفت سازی می توانند این مشکل را تا حدی حل کنند، اما این مشکل به طور کامل از بین نمی رود. (Minton 1988) استراتژی های تخمین تجربی این هزینه ی محاسباتی و سود هر قانون را بررسی کرد و قوانین را یادگیر می گیرد که میزان سود تخمینی از میزان هزینه ی تخمینی کمتر باشد و همچنین قوانینی که اثر منفی دارند را حذف می کند. وی نحوه ی استفاده از این تحلیل تاثیر قوانین[2] را برای تعیین تاثیرگذاری یادگیری

[1] match
[2] utility analysis

توضیحی در Prodigy توصیف می کند. برای مثال، در سری ای از مسائل چیدن مکعب، Prodigy ۳۲۸ فرصت یادگیری قوانین جدید دارد اما فقط ۶۹ قانون از این قوانین را استخراج می کند و سرانجام این مجموعه را به ۱۹ قانون کاهش می دهد، و قوانین کم کاربرد را حذف می کند. (Tambe et al. 1990) و (Doorenbos 1993) چگونگی تشخیص انواع قوانین که در کل جفت شدنشان هزینه بر خواهد بود، را به همراه متد های بازنویسی چنین قوانینی در فرم های کارا تر و متد هایی برای الگوریتم های بهینه سازی جفت سازی ارائه می کنند. (Doorenbos 1993) چگونگی اینکه این متد ها به Soar امکان جفت شدن با 100,000 قانون یادگرفته شده بدون افزایش قابل توجه در هزینه جفت شدن بر حالت در قلمرو ی مسئله ای را می دهند را مورد بحث و بررسی قرار می دهد.

مشکل کاربردی دیگر یادگیری توضیحی در یادگیری کنترل جستجو این است که در بسیاری از موارد حتی ساختن توضیحات برای مفهوم هدف بسیار به طور رام نشدنی ای سخت است. برای مثال، در شطرنج ممکن است بخواهیم مفهوم هدفی چون "وضعیت هایی که عملگر A ما را به سمت راه حل بهینه می برد را یاد بگیریم". متاسفانه برای اثبات یا توضیح اینکه چرا A ما را به سمت راه حل بهینه می برد نیاز دارد که نشان دهیم تمامی دیگر عملگر ها نتیجه ای ضعیفتر خواهند داشت. این کار معمولا تلاشی نمایی در عمق جستجو را نیاز خواهد داشت. (Chien 1993) و (Tadepalli 1990) متدهایی برای توضیحات "تنبل"[1] یا افزایشی[2] که در آن روشی ابتکاری برای ایجاد توضیحات جزئی و تخمینی اما قابل اجرا (tractable) را معرفی می کنند. قوانین از این توضیحات ناکامل مشابه زمانی که توضیحات کامل بودند استخراج می شود. البته این قوانین یادگرفته شده ممکن است بخاطر توضیحات ناکامل نادرست باشند. سیستم این مشکل را با نظارت بر کارایی قانون بر روی وضعیت های بعدی اصلاح می کند. اگر قانونی در مشاهدات بعدی اشتباه کند، آنگاه توضیح اصلی به صورت توانی ایجاد خواهد شد تا وضعیت جدید را بپوشاند و قانونی بازنگری شده از این توضیح توانی استخراج خواهد شد.

تلاشهای بسیار تحقیقاتی دیگری درباره ی کاربرد یادگیری توضیحی برای بهبود حلال های مسائل جستجویی انجام گرفته است (Mitchell 1981; Silver 1983; Shavlik 1990; Mahadevan et al. 1993; Gervasio and DeJong 1994; DeJong 1994). (Bennett and DeJong 1996) یادگیری توضیحی را برای مسائل برنامه ریزی ربات که سیستم تئوری قلمروی ناکاملی دارد که جهان واقعی و حرکات را توصیف می کند را بررسی کرده اند. (Dietterich and Flann 1995) اجتماع یادگیری توضیحی را با روشهای یادگیری تقویتی بحث شده در فصل ۱۳ را بررسی کرده اند. (Mitchell and Thrun 1993) نیز کاربرد شبکه های عصبی مبتنی بر یادگیری توضیحی (به الگوریتم EBNN در فصل ۱۲ مراجعه کنید) را برای مسائل یادگیری تقویتی بررسی کرده اند.

۱۱.۵ خلاصه و منابع برای مطالعه ی بیشتر

نکات اصلی این فصل شامل موارد زیر می شود:

بر خلاف متدهای یادگیری استقرایی محض که به دنبال فرضیه ای می گردند که با داده های آموزشی متناسب باشد، متدهای تحلیلی محض به دنبال فرضیه ای می گردند که با دانش اولیه ی یادگیر تطبیق داشته و نمونه های آموزشی را نیز پوشش دهد. انسانها نیز گاهی از دانش قبلی برای هدایت بیان فرضیه های جدید استفاده می کنند. این فصل روشهای تحلیلی محض را بیان می کند، در فصل آتی به روشهای یادگیری ترکیبی تحلیلی استقرایی می پردازیم.

---

[1] lazy
[2] incremental

یادگیری مبتنی بر توضیحات نوعی یادگیری تحلیلی است که یادگیر هر نمونه ی آموزشی را (۱) با استفاده از تئوری قلمرو توضیح داده (۲) این توضیحات را برای تعیین شرط کلی درستی توضیحات بررسی کرده و (۳) فرضیه اش را برای تطابق با این شروط کلی بازنگری می کند.

الگوریتم Prolog-EBG الگوریتم یادگیری مبتنی بر توضیحاتی است که از horn clause های درجه اول برای نمایش تئوری قلمروی ورودی و فرضیه ی یادگرفته استفاده می کند. در Prolog-EBG توضیح، یک اثبات Prolog است و فرضیه استخراجی از این توضیح ضعیفترین پیشنویس این اثبات است. نتیجه، اینکه فرضیه ی خروجی Prolog-EBG را می توان از تئوری قلمرواش نتیجه گیری کرد.

روشهای یادگیری تحلیلی مثل Prolog-EBG، ویژگی های میانی مفیدی را به عنوان اثر جانبی بررسی نمونه های آموزشی ایجاد می کند. این روش تحلیلی برای ایجاد ویژگی مکمل روش آماری ایجاد ویژگی های میانی در متدهایی چون Backpropagation است. (ویژگی های واحد های پنهان)

با وجود اینکه Prolog-EBG فرضیه ای را که از محدوده ی نتیجه گیری تئوری قلمرواش تجاوز کند خروجی نمی دهد، اما روشهای یادگیری استنتاجی چنین قابلیتی را دارا هستند. برای مثال، تئوری قلمروی ---

یکی از کلاسهای مهم مسائلی که تئوری قلمروی کامل و درست برایشان موجود است، کلاس مسائل جستجوی فضاهای حالت بزرگ است. سیستمهایی چون Prodigy و SOAR کارایی متدهای یادگیری مبتنی بر توضیحات را برای پیدا کردن خودکار دانش کنترل جستجوی ای که سرعت حل مسائل را در چنین مسائلی ممکن می سازد اثبات کرده اند.

برخلاف وضوح کارایی متدهای یادگیری توضیحی برای انسانها، پیاده سازی استنتاجی محض مثل Prolog-EBG ضعف هایی، نظیر اینکه فرضیه ی خروجی فقط زمانی که تئوری قلمرو صحیح است درست می باشد، دارد. در فصل بعدی به روش هایی خواهیم پرداخت که متدهای یادگیری تحلیلی و استقرایی را ترکیب کرده تا از تئوری قلمروی ناکامل و داده های آموزشی محدود یادگیری ممکن شود.

ریشه های متدهای یادگیری تحلیلی را می توان در میان کارهای اولیه ی Fikes (1972) در یادگیری عملگر های کلی (macro-operators) در بررسی عملگر های ABSYTIPS و کار Soloway (1977) در استفاده ی دانش قبلی محض در یادگیری پیدا کرد. روشهای یادگیری مبتنی بر توضیحات، مشابه آنچه در طول فصل به آن پرداختیم، اولین بار در تعدادی از سیستم های طراحی شده در اوایل دهه ی ۸۰ شامل DeJong (1981); Mitchell (1981); Winston et al. (1983); Silver (1983) ظهور پیدا کرد. DeJong and Mooney (1986) و Mitchell et al. (1986) توضیحات کلی لازم ----

مهم ترین تلاش های استفاده از یادگیری مبتنی بر توضیحات با تئوری قلمورو های کامل در محدوده ی یادگیری کنترل جستجو یا یادگیری افزایش سرعت (speedup) بوده است. سیستم Soar که توسط Laird et al. (1986) و Prodigy که توسط Carbonell et al. (1990) توصیف شد در میان پیشرفته ترین سیستم هایی هستند که از متدهای یادگیری توضیحی برای یادگیری حل مسائل استفاده می کنند. Rosenbloom and Laird (1986) رابطه ی نزدیک بین متد یادگیری Soar (یا همان chunking) و دیگر متدهای یادگیری توضیحی را بحث می کنند. اخیرا، Dietterich and Flann (1995) ترکیب روشهای مبتنی بر توضیحات را با یادگیری تقویتی برای کنترل جستجو بررسی کرده اند.

با وجود اینکه هدف عمده ی ما در اینجا مطالعه ی یادگیری الگوریتم های یادگیری ماشین است، خالی از لطف نیست که اشاره شود که تحقیقات بر روی یادگیری انسان به این حدس گرویده که یادگیری انسان نیز مبتنی بر توضیحات است. برای مثال، Ahn et al. (1987) و

Qin et al. (1992) مدارکی مبنی بر این حدس کـه انسـان هـا از روشـهای مبتنـی بـر توضـیحات اسـتفاده مـی کننـد ارائـه مـی کننـد. (Wisniewski and Medin 1995) تحقیقاتی را بر روی یادگیری انسان ها که اثر متقابل غنی ای از دانش قبلی و داده هـای مشـاهده شده برای تاثیر در فرآیند یادگیری را نشان می دهد را گزارش می کنند. Kotovsky and Baillargeon (1994) تحقیقـاتی را توصـیف کرده که نشان می دهد حتی بچه های ۱۱ ماهه نیز از دانش قبلی ای که یاد گرفته اند کمک می گیرند.

بررسی انجام گرفته بر روی یادگیری توضیحی مشابه نوع خاصی از متدهای بهینه سازی برنامه هاست که توسط برنامه های Prolog بـه کـار می رود، مثل (Partition evaluation). (Harmelen and Bundy 1988) به رابطه ی بین این دو می پردازد.

تمرینات

۱۱.۱ مسئله ی یادگیری مفهوم هدف "زوج افرادی که در یک خانه زندگی می کنند" را در نظر بگیرید که با HouseMates(x,y) نمـایش داده می شود، در زیر نمونه ای از این مفهوم آورده شده است:

HouseMates(Joe,Sue)

Person(Joe) Person(Sue)

Sex(Joe,Male) Sex(Sue,Female)

HairColor(Joe,Black) HairColor(Sue,Brown)

Height(Joe,Short) Height(Sue,Short)

Nationality(Joe,US) Nationality(Sue,US)

Mother(Joe,Marry) Mother(Sue,US)

Age(Joe,8) Age(Sue,6)

درباره ی این مفهوم هدف تئوری قلمروی زیر موجود است:

HouseMates(x,y)←InSameFamily(x,y)

HouseMates(x,y)←FraternityBrothers(x,y)

InSameFamily(x,y) ← Married(x,y)

InSameFamily(x,y) ← Youngster(x)∧ Youngster(y)∧SameMother(x,y)

SameMother(x,y) ← Mother(x,z)∧Mother(y,z)

Youngster(x) ← Age(x,a)∧LessThan(a,10)

از Prolog-EBG برای یادگیری این مفهوم با تئوری قلمرو و داده های آموزشی بالا استفاده کنید. در کل،

(a) رفتار Prolog-EBG را به طور دستی دنبال کنید؛ توضیحات ایجاد شده برای نمونه های آموزشی و حاصل برازش مفهوم هدف را از این توضیحات و دسته horn clause های حاصل را ثبت کنید.

(b) مفهوم "افرادی که با Joe در یک خانه زندگی می کنند" را به جای "زوج افرادی که در یک خانه زندگی می کنند" را در نظر بگیرید. این مفهوم هدف را با فرمولهای بالا بازنویسی کنید. همان نمونه ی آموزشی و همان تئوری قلمرو را در نظر بگیرید، چه horn clause هایی توسط Prolog-EBG تولید می شود؟---

۱۱.۲ همانطور که در بخش ۱۱.۳.۱ نیز اشاره شد، Prolog-EBG می تواند ویژگی های جدید مفیدی که به طور صریح در نمونه ها بیان نشده اما بر حسب ویژگی های در تعمیم روی نمونه ها مفید است را ایجاد کند. این ویژگی ها بر اثر بررسی توضیحات نمونه های آموزشی ایجاد می شود. متد دیگری که برای پیدا کردن ویژگی های مفید غیر صریح Backpropagation در شبکه های عصبی است، در این متد ویژگی های جدید در واحدهای پنهان بر اساس ویژگی های آماری تعداد زیادی از نمونه ها ایجاد می شود. آیا می توانید راهی برای ترکیب این روشهای آماری و استقرایی برای ایجاد ویژگی های جدید ارائه کنید؟ (توجه: این مسئله هنوز جزو قسمت های آزاد تحقیق محسوب می شود)

≺¬≠≤]⌈∧∨

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

افزایش سرعت	Speedup
افزایش سرعت جستجو ی برنامه های پیچیده تعیین	speed up complex search programs
افزایشی	Incremental
برگ	Leaf nodes
برنامه ریز	Planner
برنامه ریزی و انجام	planning and schedualing
برنامه نویسی منطقی استقرایی	inductive logic programming
بستگی ستنتاجی	deductive closure
پس راندن	Regress
تحلیل تاثیر قوانین	utility analysis
تعیین	Determination
تنبل	Lazy
توجیه	Justification
تئوری قلمرو	domain theory
جستجوی زنجیر وار معکوس	backward chaining search
جفت کردن	Match
حلال مسئله	problem solver
خودکار	Autonomous
درخت های اثبات	proof tree
درست	Correct

Subgoal	زیر هدف
weakest preimage	ضعیفترین پیشنویس
Irrevelant	غیر مرتبط
Assertion	فرض ها
state-space	فضای وضعیتی
Constraint	قید
Perfect	کامل
knowledge compilation	گرداوری دانش
sequential covering algorithm	الگوریتم ترتیبی
Superset	مجموعه توانی ای
Relevant	مرتبط
search-intensive planning and optimization problems	مسائل متمرکز بر جستجوی برنامه ریزی و بهینه سازی
domain-independent	مستقل از قلمرو
Logical	منطقی
negation-as-failure	منفی در زمان شکست
Feature	ویژگی
means-ends	هدفهای میانی
Complete	همه جانبه
analytical learning	یادگیری تحلیلی
explanation-based learning (ELB)	یادگیری توضیحی
knowledge level learning	یادگیری مرتبه ی دانش
macro-operators	عملگر های کلی

Telegram: @Farabar_BI

www.farabar.net

فصل دوازدهم: ترکیب یادگیری تحلیلی و استقرایی

متد های استقرایی محض فرضیه ای کلی را با پیدا کردن قائده های نمونه های آموزشی ایجاد می کنند. متدهای تحلیلی محض نیز از دانش قبلی برای نتیجه گیری منطقی فرضیه کلی استفاده می کنند. این فصل متد هایی را بررسی خواهد کرد که مکانیسم های استقرایی و تحلیلی را ترکیب کرده و از منافع هر دو روش استفاده می کنند: دقت تعمیمی بیشتری با استفاده از دانش قبلی موجود دارند و در زمانی که دانش قبلی ضعف دارد به داده های آموزشی مشاهده شده اتکا خواهند کرد. متد های ترکیبی حاصل از متدهای استقرایی محض و تحلیلی محض کارایی بیشتری خواهند داشت. این فصل متدهای یادگیری استقرایی تحلیلی را بر اساس نمایش های نمادین و شبکه های عصبی مصنوعی بررسی خواهد کرد.

۱۲.۱ انگیزه

در فصل های قبلی دو الگو[1] از یادگیری ماشین را دیدیم: یادگیری استقرایی و یادگیری تحلیلی. متد های استقرایی، مثل یادگیری درختی و Backpropagation در شبکه های عصبی، به دنبال فرضیه های کلی می گردند که متناسب با داده های آموزشی باشد. متد های تحلیلی، مثل Prolog-EBG به دنبال فرضیه هایی می گردند که با پوشش داده های مشاهده شده با دانش قبلی نیز متناسب باشد. این دو الگوی یادگیری بر اساس توضیحات مختلف برای یادگیری فرضیه ها ایجاد شده و مزیت ها و ضعف های مکمل دارند. ترکیب این متد ها، متد های یادگیری ای با قابلیت ها و قدرت بیشتری ایجاد می کند.

متد های یادگیری تحلیلی محض این مزیت را دارند که تعمیم را با استفاده از میزان داده های کمتری انجام می دهند و از دانش قبلی برای کنترل یادگیری استفاده می کنند. با این وجود، ممکن است با دانش قبلی نادرست یا ناکامل به نتایج غلتی بیانجامند. متد های استقرایی محض این مزیت را دارند که نیاز به دانش قبلی ندارند و نتایج را کاملا از خود داده های اموزشی استخراج می کنند. اما زمانی که داده های آموزشی به اندازه ی کافی زیاد نیست امکان دارد بوسیله ی بایاس استقراییشان، که برای تعمیم روی داده های مشاهده شده اعمال شده، به نتایج غلتی

[1] paradigm

www.farabar.net

برسند. جدول ۱۲.۱ خلاصه ی این مزیت ها و ضعف های مکمل متد های یادگیری استقرایی و تحلیلی را نشان می دهد. این فصل به این سوال که "چگونه می توان این دو را در یک الگوریتم که بهترین جنبه های هر دو را داشته باشد گنجاند؟" می پردازد.

تفاوت متد های یادگیری استقرایی و توضیحی را می توان در طبیعت توجیه ی که برای فرضیه های یادگرفته شده اشان ارائه می دهند دید. فرضیه ی خروجی متد های یادگیری تحلیلی محض مثل Prolog-EBG توجیه ی منطقی برای فرضیه های یادگرفته شده ارائه می دهند؛ فرضیه خروجی را می توان از تئوری قلمرو و نمونه های آموزشی نتیجه گیری کرد. فرضیه های خروجی متد های استقرایی محض مثل Backpropagation توجیه آماری برای فرضیه های یادگرفته شده ارائه می دهند؛ فرضیه ی خروجی از متغیر های آماری با فرض اینکه نمونه های آموزشی به اندازه ی کافی زیاد هستند که نماینده ی توزیع نمونه ای حاکم بر آن باشند توجیه می شود. این توجیه آماری برای استقرا در نتایج PAC-learning در فصل ۷ به طور کامل بحث شده است.

با معلوم بودن اینکه متد های تحلیلی فرضیه هایی با توجیه منطقی خروجی می دهند و روش های استقرایی متد هایی با توجیه آماری خروجی می دهند، روشن است که ترکیب این روش ها چگونه مفید خواهد بود: توجیه های منطقی با فرض هایشان یا همان دانش قبلی که بر پایه آن ساخته شده اند محدود می شوند و در زمانی که دانش قبلی نادرست و یا در دسترس نباشد این روشها بدون بازده و غیر قابل اعتماد خواهند بود. توجیه های آماری فقط با داده ها و فرض های آماری که می کنند محدود می شوند و زمانی که فرضهایشان درباره ی توزیع غیر قابل اعتماد است و یا داده ها اندک اند این روش ها بدون بازده و غیر قابل اعتماد خواهند بود. خلاصه اینکه این دو روش برای انواع مختلفی از مسائل درست کار می کنند. با ترکیب آنها می توان امید داشت که روشی کلی تر برای یادگیری که بر روی طیف وسیعی از مسائل یادگیری درست کار می کنند بدست آورد.

جدول ۱۲.۱ طیفی از مسائل یادگیری که در دانش قبلی و حجم داده های آموزشی متفاوت اند را در بر می گیرد. در یک طرف طیف حجم بسیار زیادی از داده های آموزشی موجود می باشد و اثری از دانش قبلی نیست. در طرف دیگر طیف، دانش قبلی بسیار قوی موجود است اما داده های آموزشی بسیار اندک است. مسائل کاربردی اغلب جایی بین این دو سر طیف هستند. برای مثال، در بررسی پایگاه داده پزشکی برای یادگیری مفهوم "علایمی که در آن درمان x بهتر از درمان y است" ممکن است فرض اولیه ای وجود داشته باشد (مثل حالتی از عمل و عکس العمل که مربوط به بیماری است) که می گوید دمای بدن بیمار بیشتر از مربوط تر از حالت میانی بیمار است---. به طور مشابه، در بررسی پایگاه داده ی یک انبار برای یادگیری مفهوم هدف "شرکت هایی که میزان انبارشان در طول ده ماه آینده دو برابر می شود" ممکن است دانش قبلی داشته باشیم که اقتصاد نوعی عمل و عکس العمل است، و سود عمده ی یک شرکت مربوط تر از رنگ مارک شرکت است. در هر دو تعریف مسئله، دانش قبلی غیر همه جانبه است، اما واضح است که در تمیز دادن ویژگی های مربوط از غیر موبوط کاراست.

	یادگیری استقرایی	یادگیری تحلیلی
هدف:	فرضیه ای که با داده ها تطابق داشته باشد	فرضیه ای که با تئوری قلمرو تطابق داشت باشد
توجیه:	توجیه آماری	توجیه استنتاجی
مزیت:	نیاز به دانش قبلی زیادی ندارد	نیاز به داده های آموزشی اندکی دارد
ضعف:	داده های اندک، بایاس غلت	تئوری قلمرو ی غلت

Telegram: @Farabar_BI
www.farabar.net

جدول ۱۲.۱ مقایسه ی یادگیری استقرایی محض با یادگیری تحلیلی محض

Inductive learning	Analytical learning
Plentiful data	Perfect prior knowledge
No prior knowledge	Scarce data

شکل ۱۲.۱ طیفی از کارهای یادگیری.

در یک طرف طیف، دانش قبلی ای وجود ندارد و متد های کاملا استقرایی با پیچیدگی نمونه ای بالا لازم است. در طرف دیگر طیف، تئوری قلمروی کامل موجود است، که استفاده از روش های تحلیلی محض مثل Prolog-EBG را ممکن می کند. مسئله های کاربردی معمولا جایی در میان این طیف قرار دارند.

سوال مطرح در این فصل این است که "چه نوع الگوریتمهایی می توانند از دانش قبلی تقریبی به همراه داده های موجود برای ایجاد برای فرضیه تعمیمی استفاده کنند؟". توجه دارید که حتی زمانی که از روشها استقرایی محض استفاده می کردیم فرصت انتخاب طراحی بر پایه ی دانش قبلی هدف یادگیری داشتیم. برای مثال، زمانی که از Backpropagation برای مسائلی چون تشخیص گفتار[1] استفاده می کردیم انتخاب هایی در طول طراحی ایجاد می شد، انتخاب هایی نظیر نحوه ی کد سازی ورودی و خروجی تابع خطایی که با شیب نزول مینیمم می شود، تعداد واحد های پنهان، پیکربندی یا ساختار شبکه[2]، ضریب یادگیری و تکانه[3] و ... بود. در این انتخاب ها طراح انسانی می تواند دانش مربوطه ی یادگیری را در الگوریتم یادگیری وارد می کند[4]. با این وجود نتیجه روشی استقرایی محض و نمونه ای از Backpropagation خواهد بود که توسط انتخاب های طراح برای تشخیص گفتار تخصصی شده است. اما در اینجا علاقه ی ما به چیز دیگری است. علاقه ی ما به سیستم هایی است که دانش قبلی را به عنوان ورودی صریح مثل داده های آموزشی را به دریافت می کنند، بنابراین این سیستم ها با وجود دریافت دانش قلمرویی[5] سیستم هایی کلی باقی می مانند. به طور خلاصه، علاقه ی ما در اینجا به الگوریتم های مستقل از قلمرو[6] است که از ورودی دانش قلمرویی استفاده می کنند.

از چه معیاری باید برای مقایسه ی روشهای مختلف و ترکیب یادگیری استقرایی و تحلیلی استفاده کنیم؟ با دانستن اینکه در کل یادگیر کیفیت تئوری قلمرو یا داده های آموزشی را نمی داند، پس بیشتر علاقه ی ما به سوی متد های کلی است که می توانند بدون توجه به مکان مسئله در طیف مذکور شکل ۱۲.۱ کار کنند. تعدادی از ویژگی های خاصی که علاقه داریم چنین روش یادگیری ای داشته باشد در زیر آمده است:

بدون تئوری قلمرو، چنین الگوریتمی حداقل باید کارایی در حد روش های استقرایی محض داشته باشد.

با داشتن تئوری قلمرو کامل، باید حداقل کارایی ای در حد روش های تحلیلی محض داشته باشد.

با داشتن تئوری قلمرو ناکامل و داده های آموزشی نا کامل، باید کارایی بهتر از روش های استقرایی محض و روش های تحلیلی محض داشته باشد.

باید الگوریتم با سطح نامعلومی از خطا داده های آموزشی تطبیق داشته باشد.

---

[1] speech recognition
[2] topology
[3] momentum
[4] task-specific knowledge
[5] domain-specific knowledge
[6] domain-independent algorithms

www.farabar.net

باید الگوریتم با سطح نامعلومی از خطا در تئوری قلمرو تطبیق داشته باشد.

توجه دارید که این لیست خواص در حالت ایده ال است. برای مثال، تطبیق خطا با داده های آموزشی حتی برای روشهای آماری بدون داشتن کوچکترین دانش قبلی یا فرضی مبنی بر توزیع خطا مشکل زاست. ترکیب یادگیری استقرایی و تحلیلی هنوز در معرض تحقیق و بررسی است. با وجود اینکه لیست بالا خواصی است که می خواهیم الگوریتممان داشته باشد، هنوز الگوریتم هایی ایجاد نشده که تمامی این قیود را در حالت کلی داشته باشند.

قسمت بعد، بحثی دقیقتر از مسائل ترکیبی استقرایی تحلیلی را در بر می گیرد. زیر قسمت های این قسمت سه روش مختلف برای ترکیب دانش قبلی تقریبی و داده های آموزشی موجود را برای کنترل جستجوی یادگیر به سمت فرضیه ی مطلوب ارائه می کنند. اثبات می شود که هر یک از این سه روش کارایی ای بهتر از روش های استقرایی محض در قلمرو های مختلف دارند. برای مقایسه، از مثالی برای تصور سه روش استفاده خواهیم کرد.

۱۲.۲ روش یادگیری استقرایی تحلیلی

۱۲.۲.۱ مسئله ی یادگیری

مسئله ی یادگیری بحث شده در این فصل را می توان به صورت زیر خلاصه کرد:

ورودی:

مجموعه ی نمونه های آموزشی D، که ممکن است خطا داشته باشد.

تئوری قلمرو B، که ممکن است خطا داشته باشد.

فضای از فرضیه ای ممکن H

خروجی:

فرضیه ای که بهترین تطابق با نمونه های آموزشی و تئوری قلمرو دارد.

منظور دقیق از " فرضیه ای که بهترین تطابق با نمونه های آموزشی و تئوری قلمرو دارد" چیست؟ در کل، آیا فرضیه ای را که کمی بیشتر تطابق بر روی نمونه های آموزشی دارد و کمی کمتر تطابق بر روی تئوری قلمرو دارد را ترجیح می دهیم یا بلعکس؟ می توان به طور دقیقتر با تعریف معیار های خطای فرضیه بر اساس تئوری قلمرو و داده های آموزشی این ابهام را با تعریف بر طرف کرد. با توجه به آنچه در فصل ۵ گفته شد $error_D(h)$ نسبت نمونه هایی که توسط h غلت دسته بندی می شوند است. بیایید $error_B(h)$ را احتمال اینکه دسته بندی نمونه ای تصادفی با h با تئوری قلمرو ی B تطابق نداشته باشد تعریف کنیم. حال می توان فرضیه ی خروجی مطلوب را با توجه به این خطا های تعریف شده تعیین کرد. برای مثال، می توان فرضیه ای را ترکیب خطی ای از این دو معیار را مینیمم می کند را مطلوب قرار دارد.

$$\arg\min_{h\in H} k_D error_D(h) + k_B error_B(h)$$

با وجود اینکه چنین تعبیری در اولین برخورد این عبارت به ذهن می رسد اما هنوز مشخص نیست که مقادیر k_D و k_B (میزان اهمیت نسبی تطابق با داده های آموزشی به نسبت تطابق با تئوری قلمرو) چه مقداری دارند. اگر تئوری قلمرو بسیار ضعیفی داشته باشیم و تعداد داده های

آموزشی زیاد و قابل اعتماد باشد بهتر است که تاثیر $error_D(h)$ بیشتر باشد. در مقابل اگر تئوری قلمرو قوی باشد و نمونه های اندک و پر خطا داشته باشیم بهترین نتیجه با افزایش مقدار نظیر $error_B(h)$ بدست خواهد امد. البته اگر یادگیر در حالت کلی کیفیت تئوری قلمرو و داده های آموزشی را نداند، چگونگی وزن دهی این دو خطا نا مشخص باقی خواهد ماند.

جواب دیگر سوال چگونگی وزن دهی دانش قبلی و داده ها روش بیزی است. با توجه به آنچه در فصل ۶ گفته شد، قضیه ی بیز چگونگی محاسبه ی احتمال ثانویه ی P(h|D) را برای فرضیه ی h و داده های آموزشی D بیان می کند. در کل، قضیه ی بیز احتمال ثانویه را بر اساس داده های مشاهده شده ی D و دانش قبلی ای در غالب P(h) و P(D) و P(D|h) محاسبه می کند. پس می توان P(h) و P(D) و P(D|h) را به صورت فرمی از دانش قبلی یا تئوری قلمرو دانست و می توان قضیه ی بیز را متدی برای وزن دهی این تئوری قلمرو و داده های مشاهده شده ی D برای تعیین احتمال ثانویه ی P(h|D) برای h دانست. نگاه بیزی نگاهی است که باید برای انتخاب فرضیه ای که احتمال ثانویه اش بیشتر است داشت و قضیه ی بیز نیز متدی مناسب برای وزن دهی سهم هر یک از دو عامل دانش قبلی و داده های مشاهده شده است. متاسفانه، فضیه ی بیز به طور ضمنی فرض می کند که دانش قبلی درباره ی P(h) و P(D) و P(D|h) کامل است. در حالی که این کمیت ها فقط به صورت غیر کامل[1] در دسترس اند، قضیه بیز به تنهایی روشی برای ترکیب آنها با داده های مشاهده شده ارائه نمی کند. (یکی از روش های ممکن در چنین شرایطی فرض توزیع احتمال اولیه بر روی خود مقادیر P(h) و P(D) و P(D|h) و محاسبه ی مقدار امید P(h|D) است. با این وجود، این روش نیاز به دانش اضافی در مورد توزیع اولیه ی P(h) و P(D) و P(D|h) دارد، پس در حالت کلی این روش کارامد نیست).

در قسمت های بعدی باز هم به سوال مفهوم "متناسب ترین فرضیه" در طی بررسی الگوریتم های خاص خواهیم پرداخت. اما در حال حاضر، می گوییم که مسئله ی یادگیری، مینیمم کردن معیار ترکیبی ای از خطا ی فرضیه بر اساس داده ها و تئوری قلمرو است.

۱۲.۲.۲ جستجوی فضای فرضیه ای

چگونه می توان تئوری قلمرو و داده های آموزشی را به بهترین وجه ترکیب کرد تا بتوان جستجویی برای فرضیه ای قابل قبول ترتیب داد؟ این سوال در یادگیری ماشین سوالی بدون جواب باقی مانده است. این فصل، چند روش پیشنهادی را بررسی خواهیم کرد. این روش ها اغلب تعمیم روش های استقرایی ای بررسی شده در فصول گذشته (مثل Backpropagation و FOIL) هستند.

یکی از راههای درک محدوده ی روش های مختلف، بازگشت به دید یادگیری به عنوان جستجویی در میان فضای فرضیه ای است. ما اکثر متد های یادگیری را به عنوان الگوریتم برای جستجو فضای فرضیه ای H در نظر می گیریم و آنها را با فضای فرضیه ای ای که جستجو می کنند توصیف می کنیم، فرضیه اولیه که جستجو با آن آغاز می شود را h_0 در نظر بگیرید و مجموعه ی عملگر های جستجو که مرحله های جستجو را معین می کند را O در نظر بگیرید و معیار هدف G که هدف جستجو را مشخص می کند در نظر بگیرید. در این فصل ما به سه روش متفاوت استفاده از دانش اولیه برای تغییر کارایی جستجوی متد های استقرایی محض را بررسی خواهیم کرد.

استفاده از دانش قبلی برای ایجاد یک فرضیه اولیه که جستجو از آن شروع شود. در این روش تئوری قلمرو ی B برای ساخت فرضیه ی اولیه ی h_0 که با B سازگار است به کار می رود. سپس از روشی استقرایی با فرضیه ی اولیه ی h_0 استفاده می شود. برای مثال، سیستم KBANN که در زیر توضیح داده شده است سیستمی است که از شبکه های عصبی با همین روش استفاده

[1] imperfect

می کند. این سیستم از دانش قبلی برای طراحی اتصال های واحد ها و وزن های اولیه ی شبکه استفاده می کند تا شبکه ی اولیه به طور کامل با تئوری قلمرو سازگار باشد. سپس فرضیه ای شبکه ای به صورت استقرایی با Backpropagation و داده های موجود بازنگری می شود. با شروع جستجو از فرضیه ای که با تئوری قلمرو سازگار است، به نظر می رسد که فرضیه ی خروجی بیشتر شبیه تئوری قلمرو باشد.

استفاده از دانش قبلی برای تغییر هدف جستجوی فضای فرضیه ای. در این روش، معیار هدف G طوری تغییر می یابد که فرضیه ی خروجی علاوه بر نمونه های آموزشی با تئوری قلمرو نیز متناسب باشد. برای مثال، سیستم EBNN که در زیر توضیح داده خواهد شد از شبکه های عصبی با این روش استفاده می کند. از آنجایی که یادگیری استقرایی شبکه های عصبی از شیب نزول برای مینیمم کردن خطای مربعی شبکه بر روی داده های آموزشی استفاده می کند، EBNN از شیب نزول برای مینیمم کردن معیاری دیگری استفاده می کند. این معیار شامل جمله ای اضافی است که خطای شبکه ی یادگرفته شده را بر اساس تئوری قلمرو بیان می کند.

استفاده از دانش قبلی برای تغییر مراحل موجود جستجو. در این روش، مجموعه ی عملگر های O توسط تئوری قلمرو تغییر داده می شود. برای مثال، سیستم FOCL که در زیر توضیح داده خواهد شد دسته ای از horn clause ها را به همین روش یاد می گیرد. این روش بر پایه ی سیستم FOIL که جستجویی حریصانه بر روی فضای ممکن horn clause ها انجام می دهد طراحی شده است. در هر مرحله این سیستم فرضیه ی فعلی را با اضافه کردن عبارتی جدید بازنگری می کند. FOCL از تئوری قلمرو برای بسط مجموعه عبارات ممکن اضافه شونده به قانون در هنگام بازنگری فرضیه ها استفاده می کند، و اضافه شدن چندین عبارتی که تئوری قلمرو آنها را استفاده می کند در یک مرحله را ممکن می سازد. در این روش، FOCL حرکات تک پله ای در فضای فرضیه ای که در حالت استقرایی الگوریتم به چندین مرحله احتیاج دارد را ممکن می سازد. این حرکات بزرگ (macro-moves) می توانند به شدت مسیر جستجو را تغییر دهند، بنابراین فرضیه ی حاصل سازگار با داده ها از فرضیه ای که از راه جستجوی استقرایی بدست می آمد متفاوت خواهد بود.

قسمت های بعدی هر یک از این روش ها را توضیح خواهند داد.

۱۲.۳ استفاده از دانش قبلی برای مقدار دهی اولیه ی فرضیه

یکی از روش ها، استفاده از دانش قبلی برای مقدار دهی اولیه فرضیه به صورتی که با تئوری قلمرو مطابقت داشته باشد و بازنگری فرضیه اولیه برای تطابق با داده های آموزشی است. این روش توسط الگوریتم KBANN (شبکه های عصبی بر پایه دانش قبلی[1]) به کار گرفته می شود. در KBANN ابتدا شبکه ی اولیه ای ساخته می شود که در تمامی نمونه ها دسته بندی اش با دسته بندی تئوری قلمرو یکی است. سپس از Backpropagation برای تنظیم وزن ها این شبکه ی اولیه برای تطابق با نمونه های آموزشی استفاده می شود.

تشخیص انگیزه ی این تکنیک بسیار ساده است: اگر تئوری قلمرو درست باشد، فرضیه ی اولیه به درستی تمامی نمونه های آموزشی را دسته بندی خواهد کرد و دیگر نیازی به بازنگری در آن نخواهد بود. با این وجود اگر فرضیه ی اولیه همه ی نمونه های آموزشی را درست دسته بندی نکنده از روشی استقرایی برای بهبود تناسب با داده های آموزشی استفاده خواهیم کرد. توجه داشته باشید که در روش استقرایی محض Backpropagation، وزن ها معمولا با مقادیر اتفاقی کوچکی مقدار دهی اولیه می شدند. مفهوم پشت KBANN این است که اگر تئوری

[1] Knowledge-Based Artificial Neural Network

قلمرو فقط تقریبا درست باشد، مقدار دهی اولیه شبکه به صورتی که با تئوری قلمرو تطابق داشته باشد تقریب شروع بهتری نسبت به مقادیر تصادفی کوچک از تابع هدف است. و متناسبا چنین عملی ما را به سوی تعمیمی با دقت بهتر در فرضیه ی نهایی هدایت خواهد کرد.

این روش مقدار دهی اولیه ی فرضیه در استفاده از تئوری قلمرو در چنین تحقیق مورد بررسی قرار گرفته است، این تحقیق ها شامل (Shavlik and Towell 1989)، (Towell and Shavlin 1994)، (Fu 1989,1993) و (Pratt 1993a, 1993b) می شود. ما از الگوریتم KBANN که در (Shavlik and Towell 1989) مطرح شده برای توضیح این روش استفاده می کنیم.

۱۲.۳.۱ الگوریتم KBANN

الگورتیم KBANN نمونه ای از روش مقدار دهی اولیه ی فرضیه (initialize-the-hypothesis) است. این الگوریتم فرض می کند که تئوری قلمروای که با دسته ای از گزاره ها بیان شده، یا همان horn clause های غیر بازگشتی بیان شده است. یک horn clause زمانی گزاره ای است که متغیری نداشته باشد. ورودی و خروجی KBANN به شرح زیر است:

ورودی:

مجموعه ای از نمونه های آموزشی

تئوری قلمرویی که از horn clause های گزاری ای غیر بازگشتی تشکیل شده است.

خروجی:

شبکه ی عصبی ای که با نمونه های آموزشی تناسب دارد و به سمت تئوری قلمرو بایاس شده است.

دو مرحله ی الگوریتم KBANN ابتدا ایجاد شبکه ای عصبی است که کاملا با تئوری قلمرو تناسب داشته باشد و دوم استفاده از Backpropagation برای بازنگری این شبکه ی اولیه برای متناسب شدن با نمونه های آموزشی است. این جزئیات الگوریتم شامل چگونگی ساخت شبکه ی اولیه در جدول ۱۲.۲ آمده و بخش ۱۲.۳.۲ توضیح داده شده است.

KBANN(Domain_Theory,Training_Examples)

Domain_Theory: مجموعه ای از horn clause های گزاره ای غیر بازگشتی.

Training_Examples: مجموعه ای از زوج مرتب های ورودی خورجی تابع هدف. (به فرم <input,output>).

مرحله ی تحلیلی: شبکه ی عصبی معادل تئوری قلمرو را ایجاد کن.

برای هر ویژگی دلخواه نمونه ها ورودی شبکه ای را ایجاد کن.

برای هر horn clause از Domain_Theory، شبکه ای اولیه با فرایند زیر ایجاد کن:

وردیهای این واحد را به ویژگی هایی که بررسی می کند متصل کن.

برای هر عبارت غیرمنفی حکم، وزن W را به ورودی واحد سیگموید مربوطه نسبت بده.

برای هر عبارت منفی حکم وزن W– را به ورودی واحد سیگموید مربوطه نسبت بده.

مقدار آستانه ی w_0 برای این واحد را مقدار W(n–.5)– قرار بده، در این رابطه n تعداد عبارات غیر منفی حکم است.

ارتباطهای دیگر بین واحدهای شبکه، را با وصل کردن هر واحد شبکه لایه ی i ام به تمامی واحدهای لایه ی i+1ام و همچنین خود لایه ی ورودی کامل کن. مقدار وزنهای تصادفی نزدیک صفر را به این ارتباطات اضافی نسبت بده---.

مرحله ی استقرایی: بازنگری شبکه ی اولیه:

از Backpropagation برای تغییر وزنهای شبکه ی اولیه برای تطبیق با Training_examples استفاده کن.

جدول ۱۲.۲ الگوریتم KBANN.

تئوری قلمرو به شبکه ی عصبی معادل ترجمه می شود (مراحل ۱-۳)، سپس به صورت استقرایی و با استفاده از Backpropagation این شبکه بازنگری می شود (مرحله ی ۴). مقدار متوسط که برای W استفاده می شود ۴ است.

۱۲.۳.۲ یک مثال

برای تصور عملکرد KBANN مسئله ی ساده ی یادگیری آمده در جدول ۱۲.۳ را که از کتاب (Towell and Shavlik 1989) گرفته شده است را در نظر بگیرید. در اینجا نمونه جسمی فیزیکی را با جنسش و وزنش و ... توصیف می کند. هدف یادگیری در اینجا یادگیری مفهوم هدف "فنجان" است که بر روی اجسام فیزیکی تعریف شده است. جدول ۱۲.۳ مجموعه ای از نمونه های آموزشی و تئوری قلمروی مربوط به مفهوم "فنجان" را نشان می دهد. توجه دارید که تئوری قلمروی تعریف شده بر روی "فنجان" این است که باید جسم Stable ، Liftable و OpenVessel باشد. همچنین تئوری قلمرو هر یک از این ویژگی ها را با ویژگی های اولیه ی دیگری تعریف می کند، ویژگی های عملیاتی ای که نمونه ها با آنها توصیف می شوند. توجه داشته باشید که تئوری قلمرو کاملا با نمونه های آموزشی سازگار نیست. برای مثال تئوری قلمرو در دسته بندی درست نمونه های دوم و سوم نمونه های آموزشی نا موفق است. با این وجود، تئوری قلمرو تقریب خوبی از مفهوم هدف به ما می دهد. KBANN از تئوری قلمرو و نمونه های آموزشی برای پیدا کردن فرضیه ای بهتر استفاده می کند.

تئوری قلمرو:

Cup ← Stable, Liftable, OpenVessel

Stable ← BottomIsFlat

Liftable ← Graspable, Light

Graspable ← HasHandle

OpenVessel ← HasConcavity, ConcavityPointsUp

نمونه های آموزشی:

	Cups				Non-Cups			
BottomIsFlat	√	√	√	√	√	√	√	√

ConcavityPointsUp	√	√	√	√	√		√	√		
Expensive	√		√				√		√	
Fragile	√	√			√	√		√		√
HandleOnTop					√		√			
HandleOnSide	√			√					√	
HasConcavity	√	√	√	√	√		√	√	√	√
HasHandle	√			√	√		√		√	
Light	√	√	√	√	√	√	√		√	
MadeOfCeramic	√				√		√	√		
MadeOfPaper				√					√	
MadeOfstyrofoam		√	√			√				√

جدول ۱۲.۳ کار یادگیری مفهوم فنجان.

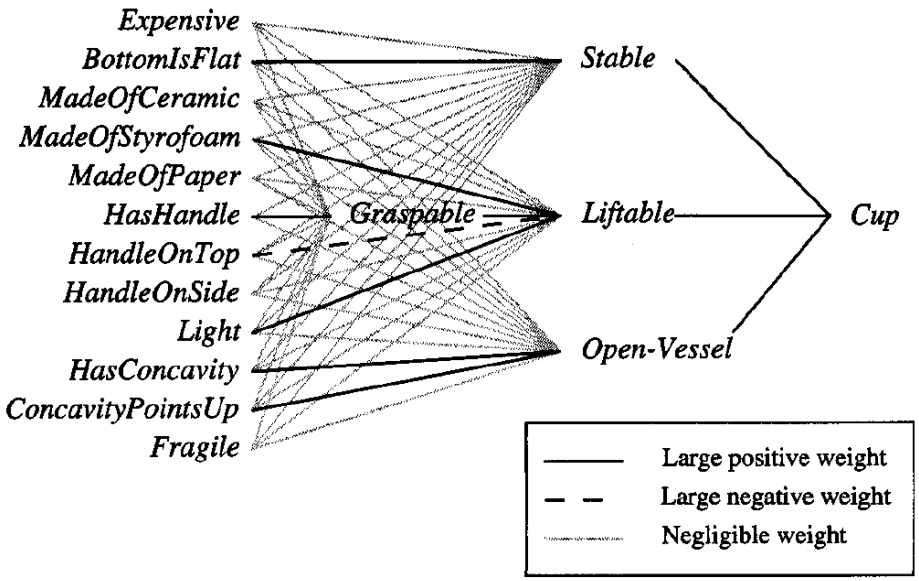
تئوری قلمرویی تقریبی و مجموعه ای از نمونه های آموزشی برای مفهوم هدف "فنجان".

در مرحله ی اول الگوریتم KBANN (سه قدم اول) شبکه ای اولیه که با تئوری قلمرو سازگار باشد ساخته می شود. برای مثال، شبکه ی ساخته شده برای مفهوم "فنجان" در شکل ۱۲.۲ آمده است. در کل شبکه با ایجاد واحد سیگموید حد آستانه ای برای هر horn clause تئوری قلمرو ساخته می شود. مقادیر ورودی کمتر از 0.5 به منزله ی غلت و بزرگتر از 0.5 به منزله ی درست در نظر گرفته می شود. سپس هر واحد طوری ساخته می شود که مقدارش در صورت درست بودن horn clause بزرگتر از ۰.۵ باشد. برای هر فرض horn clause ورودی ای متناسب به واحد سیگموید داده می شود. سپس وزن های گره های سیگموید طوری تعیین می شود که واحد به صورت AND منطقی عمل کند. در کل، برای هر ورودی متناسب با یک شرط غیر منفی، مقدار وزن W (که مقدار مثبتی است) قرار داده می شود. برای هر ورودی متناسب با یک شرط منفی مقدار W– نسبت داده می شود. و مقدار آستانه ی w_0 نیز (n-.5)W– نیز در نظر گرفته می شود، در این مقدار n تعداد ورودی های غیر منفی است. زمانی که مقادیر مقادیر ۱ یا صفر است، مطمئن خواهیم بود که مجموع وزندارشان به علاوه ی w_0 مثبت است (بنابراین خروجی نیز از ۰.۵ بیشتر خواهد بود) پس فقط و فقط اگر تمامی عبارات شرط مثبت باشند خروجی نیز مثبت خواهد بود. توجه دارید که واحد های سیگموید در لایه ی ثانویه نیازی نیست که دقیقا ۱ یا صفر باشند و بحث بالا کاملا درست نخواهد بود. با این وجود، اگر مقدار W به اندازه ی کافی بزرگ انتخاب شود، KBANN می تواند به درستی تئوری قلمرو را برای هر شبکه ای با هر عمق دلخواه را نمایش دهد. (Towell and Shavlik 1994) در گزارشاتشان از استفاده از این الگوریتم از W=4 استفاده کرده اند.

ورودی هر واحد سیگموید به ورودی شبکه یا خروجی واحد های مربوطه متصل است تا گراف وابستگی ویژگی مربوطه ی تعریف شده در تئوری قلمرو را نشان دهد. به عنوان آخرین قدم این مرحله تعداد قابل توجهی ورودی به واحد آستانه اضافه می شود که وزنهای نظیرشان تقریبا صفر

است. نقش این ارتباط های مذکور، دادن توانایی لازم به شبکه برای یادگیری وابستگی احتمالی ویژگی مربوطه به دیگر ویژگی های شبکه است. خطوط پر رنگ در شبکه ی ۱۲.۲ ارتباط هایی با وزن W و خطوط کمرنگ ارتباط هایی با وزن تقریبا صفر را نشان می دهد. مشهود است که با بزرگ بودن W به اندازه ی کافی این شبکه مقادیر نظیر تئوری قلمرو را نشان خواهد داد.

مرحله دوم KBANN (پله ی چهارم در جدول ۱۲.۲) استفاده از Backpropagation در بازبینی وزن های اولیه ی شبکه است. البته اگر تئوری قلمرو و نمونه های آموزشی خطایی نداشته باشند، شبکه ی اولیه با نمونه های آموزشی نیز سازگار خواهد بود. در مثال "فنجان"، چون تئوری قلمرو و داده های آموزشی باهم سازگار نیستند، این مرحله وزن های شبکه ی اولیه را تغییر خواهد داد. شبکه ی آموزش داده ی مثال "فنجان" در شکل ۱۲.۳ آورده شده است، خطوط توپر بالا ترین مقادیر وزن ها و خط چین بالا ترین مقادیر وزن های منفی را نشان می دهد و خطوط کمرنگ نماینده ی وزن های قابل صرف نظر است. با وجود اینکه شبکه ی اولیه چندین نمونه ی آموزشی را اشتباه دسته بندی می کند، شبکه ی بازبینی شده ی شکل ۱۲.۳ کاملا تمامی نمونه های آموزشی را دسته می کند.



شکل ۱۲.۳ حاصل بازبینی استقرایی شبکه ی اولیه.
KBANN از نمونه های آموزشی برای تغییر وزنهای شبکه ی اولیه ناشی از تئوری قلمرو استفاده می کند. توجه دارید که وابستگی Liftable به MadeOfStyrofoam و HandleOnTop در تئوری قلمرو نبود.

مقایسه ی وزن های اولیه با وزن های نهایی شبکه نتایج بسیار جالبی در بر خواهد داشت. همانطور که در شکل ۱۲.۳ نیز دیده می شود، وابستگی شدیدی در مرحله ی استقرایی کشف می شود، مثل وابستگی واحد Liftable به ویژگی MadeOfStyrofoam. توجه به اینکه گره Liftable توسط horn clause ی در تئوری قلمرو تعریف شده اند اما توسط Backpropagation به ویژگی دیگری از شبکه وابسته می گردد. بعد از آموزش شبکه، این واحد مفهومی متفاوت با مفهوم اولیه ی Liftable خواهد داشت.

۱۲.۳.۳ نکات

خلاصه، KBANN به صورت تحلیلی شبکه ای هم ارز با تئوری قلمرو ارائه شده ایجاد می کند، سپس در این فرضیه ی اولیه را برای تناسب بهتر با نمونه های آموزشی تجدید نظر می کند. در این کار، این الگوریتم برای تصحیح عدم تطابق تئوری قلمرو و داده های مشاهده شده وزن های شبکه را تغییر می دهد.

برتری مهم در مورد KBANN بر روشهای استقرایی محض مثل Backpropagation (که از شبکه ای با وزن های تصادفی شروع می شود) این است که زمانی که تئوری قلمرو نسبتا درست است تعمیم بهتری نسبت به Backpropagation دارد، این برتری زمانی که نمونه های آموزشی خطای زیادی دارد بیشتر دیده خواهد شد. KBANN و دیگر روش هایی که از فرضیه ابتدایی کمک می گیرند در مسائل واقعی کارایی بهتری نسبت به سیستم های استقرایی محض از خود نشان داده اند. برای مثال، (Towell 1990) کاربرد KBANN در مسئله ای درباره ی ساختار ملکولی ژنتیکی را ارائه می کند. در این کاربرد هدف یادگیری تشخیص قسمت هایی از DNA به نام promoter region است که بر عملکرد ژن تاثیر دارد. در این تحقیق به KBANN تئوری قلمرویی اولیه که از یک محقق ژنتیک[1] دریافت شده بود به همراه مجموعه ای از ۵۳ نمونه ی مثبت و ۵۳ نمونه ی منفی داده شد. کارایی سیستم با استراتژی leave-one-out مورد بررسی قرار گرفت و سیستم ۱۰۶ بار آموزش داده شد. در هربار اجرای حلقه ی KBANN با ۱۰۵ نمونه از ۱۰۶ نمونه آموزش داده شد و بر روی نمونه ی باقیمانده تست شد. نسبت خطای[2] این ۱۰۶ آزمایش را می توان تخمینی از خطای واقعی دانست. KBANN نسبت خطای 4/106 را پیدا کرد درحالی که نسبت خطای Backpropagation، 8/106 شد. نسخه های مختلف KBANN توسط (Fu 1993) به کارگرفته شده که وی به خطای 2/106 نیز بر روی همان داده ها رسیده است. بنابراین، اثر دانش قبلی در این آزمایشات کاهش قابل توجه نسبت خطا بوده است. نمونه های آموزشی این آزمایش در http://www.ics.uci.edu/~mlearn/MLRepository.html موجود می باشد.

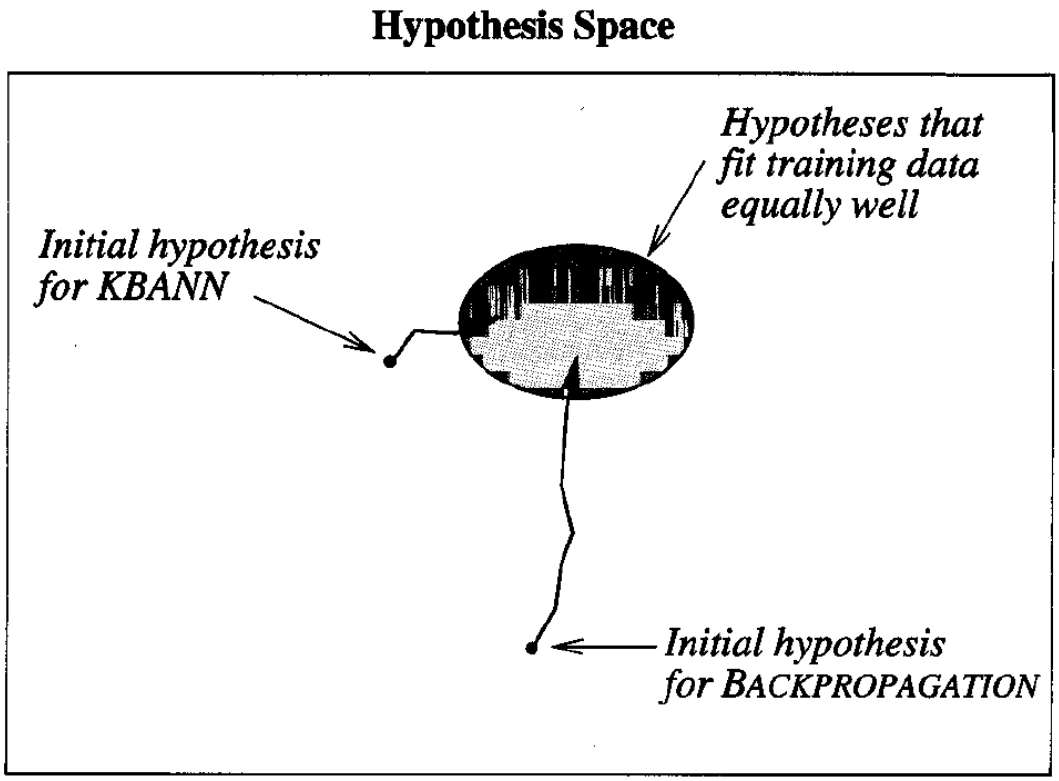
در هر دو (Fu 1993) و (Towell 1990) ذکر شده که دسته قوانین استخراجی از شبکه ی آموزش یافته تئوری قلمرویی تجدید نظر یافته به ما می دهند که سازگاری بیشتری با نمونه های آموزشی دارد. با وجود اینکه گاهی بدست آوردن دسته قوانین Horn clause از شبکه یاد گرفته شد ممکن است اما در حالت کلی این کار با مشکلاتی همراه است زیرا که بعضی وزن ها معادل horn clause صریحی ندارند. (Craven and Shavlik 1994) و (Craven 1996) متد های متفاوتی برای استخراج قوانین سمبولیک از شبکه های یادگرفته شده ارائه می کنند.

برای پی بردن به اهمیت KBANN بد نیست که تفاوت جستجوی فرضیه ای آنرا با الگورتیم استقرایی محض Backpropagation مقایسه کنیم. فضای فرضیه ای ای که این دو روش جستجو می کنند یکی است و در شکل ۱۲.۴ نشان داده شده است. همانطور که مشاهده می کنید، تفاوت اساسی در فرضیه ابتدایی است که از آنجا جستجو آغاز می گردد. در شرایطی که چندین فرضیه ی مختلف (بردار وزنهای مختلف) با داده ها سازگارند، شرطی که معمولا زمانی که نمونه ها خطا دارند پیش می آید، KBANN بیشتر به سمت فرضیه ای میل می کند که تعمیم روی داده ها را مشابه تئوری قلمرو انجام می دهد. از طرف دیگر، Backpropagation معمولا به فرضیه هایی میل می کند که که وزن های کوچکی دارند، چنین فرضیه هایی مشابه درونیابی هموار بین نمونه های آموزشی است. به طور خلاصه، KBANN از تئوری خاص قلمرو استفاده کرده تا تعمیم روی نمونه ها را بایاس کند در حالی که Backpropagation از قائده ای مستقل از قلمرو برای بایاس کردن شبکه به سمت وزن های کوچکتر استفاده می کند. توجه دارید که در این خلاصه ما از اثر مینیمم های موضعی چشم پوشی کرده ایم.

[1] geneticist
[2] error rate

Telegram: @Farabar_BI

www.farabar.net



شکل ۱۲.۴ جستجوی فضای فرضیه ای KBANN.

KBANN شبکه را متناسب با تئوری قلمرو مقدار دهی اولیه می کند، در حالی که backpropagation شبکه را با مقادیر تصادفی مقدار دهی اولیه می کند. هر دو سیستم از تغییر وزنهای کوچک با استفاده از شیب نزول استفاده می کنند. زمانی که چندین فرضیه با نمونه های آموزشی سازگارند، ناحیه ی هاشور زده، KBANN و Backpropagation ممکن است به فرضیه های متفاوتی ختم شوند، زیرا که نقاط شروع متفاوتی دارند.

محدودیت KBANN این حقیقت است که این روش فقط با تئوری قلمرو های گزاره ای سازگار است؛ به عبارت دیگر فقط horn clause هایی را می توان به این روش داد که متغیری در آنها وجود نداشته باشد. البته هنگامی که تئوری قلمرو بسیار پر خطا باشد ممکن است این روش به نتیجه ی کاملا اشتباهی برسد و در این حالت دقت تعمیم این روش از Backpropagation نیز کمتر خواهد بود. با این وجود، این روش و الگوریتم های مربوطه در مسائل کاربردی زیادی مفید واقع شده اند.

KBANN روش مقدار دهی ابتدایی فرضیه ی را برای ترکیب یادگیری تحلیلی و یادگیری استقرایی را توصیف می کند. دیگر نمونه های این روش در (Fu 1993)، (Gallant 1988)، (Bradshaw 1989)، (Yang and Bhargava 1990) و (Lacher 1991) آمده است. این روش ها در تکنیک ساخت شبکه ی اولیه، کاربرد Backpropagation برای تنظیم وزن ها و در متد استخراج توصیف سمبولیک از شبکه های بازنگری شده با هم متفاوتند. (Pratt 1993a, 1993b) روشی از فرضیه ی ابتدایی را معرفی می کند که در آن دانش قبلی از شبکه ی آموزش یافته ای برای کاری مشابه بدست می آید و از تئوری قلمرو سمبولیک استفاده نمی شود. متد های یادگیری مقادیر شبکه ی باور بیزی ای که در قسمت ۶.۱۱ معرفی شده اند، را می توان دانش قبلی برای مقدار دهی اولیه فرضیه قرار داد. در این حالت دانش قبلی متناسب با مجموعه ای از فرض های استقلال خواهد بود که ساختار گرافی شبکه ی بیزی را مشخص می کند، جداول احتمالات این مقادیر از داده های آموزشی استخراج می شود.

www.farabar.net

۱۲.٤ استفاده از دانش قبلی برای تغییر هدف جستجو

روش بالا با جستجوی شیب نزول و با شروع از فرضیه ای که با تئوری قلمرو سازگار است شروع می شود و در ادامه برای تطابق با داده های آموزشی این فرضیه تغییر داده می شود. روش جایگزینی برای استفاده از دانش قبلی وجود دارد، آن هم اضافه کردن اطلاعات دانش قبلی در معیار خطای شیب نزول است، پس شبکه مجبور خواهد بود تا به تابعی ترکیبی از نمونه های آموزشی و تئوری قلمرو میل کند. در این بخش، استفاده از تئوری قلمرو بدین صورت را بررسی خواهیم کرد. در کل، به دانش قبلی به دید مشتق معلوم تابع هدف نگاه می کنیم. انواع خاصی از دانش قبلی را می توان به صورت طبیعی با این فرم نشان داد. برای مثال، در آموزش شبکه ی عصبی برای تشخیص کاراکتر های دست نویس می توانیم مشتقات خاصی از تابع هدف را برای بیان دانش قبلی اینکه "کاراکتر به انتقال و دوران تصویر وابسته نیست" استفاده کنیم.

در زیر به الگوریتم TangentProp می پردازیم که شبکه ای عصبی را با توجه به مقادیر نمونه های آموزشی و مشتقات آموزشی آموزش می دهد. قسمت ۱۲.۴.۴ چگونگی به دست آوردن این مشتقات از تئوری قلمرویی مشابه آنچه در مثال "فنجان" مورد استفاده قرار گرفت را توضیح خواهد داد (قسمت ۱۲.۳). در کل، این قسمت چگونگی ایجاد توضیحات برای استخراج مشتقات آموزشی از تک نمونه های آموزشی در الگوریتم EBNN برای استفاده در TangentProp را توضیح خواهد داد. TangentProp و EBNN نشان داده اند که در بسیاری از تئوری قلمروها شامل تشخیص کاراکترهای دستنویس و تشخیص اشیا و درک و کنترل ربات از روشهای استقرایی محض کارایی بهتری دارند.

۱۲.٤.۱ الگوریتم Tangentprop

الگوریتم Tangentprop (Simard 1992) اطلاعات تئوری قلمرو را با مشتقاتی از تابع هدف نسبت به تغییر ورودی هایش بیان می کند. کار یادگیری ای را با فضای نمونه ای X و تابع هدف f در نظر بگیرید. تا الان فرض بر این بود که نمونه های آموزشی به صورت زوج مرتب های $< x_i, f(x_i) >$ بیان می شود که در آن x_i یک نمونه و $f(x_i)$ مقدار یادگیری آن است. الگوریتم TangentProp فرض می کند که علاوه بر مقدار تابع مقدار مشتقات تابع هدف نیز در نمونه های آموزشی آورده شده است. برای مثال، اگر نمونه ی x_i با یک مقدار واقعی توصیف شود هر نمونه ی آموزشی ممکن است به صورت $< x_i, f(x_i), \frac{\partial f(x)}{\partial x}|_{x_i} >$ بیان شود. که در این نمونه ی آموزشی $\frac{\partial f(x)}{\partial x}|_{x_i}$ مشتق تابع هدف f نسبت به x در نقطه ی $x = x_i$ است.

برای ایجاد شهود مزیت داشتن مقادیر مشتق علاوه بر مقادیر تابع، کار یادگیری ساده ی آمده در شکل ۱۲.۵ را در نظر بگیرید. در این شکل هدف یادگیری تابع هدف f نمودار سمت چپ با استفاده از سه نمونه ی آموزشی $< x_1, f(x_1) >, < x_2, f(x_2) >$ و $< x_3, f(x_3) >$ است. با داشتن این سه نمونه ی آموزشی می توان انتظار داشت که الگوریتم Backpropagation تابع فرضیه ای هموار مثل تابع g را که در شکل وسط آورده شده یاد بگیرد. شکل سمت راست اثر داشتن مشتقات آموزشی یا شیب ها را به عنوان اطلاعات اضافی برای هر نمونه ی آموزشی (مثل $< x_1, f(x_1), \frac{\partial f(x)}{\partial x}|_{x_1} >$) را نشان می دهد. با تناسب یادگیر هم با مقادیر آموزشی $f(x_i)$ و هم مشتقات آموزشی $\frac{\partial f(x)}{\partial x}|_{x_i}$، یادگیر شانس بیشتری در تعمیم روی داده های آموزشی خواهد داشت. به طور خلاصه اثر در نظر گرفتن مشتقات آموزشی کاهش بایاس زبانی[1] الگوریتم Backpropagation به سوی تعمیم هموار بین نقاط خواهد بود و در مقابل اطلاعات ورودی محض مشتقات آموزشی جایگزین خواهند شد. فرضیه ی حاصل h در راسترین تصویر شکل تخمینی با دقت بسیاری بهتری از تابع حقیقی f دارد.

[1] syntactic

در نمونه ی بالا فقط انواع ساده ی مشتق تابع هدف در نظر گرفته شده است. در واقع، Tangentprop فقط مشتقات آموزشی ای را دریافت می کند که بر حسب تبدیلات ورودی x باشند. برای مثال، فرض کنید، هدف یادگیری تشخیص کاراکتر های دست نویس است. فرض کنید که ورودی x متناسب با تصویر یک کاراکتر باشد و هدف نیز دسته بندی درست این کاراکتر باشد. در این کار ممکن است علاقه داشته باشیم که یادگیر بداند که "تابع هدف به چرخش های کوچک کاراکتر در تصویر حساس نیست". برای بیان این دانش قبلی به یادگیر، ابتدا یک تبدیل مثل s(α,x) را تعریف می کنیم که تصویر x را α درجه دوران می دهد. حال می توانیم فرضمان را درباره ی دوران را با این عبارت که مشتق تابع هدف نسبت به این دوران صفر است بیان کنیم (بدین معنا که دسته بندی کاراکتر با چرخش عوض نمی شود). به عبارت دیگر، می توان مشتق آموزشی زیر را برای هر نمونه ی آموزشی x_i در نظر گرفت،

$$\frac{\partial f(s(\alpha, x_i))}{\partial \alpha} = 0$$

در این رابطه f تابع هدف و $s(\alpha, x_i)$ نیز تصویر حاصل از اعمال تبدیل s بر روی تصویر x_i است.

اما TangentProp چگونه از چنین مشتقات آموزشی برای تغییر مناسب وزن های شبکه عصبی استفاده می کند؟ در TangentProp این مشتقات آموزشی در تابع خطایی که توسط Backpropagation مینیمم می شود قرار می گیرد. با توجه به آنچه در فصل ۴ درباره ی الگوریتم Backpropagation گفته شد، این الگوریتم از شیب نزول برای مینیمم کردن مجموع خطاهای مربعی استفاده می کند،

$$E = \sum_i \left(f(x_i) - \hat{f}(x_i)\right)^2$$

در این رابطه x_i نشان دهنده ی iامین نمونه ی آموزشی است و f نیز خود تابع هدف است و $\hat{f}$ تابع یادگرفته شده توسط شبکه ی عصبی است.

در *TangentProp* جمله ای اضافی به تابع خطا اضافه می شود تا اختلاف بین مشتقات آموزشی و مشتقات تابع یادگرفته شده را در شبکه ی عصبی یا $\hat{f}$ نیز تاثیر دهد. در کل *TangentProp* تبدیلات چند گانه را قبول می کند (برای مثال، ممکن است بخواهیم به طور همزمان عدم تاثیر چرخش و انتقال کاراکتر را به یادگیر نشان دهیم). هر تبدیل باید به فرم $s_j(\alpha, x)$ باشد که در آن α پارامتری پیوسته و s_j نیز مشتق پذیر است و داریم $s_j(0, x) = x$ (برای مثال، برای چرخش ۰ درجه تبدیل همانی خواهد بود). برای هر تبدیل با فرم $s_j(\alpha, x)$، Tangentprop خطای مربعی بین مشتق آموزشی و مقدار واقعی مشتق شبکه ی یادگرفته شده را در نظر می گیرد. خطای تغییر یافته را می توان به فرم زیر بیان کرد،

$$E = \sum_i \left[\left(f(x_i) - \hat{f}(x_i)\right)^2 + \mu \sum_j \left(\frac{\partial f\left(s_j(\alpha, x_i)\right)}{\partial \alpha} - \frac{\partial \hat{f}\left(s_j(\alpha, x_i)\right)}{\partial \alpha} \right)^2_{\alpha=0} \right] \quad (12.1)$$

در این رابطه μ ثابتی است که توسط کاربر تعیین می شود که اهمیت نسبی تناسب با مشتقات آموزشی را در مقابل اهمیت تناسب با مقادیر آموزشی را بیان می کند. توجه دارید که جمله ی اول این تعریف E همان تعریف اصلی خطای مربعی برای مقادیر آموزشی است و جمله دوم خطای مربعی مقادیر مشتقات آموزشی است.

(Simard 1992) قانون شیب نزول را برای مینیم کردن تابع خطای تعمیم یافته ی E ارائه می کند. این تابع خطا را نیز می توان بـه همـان روش فصل ۴ برای استخراج قانون ساده ی backpropagation استخراج کرد.

۱۲.٤.۲ مثالی توصیفی

(Simard 1992) نتایج بدست آمده از مقایسه ی تعمیم دقت TangentProp و متد استقرایی محض Backpropagation را بـرای مسئله ی تشخیص کاراکتر های دستنویس مشخص می کند. به صورت خاص تر، هـدف در ایـن یـادگیری دسـته بنـدی تصـاویر حـاوی تـک رقم های ۰ تا ۹ است. در یک آزمایش هر دو الگوریتم TangentProp و Backpropagation با مجموعه ای از نمونه های آموزشی با اندازه های متفاوت آموزش داده شدند و سپس کارایی خروجی بر روی مجموعه ای از ۱۶۰ نمونه بررسی شد. دانش قبلی در نظر گرفته شـده در TangentProp این حقیقت بود که دسته بندی اعداد به انتقال افقی و عمودی تصویر وابسته نیست (مثلا مشتق تابع هدف نسـبت بـه ایـن تبدیلات صفر در نظر گرفته شده است). نتایج حاصل از این آزمایش در شکل ۱۲.۴ نشـان داده شـده اسـت، ایـن نتـایج نشـان مـی دهـد کـه TangentProp با استفاده از دانش قبلی در تعمیم روی نمونه ها کارایی بیشتری از روش استقرایی محض Backpropagation دارد.

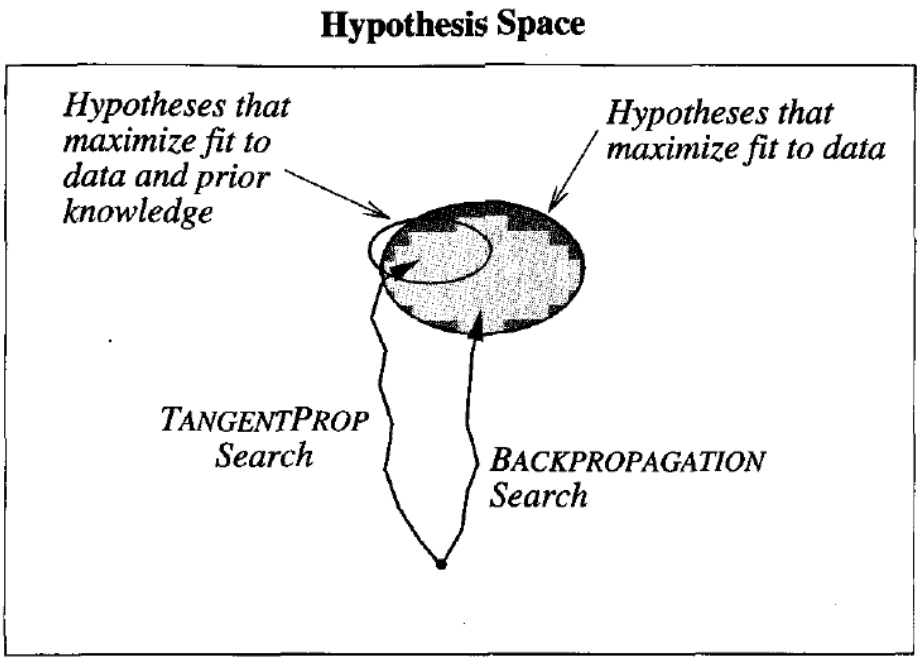
۱۲.٤.۳ نکات

خلاصه، TangentProp از دانش قبلی به فرم مشتقات انتظاری تابع هدف نسبت به تبدیلات استفاده می کند. این روش دانـش قبلـی را بـا داده های آموزشی مشاهده شده ترکیب می کند تا با مینیمم شدن این تابع هدف خطای شبکه هم بـرای مقـادیر آموزشـی و هـم بـرای مقـادیر مشتقات آموزشی مینیمم شود (شبکه هم با مقادیر آموزشی و هم دانش قبلی سازگار شود). مقدار ثابت μ درجه ی نسبی اهمیت این دو خطـا را در خطای کل تعیین می کند. رفتار این الگوریتم به مقدار μ حساس است و μ باید توسط طراح تعیین شود.

با وجود اینکه TangentProp در ترکیب دانش قبلی و داده های آموزشی برای هدایت یادگیری شبکه ی عصبی موفق است امـا نسـبت بـه خطای دانش قبلی آسیب پذیر است. حالتی را در نظر بگیرید که دانش قبلی اشتباه باشد، به عبارت دیگر، مشتقات آموزشی ورودی مشتقات تـابع هدف واقعی را نشان ندهند. در چنین شرایطی، الگوریتم سعی می کند تا به سمت مشتقات آموزشی اشتباه همگرا شود، و در نتیجه ممکن اسـت تعمیم الگورتیم از Backpropagation نیز ضعیف تر شود. اگر در حالت پیشرفته تر میزان درجه ی خطای مشـتقات آموزشـی معلـوم باشـد می توان از چنین اطلاعاتی برای تعیین ثابت μ استفاده کرد و اهمیت نسبی مشتقات آموزشی را نسبت به مقادیر آموزشی مشخص کرد. امـا در کل معمولا چنین اطلاعاتی در حالت کلی در دسترس نیست. در قسمت بعد به الگوریتم EBNN خواهیم پرداخت که به طور خودکار مقدار μ را برای هر نمونه ی آموزشی به طور مجزا و بر حسب احتمال غلت بودن دانش قبلی تعیین می کند.

مقایسه ی جستجوی فضای فرضیه ای (فضای وزن ها) ی الگوریتم TangentProp و KBANN و Backpropagation نتایج جـالبی در بر دارد. TangentProp دانش قبلی را در جستجوی فرضیه ای تاثیر می دهد و هدف جستجوی شیب نزول را با آن تغییر می دهـد. ایـن تغییـر متناسـب بـا تغییـر هـدف جسـتجوی فضـای فرضـیه ای اسـت، مثـالی شـهودی از ایـن تغییـر در شـکل ۱۲.۶ آمـده اسـت. مشـابه Backpropagation (و نه مشابه KBANN)، TangentProp جستجوی خود را با شـبکه ای بـا وزن هـای کوچـک و تصـادفی آغـاز می کند. با این وجود قانون آموزش شیب نزول این الگوریتم با Backpropagation تفاوت دارد و این تفاوت باعث می شود که ایـن روش به فرضیه ی انتهـایی دیگـری میـل کنـد. همـانطور کـه در شـکل نیـز نشـان داده شـده اسـت، مجموعـه ای از فرضـیه هـا کـه تـابع هـدف TangentProp را مینیمم می کند ممکن است با مجموعه فرضیه هایی که هدف Backpropagation را مینیمم می کند متفاوت باشد. مخصوصا زمانی که نمونه های آموزشی و دانش قبلی هر دو درست باشند، و همچنین بتوان تابع هدف را با شبکه ی ANN در نظر گرفته شده

نمایش داد، مجموعه ی بردار های وزنی که هدف TangentProp را راضی می کند زیر مجموعه ای از مجموعه بردارهای راضی کننده ی هدف backpropagation خواهد بود. تفاوت این دو مجموعه فرضیه انتهایی، مجموعه ی فرضیه های غلتی است که backpropagation در نظر گرفته، اما TangentProp آنها را بر اساس دانش قبلی اش رد می کند.



شکل ۱۲.۶ جستجوی فضای فرضیه ای TangentProp.
TangentProp درست مشابه Backpropagation شبکه را مقادیر کوچک تصادفی مقدار دهی اولیه می کند، با این وجود، این الگوریتم از تابع خطای متفاوتی برای هدایت جستجوی شیب نزول استفاده می کند. خطای مورد استفاده ی TangentProp هم خطای مقادیر آموزشی و هم خطای مشتقات آموزشی که توسط دانش قبلی به ما داده می شود را در نظر می گیرد.

توجه دارید که جایگزین دیگر برای متناسب سازی شبکه با مشتقات آموزشی تابع هدف اضافه کردن نمونه های آموزشی جدید نزدیک نمونه های آموزشی قبلی است، این مقادیر را می توان از مشتقات آموزشی و مقادیر آموزشی نمونه ها تخمین زد. برای مثال، در مثال تشخیص تصویر کاراکترها می توان تصاویر را به اندازه های نسبتا کوچک انتقال داد و نمونه های آموزشی جدید بدست آورد و آنرا با همان دسته بندی نمونه ی اصلی به شبکه داد. می توان انتظار داشت که این نمونه های تخمینی با روش Backpropagation به فرضیه ای مشابه فرضیه ی خروجی TangentProp برسد. (Simard 1992) خطای بدست آمده از این دو روش را در چنین حالاتی بررسی کرده و به این نتیجه می رسد که با این حال کارایی TangentProp به نسبت بیشتر از روش تخمینی است. جالب است بدانید که سیستم ALVINN که برای هدایت اتوموبیل طراحی شده بود (به فصل ۴ مراجعه کنید)، روشی مشابه روش تخمین مقادیر آموزشی جدید را به کار برده است. در این سیستم از دانش قبلی اینکه انتقال افقی تصویر متناسب با هدایت فرمان است استفاده کرده تا نمونه های جدیدی تخمین زده و نمونه های آموزشی مشاهده شده را افزایش دهد.

۱۲.٤.٤ الگوریتم EBNN

الگوریتم EBNN یا یادگیری توضیحی شبکه ی عصبی[1] (Mitchell and Thrun 1993a; Thrun 1996) از دو نظر بر پایه ی الگوریتم TangentProp ساخته شده است. ابتدا اینکه به جای اینکه به کاربر برای مشتقات آموزشی وابسته باشد، EBNN خودش مشتقات آموزشی را برای هر نمونه ی مشاهده شده محاسبه می کند. این مشتقات آموزشی بر اساس توضیح هر نمونه ی آموزشی بر اساس تئوری قلمرو موجود محاسبه و از توضیحات استخراج می شوند. دوم اینکه EBNN مشکل چگونگی وزن دهی اهمیت نسبی مولفه های استقرایی و تحلیلی را حل می کند (برای مثال، چگونگی انتخاب پارامتر μ در رابطه ی ۱۲.۱ را تعیین می کند). مقدار μ برای هر یک از نمونه های آموزشی به طور مستقل انتخاب می شود، این میزان بسته به این که دقت تئوری قلمرو در تخمین مقدار آموزشی چقدر است انتخاب می شود. بنابراین مولفه ی تحلیلی یادگیری برای نمونه هایی که توسط تئوری قلمرو درست دسته بندی می شوند موکد و در نمونه هایی که توضیح ضعیف است کمرنگ می شود.

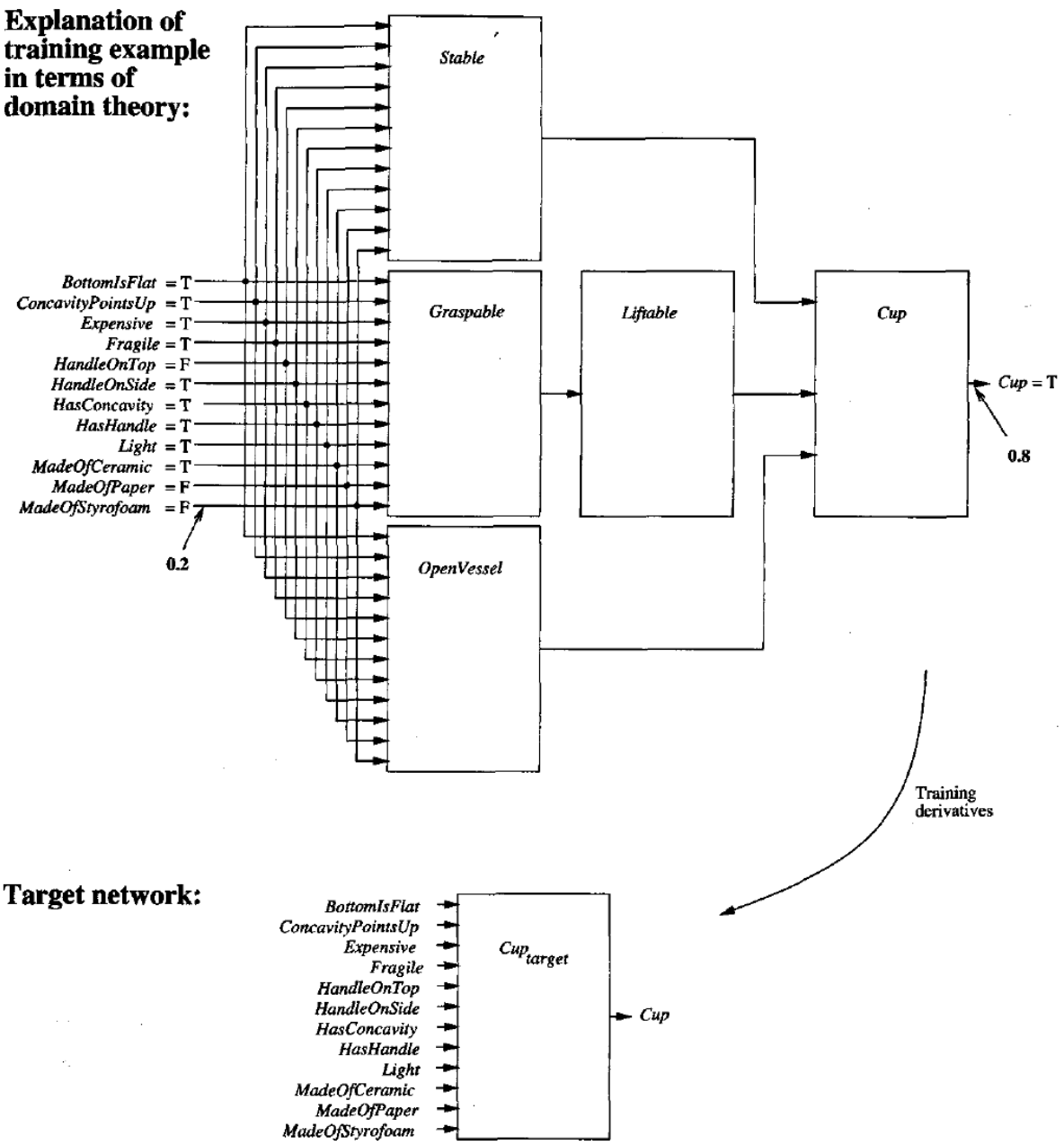
ورودی های الگوریتم EBNN شامل (۱) مجموعه ای از نمونه های آموزشی به فرم $< x_i, f(x_i) >$ می شود که هیچ مشتق آموزشی ای در آن ارائه نشده است، (۲) تئوری قلمرویی مشابه تئوری قلمرو هایی که در یادگیری توضیحی از آن استفاده کردیم (به فصل ۱۱ مراجعه کنید) با این تفاوت که در اینجا تئوری قلمرو به جای دسته ای horn clause ها با شبکه عصبی آموزش یافته ای نمایش داده می شود. خروجی EBNN شبکه ای عصبی است که تابع هدف f را تخمین می زند. این شبکه ی یادگرفته شده هم طوری آموزش دیده است که هم متناسب با نمونه های آموزشی $< x_i, f(x_i) >$ باشد و هم متناسب با مشتقات نتیجه گیری شده از تئوری قلمرو است. متناسب بود با نمونه های آموزشی $< x_i, f(x_i) >$ مولفه ی استقرایی یادگیری و متناسب بودن با مشتقات آموزشی نتیجه گیری شده از تئوری قلمرو مولفه ی تحلیلی یادگیری است.

برای تصور این نوع تئوری قلمرو که EBNN از آن استفاده می کند شکل ۱۲.۷ را در نظر بگیرید. قسمت بالایی این شکل تئوری قلمرویی برای EBNN را نشان می دهد که برای تابع هدف "فنجان" ایجاد شده است، هر مستطیل در این شکل یک شبکه ی عصبی مجزا را در تئوری قلمرو نشان می دهد. توجه دارید که در این مثال برای هر horn clause در قلمرو سمبولیک جدول ۱۲.۳ یک شبکه وجود دارد. برای مثال، شبکه ای به نام Graspable تمامی ویژگی های شبکه را به عنوان ورودی دریافت می کند و خروجی ای متناسب با ویژگی Graspable (در دست جا شدن) می دهد (EBNN گزاره ای درست را با مقدار ۰.۸ مقدار غلت گزاره ای را با ۰.۲ نشان می دهد). این شبکه مشابه Horn clause مربوط به ویژگی Graspable در جدول ۱۲.۳ عمل خواهد کرد. بعضی شبکه ها خروجی دیگر شبکه ها را به عنوان ورودی دریافت می کنند (برای مثال، شبکه ی سمت راست که Cup علامت گذاری شده است از خروجی شبکه های Stable، Liftable و OpenVessel ورودی می گیرد). بنابراین، شبکه ای که تئوری قلمرو را می سازد می تواند مشابه Horn claue ها که گاهی ترکیبی زنجیروار از horn clause ها بودند ترکیبی زنجیر وار از شبکه ها باشد. در کل این تئوری قلمرو ممکن است این اطلاعات توسط منبع خارجی به یادگیر داده شود یا ممکن است این شبکه نتیجه ی یادگیری قبلی همان سیستم باشد. EBNN از این شبکه های تئوری قلمرو برای یادگیری تابع هدف جدید استفاده می کند. EBNN شبکه های تئوری قلمرو را در طی این فرایند تغییر می دهد.

[1] Explanation-based Neural Network learning

Telegram: @Farabar_BI

www.farabar.net



شکل ۱۲.۷ توضیح نمونه ی آموزشی ای در EBNN.

توضیحات از پیشبینی تابع هدف توسط شبکه های تئوری قلمرو (شکل بالا یی) تشکیل یافته است. مشتقات آموزشی از این توضیحات برای آموزش شبکه ی هدف مجزا (شکل پایینی) استفاده می شود. هر بلوک مستطیلی نشان دهنده ی شبکه ی عصبی چند لا یه ای است.

www.farabar.net

هدف EBNN یادگیری شبکه ای جدید است که تابع هدف را توصیف کند. به این شبکه ی جدید شبکه ی هدف[1] می گوییم. در مثال شکل ۱۲.۷ شبکه ی هدف شبکه ی Cup_{target} است که در بالای شکل نشان داده شده و ویژگی های یک جسم را به عنوان ورودی دریافت کرده و مشخص می کند که جسم فنجان است یا خیر.

EBNN با استفاده از الگوریتم TangentProp که در قسمت قبل آمد شبکه ی هدف را یاد می گیرد. از قسمت های پیش می دانید که الگوریتم TangentProp شبکه ای را آموزشی می دهد که هم با مقادیر آموزشی و هم با مشتقات آموزشی سازگار باشد. الگوریتم EBNN مقادیر آموزشی ای $< x_i, f(x_i) >$ را که از ورودی دریافت می کند و به الگوریتم TangentProp می دهد. علاوه بر این، EBNN مقادیر مشتقات آموزشی ای را که بر اساس تئوری قلمرو محاسبه می کند را به الگوریتم TangentProp می دهد. برای درک چگونگی محاسبه ی مشتقات آموزشی، دوباره به شکل ۱۲.۷ توجه کنید. قسمت بالایی این شکل پیشبینی تئوری قلمرو از مقادیر تابع هدف برای نمونه ی آموزشی x_i را نشان می دهد. EBNN مقدار مشتق این پیشبینی را با توجه به ویژگی نمونه ی ورودی محاسبه می کند. برای مثال در این شکل، نمونه ی x_i توسط ویژگی هایش مثل MadeOfStyrofoam = 0.2 (مثلا غلت) و پیشبینی تئوری قلمرو با Cup = 0.8 (مثلا درست) توصیف می شوند. EBNN مشتقات جزئی این پیشبینی را نسبت به ویژگی های نمونه را به صورت مجموعه ی مشتقات زیر محاسبه می کند.

$$\left[\frac{\partial Cup}{\partial BottomIsFlat}, \frac{\partial Cup}{\partial ConcavityPointsUp}, \cdots \frac{\partial Cup}{\partial MadeOfStyrofoam}\right]_{x=x_i}$$

این مجموعه مشتقات گرادیان تابع انتظاری تئوری قلمرو را نسبت به ورودی ها نشان می دهد. زیرنویس $x = x_i$ به این حقیقت اشاره می کند که این مشتقات در نقطه ی $x = x_i$ بررسی می شوند. در حالت کلی تر، زمانی که تابع هدف چندین خروجی دارد، گرادیان تمامی خروجی ها نسبت به تمامی ورودی ها محاسبه خواهد شد. ماتریس حاصل از این عملیات ژاکوبین (Jacobian) تابع هدف نامیده می شود.

برای درک اهمیت این مشتقات آموزشی در کمک به یادگیری شبکه ی هدف، مشتق $\frac{\partial Cup}{\partial Expensive}$ را در نظر بگیرید. اگر تئوری قلمرو بیان کند که ویژگی Expensive تاثیری بر تابع هدف Cup ندارد، مشتق $\frac{\partial Cup}{\partial Expensive}$ که از توضیحات استخراج می شود مقدار صفر خواهد داشت. صفر بودن مشتق بدین معناست که ویژگی Expensive تاثیری بر پیشبینی مقدار Cup ندارد. از طرف دیگر، مشتق بسیار بزرگ مثبت یا منفی بدین معناست که مقدار ویژگی تاثیر بسیار زیادی در تعیین مقدار هدف دارد. بنابراین، مشتقات استخراجی از توضیحات تئوری قلمرو اطلاعات مهمی در تعیین مرتبط یا نا مرتبط بودن ویژگی ها به مقدار هدف را ارائه می کند. زمانی که این مشتقات استخراجی به عنوان مشتقات آموزشی به TangentProp داده می شود تا شبکه ی هدف Cup_{target} را یاد بگیرد، این مشتقات بایاسی مفید برای جهت دهی تعمیم شبکه ارائه می کنند. در چنین شرایطی، بایاس نحوی معمول استقرایی شبکه های عصبی با این بایاس که از مشتقات آموزشی استخراجی از تئوری قلمرو تشکیل یافته جایگزین می شود.

در بالا نحوه ی استفاده از پیشبینی تئوری قلمرو در ایجاد مجموعه ای از مشتقات آموزشی آورده شده است. به عبارت دقیقتر الگوریتم کامل EBNN به صورت زیر می باشد. با داشتن نمونه های آموزشی و تئوری قلمرو، EBNN ابتدا شبکه ای کامل (fully-connected) یک طرفه (feedforward) برای نمایش تابع هدف ایجاد می کند. این شبکه ی هدف، مشابه الگوریتم Backpropagation، با وزن های

[1] target network

کوچک مقدار دهی اولیه می شود. سپس برای هر نمونه ی آموزشی $< x_i, f(x_i) >$، EBNN مشتق آموزشی مربوطه را در فرایندی دو مرحله ای مشخص می کند. ابتدا از تئوری قلمرو برای پیشبینی مقدار تابع هدف برای نمونه ی x_i استفاده می شود. بیایید پیشبینی تابع هدف برای نمونه ی x_i را $A(x_i)$ در نظر بگیریم. به عبارت دیگر، $A(x_i)$ تابعی است که توسط ترکیب شبکه های تئوری قلمرو که برای x_i توضیحی ارائه می کنند تعریف می شود. دوم، وزن ها و توابع فعالیت شبکه های تئوری قلمرو برای استخراج $A(x_i)$ نسبت به تمامی ویژگی های x_i بررسی می شوند (ژاکوبین A(x) در نقطه ی $x = x_i$). استخراج این مشتقات با فرایندی بسیار مشابه محاسبه ی عبارت δ در Backpropagation ادامه می یابد (تمرین ۱۲.۵). بلاخره EBNN از تفاوت جزئی الگوریتم TangentProp استفاده می کند و شبکه هدف را طوری آموزش می دهد که مقدار تابع خطای زیر مینیمم کند.

$$E = \sum_i \left[\left(f(x_i) - \hat{f}(x_i) \right)^2 + \mu_i \sum_j \left(\frac{\partial A(x)}{\partial x^j} - \frac{\partial \hat{f}(x)}{\partial x^j} \right)^2_{(x=x_i)} \right] \qquad (12.2)$$

که در این رابطه داریم

$$\mu_i \equiv \frac{|A(x_i) - f(x_i)|}{c} \qquad (12.3)$$

در اینجا x_i همان i امین نمونه ی آموزشی است و A(x) همان پیشبینی تئوری قلمرو برای ورودی x است. نماد x^j نیز برای نشان دادن j امین مولفه ی بردار x به کار رفته است (برای مثال، j امین گره ورودی شبکه ی عصبی). ثابت c نیز یک ثابت نرمالایز[1] است، این ثابت باعث می شود که مقدار μ_i همیشه $0 \le \mu_i \le 1$.

با وجود اینکه نمایش در اینجا غیرگویا به نظر می رسد، اما ایده ی این عبارت بسیار ساده است. خطای آورده شده در رابطه ی ۱۲.۲ همان فرم کلی تابع خطای رابطه ی ۱۲.۱ (رابطه ی مربوطه ی TangentProp) را دارد. جمله ی اول این تابع خطا، خطای مجموع مربعی بین مقادیر آموزشی $f(x_i)$ و مقدار پیشبینی شبکه ی هدف $\hat{f}(x_i)$ را تاثیر می دهد. جمله ی دوم خطای مربعی بین مقادیر مشتقات آموزشی استخراجی از تئوری قلمرو $\frac{\partial A(x)}{\partial x^j}$ و مقادیر واقعی مشتقات شبکه ی هدف $\frac{\partial \hat{f}(x)}{\partial x^j}$ را نشان می دهد. بنابراین، عبارت سمت چپ نقش قید بخش استقرایی را دارد بدین معنا که فرضیه ی خروجی باید با مقادیر آموزشی مشاهده شده مطابقت داشته باشد، در حالی که جمله ی سمت راست نقش تحلیلی را دارد بدین معنا که فرضیه ی خروجی باید با مشتقات آموزشی استخراجی از تئوری قلمرو مطابقت داشته باشد. توجه دارید که مشتق $\frac{\partial \hat{f}(x)}{\partial x^j}$ در رابطه ی ۱۲.۲ فقط حالت خاصی از رابطه ی $\frac{\partial \hat{f}(s_j(\alpha,x))}{\partial \alpha}$ است که در رابطه ی ۱۲.۱ آمده بود، در این حالت خاص $s_j(\alpha, x)$ تبدیلی است که x_i^j را به $x_i^j + \alpha$ تبدیل می کند، عبارت دقیق تغییر وزن در EBNN در (Thrun 1996) آمده است.

اهمیت نسبی مولفه های استقرایی و تحلیلی یادگیری در EBNN با ثابت μ_i تعیین می شود که در رابطه ی ۱۲.۳ تعریف شده است. مقدار μ_i توسط تفاوت بین پیشبینی تئوری قلمرو $A(x_i)$ و مقدار آموزشی $f(x_i)$ تعیین می شود. بنابراین وزن مولفه ی تحلیلی یادگیری برای نمونه های آموزشی ای که پیشبینی درستی ندارند کمتر در نظر گرفته می شود.این وزن دهی ابتکاری فرض می کند که مشتقات آموزشی استخراجی از تئوری قلمرو در مواردی که تئوری قلمرو مقدار آموزشی را درست پیشبینی می کند درست تر اند. با این وجود، می توان

[1] normalizing constant

وضعیت هایی را بوجود آورد که چنین ابتکاری موفق نباشد، در عمل این روش در چندین تئوری قلمرو موفق از Water در آمده است (Mitchell and Thrun 1993a; Thrun 1996).

۱۲.٤.٥ نکات

به طور خلاصه، الگوریتم EBNN از تئوری قلمرو برای ایجاد مجموعه ای از شبکه های عصبی آموزش دیده استفاده می کند و از آن به همراه نمونه های آموزشی برای آموزش فرضیه ی خروجی اش استفاده می کند. برای هر نمونه ی آموزشی EBNN از تئوری قلمرو برای توضیح نمونه استفاده می کند، سپس مشتق آموزشی را از این توضیح استخراج می کند. برای هر ویژگی نمونه، مشتق آموزشی ای که میزان تاثیر تغییر کوچک ویژگی در مقدار تابع هدف است را از تئوری قلمرو استخراج می کند. این مشتقات آموزشی به نسخه ای از TangentProp که به مشتقات آموزشی و مقادیر آموزشی شبکه ی هدف را تطبیق می دهد داده می شود. تطابق با مشتقات، شبکه ی یادگرفته را ملزم به تطابق وابستگی های موجود در تئوری قلمرو می کند، در حالی که تطابق مقادیر آموزشی شبکه یادگرفته را ملزم به تطابق با خود مقادیر آموزشی می کند. وزن μ_i ضریب مشتقات آموزشی، به طور جداگانه برای هر نمونه ی آموزشی، بر اساس دقت تئوری قلمرو برای پیشبینی مقادیر آموزشی این نمونه تعیین می شود.

نشان داده شده است که EBNN متدی کارا برای یادگیری از تئوری قلمرو های تخمینی در بسیاری از قلمرو هاست. (Thrun 1996) این روش را در یادگیری نسخه های مختلف مسئله ی یادگیری Cup، که در بالا توضیح داده شد، به کار می برد و گزارش می دهد که EBNN دقت بهتری نسبت به Backpropagation، مخصوصا زمانی که تعداد نمونه های آموزشی کم است، دارد. برای مثال، بعد از ۳۰ نمونه ی آموزشی، EBNN به خطای ریشه ی میانگین مربعی[1] ۵.۵ بر روی مجموعه ای مجزا از داده های تست رسید، در حالی که خطای Backpropagation ۱۲.۰ بود. (Mitchell and Thrun 1993a) استفاده از EBNN را در یادگیری کنترل ربات شبیه سازی شده، که تئوری قلمرو اش شبکه های عصبی ای که تاثیرات حرکات ربات را بر وضعیت نشان می دهند به کار می برند. به طور مشابه، EBNN با تخمینی کارایی بهتری نسبت به backpropagation کسب می کند. در اینجا backpropagation حداقل ۹۰ بار حلقه ی آموزش را برای رسیدن به سطحی از دقت تکرار می کند در حالی که EBNN به همان سطح از خطا را در ۲۵ حلقه می رسد. (O'Sullivan et al. 1997) و (Thrun 1996) چندین کاربرد دیگر EBNN را در درک و کنترل رباتهای واقعی را با تئوری قلمرویی از شبکه ها که تاثیر اعمال را برای ربات در محیط بسته با میکروفن[2]، دید و سنسورهای لیزری تعیین می شود، را توصیف می کنند.

EBNN رابطه ی جالبی با دیگر متد های یادگیری توضیحی، مثل Prolog-EBG که در فصل ۱۱ توضیح داده شده، دارد. با توجه به آنچه در آنجا گفته شده، Prolog-EBG نیز توضیحاتی (پیشبینی مقادیر هدف نمونه) بر اساس تئوری قلمرو ایجاد می کند. در Prolog-EBG توضیحات با استفاده از تئوری قلمرویی متشکل از horn clause ها و فرضیه ی هدف با محاسبه ی ضعیفترین پیشفرضی که در آن توضیحات درست باشند بازبینی می شود. بنابراین وابستگی های نسبی در این توضیحات با فرضیه های horn clause های یادگرفته شده بیان می شود. EBNN نیز توضیحات مشابهی ایجاد می کند، اما توضیحات EBNN بر پایه ی تئوری قلمرویی از شبکه های عصبی، به جای horn clause ها، است. مشابه Prolog-EBG، وابستگی های نسبی از این توضیحات استخراج شده و برای بازبینی فرضیه ی هدف به کار می رود. در EBNN این وابستگی ها به فرم مشتقاتی بیان می شود زیرا که مشتقات نمایش طبیعی وابستگی در توابع پیوسته، مشابه شبکه های

---

[1] root-mean-squared
[2] sonar

عصبی است. در مقابل، نمایش طبیعی وابستگی ها در توضیحات نمادین یا اثباتهای منطقی توصیف مجموعه ای از نمونه هایی است که این اثبات برایشان صادق است.

تفاوت های بسیاری بین قابلیت های EBNN و متدهای نمادین یادگیری فصل ۱۱ وجود دارد. تفاوت اصلی در این است که EBNN از تئوری قلمرو های ناکامل استفاده می کند، در حالی که Prolog-EBG از تئوری قلمروهای کامل استفاده می کند. این تفاوت از این حقیقت که EBNN بر پایه ی مکانیستم استقرایی تطابق با مقادیر آموزشی مشاهده شده و استفاده از تئوری قلمرو فقط به عنوان قید اضافی ای بر روی فرضیه ی یادگرفته شده ساخته شده ناشی شده است. تفاوت مهم دوم از این حقیقت ناشی می شود که Prolog-EBG دسته ای افزایشی از horn clause ها را یاد می گیرد در حالی که EBNN از شبکه ی عصبی ای با اندازه ی ثابت استفاده می کند. همانطور که در فصل ۱۱ نیز گفته شد، یکی از مشکلات یادگیری دسته قوانین horn clause این است که هزینه ی دسته بندی نمونه های جدید با ادامه ی فرایند یادگیری و افزایش horn clause های جدید افزایش می یابد. این مشکل در EBNN وجود ندارد زیرا که شبکه ی هدف با اندازه ی ثابت زمان ثابتی برای دسته بندی نمونه نیاز خواهد داشت. با این وجود، شبکه ی عصبی با اندازه ی ثابت در مقابل مشکلاتی دارد، زیرا که ممکن است نتواند توابع به اندازه ی کافی پیچیده را نشان دهد، در حالی که دسته ای از horn clause ها می توانند با افزایش تعداد هر تابع پیچیده ای را نمایش دهند. (Mitchell and Thrun 1993b) بحث دقیقتری از رابطه ی EBNN و متدهای یادگیری توضیحی نمادین انجام می دهد.

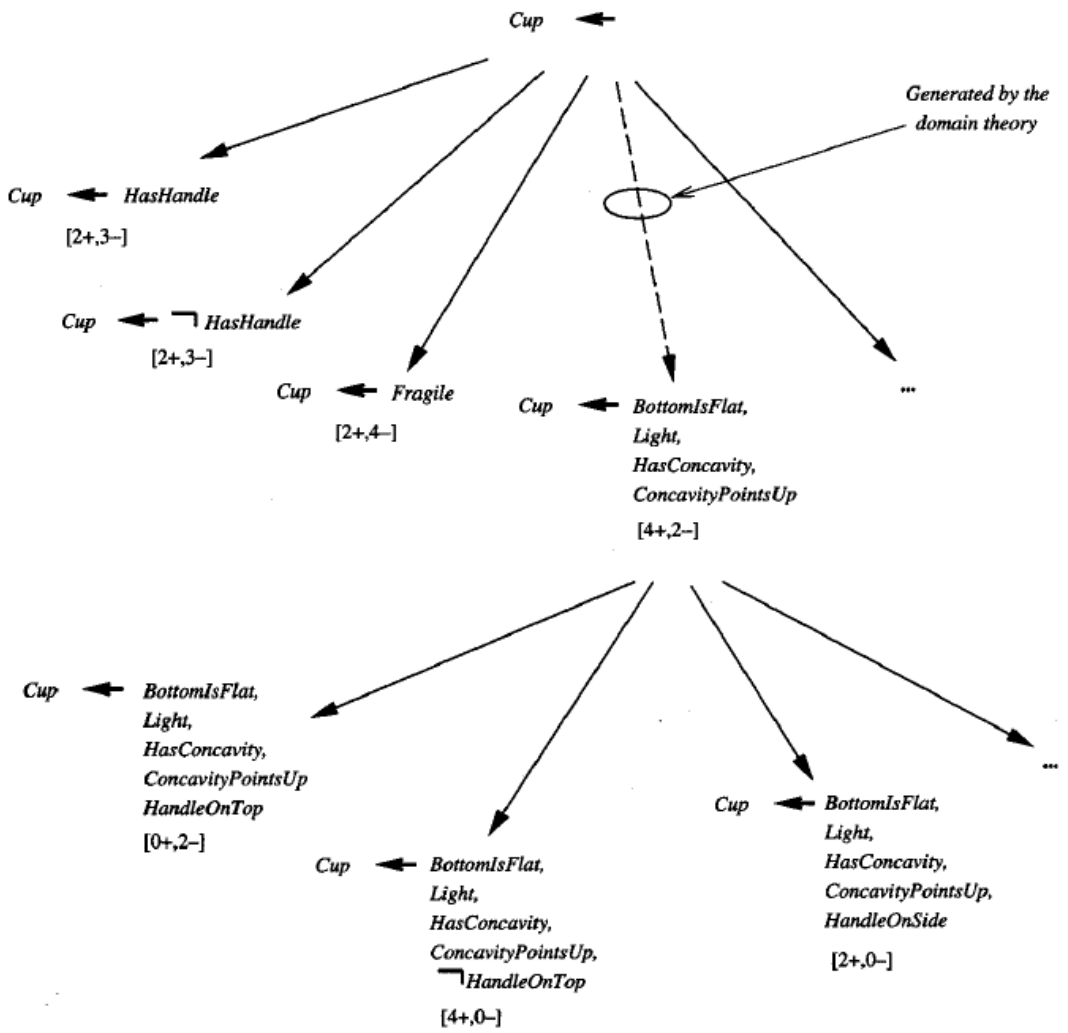
۱۲.۵ استفاده از دانش قبلی برای تغییر عملگر های جستجو

در دو قسمت گذشته دو نقش دانش قبلی در یادگیری را مطرح کردیم: مقدار دهی اولیه ی فرضیه ی یادگیر و تغییر تابع هدفی که جستجوی فضای فرضیه ای را کنترل می کند. در این بخش، روش سومی برای استفاده از دانش قبلی برای تغییر جستجوی فضای فرضیه ای را مورد بحث قرار خواهیم داد: استفاده از دانش قبلی برای تغییر مجموعه ی عملگر هایی که مراحل قانونی در جستجوی فضای فرضیه ای را تعیین می کنند. این روش در الگوریتم های FOCL، (Pazzani 1991; Pazzani and Kibler 1992) و ML-SMART (Bergando and Giordana 1990) استفاده شده است. ما در اینجا از الگوریتم FOCL برای تصور روش استفاده خواهیم کرد.

۱۲.۵.۱ الگوریتم FOCL

FOCL تعمیمی از سیستم استقرایی محض FOIL است که در فصل ۱۰ به طور کامل به آن پرداختیم. هر دو سیستم FOIL و FOCL مجموعه ای از horn clause های درجه اول برای پوشاندن نمونه های مشاهده شده ایجاد می کند. هر دو سیستم از الگوریتمی ترتیبی که ابتدا یک قانون یاد می گیرد و تمامی نمونه های مثبت پوشانده شده با آن را حذف می کند و سپس فرایند را برای نمونه های باقی مانده تکرار می کند استفاده می کنند. در هر دو سیستم، هر horn clause جدید با جستجویی کلی به جزئی ساخته می شود، این جستجو از کلی ترین horn clause ممکن آغاز می شود (مثلا horn clause ی که هیچ شرطی ندارد). سپس چندین جزئی سازی ممکن این horn clause ساخته می شود و جزئی سازی ای که بیشترین بهره ی اطلاعات را بر روی نمونه های آموزشی دارد انتخاب می شود. این فرایند تکرار می شود تا جزئی سازی های بیشتری ایجاد شود و باز هم بهترین آنها انتخاب شده تا اینکه به horn clause ی برسد که کارایی رضایت بخش را داشته باشد.

تفاوت FOIL و FOCL در نحوه ی ایجاد خاص سازی ها در طی جستجوی کلی به جزئی برای یافتن یک horn clause است. همانطور که در فصل ۱۰ نیز گفته شد، FOIL هر جزئی سازی را با اضافه کردن یک عبارت جدید به شروط horn clause می سازد. FOCL نیز از همین روش برای ایجاد خاص سازی های ممکن استفاده می کند اما علاوه بر آنها خاص سازی های مبتنی بر تئوری قلمرو را نیز در نظر می گیر. یال های پر رنگ درخت جستجو در شکل ۱۲.۸ مراحل جستجو ی کلی به جزئی در نظر گرفته شده در جستجوی FOIL را نشان می دهند. یال های خط چین در این درخت جستجو (شکل ۱۲.۸) خاص سازی های اضافی در نظر گرفته شده توسط FOCL و مبتنی بر تئوری قلمرو هستند.



شکل ۱۲.۸ جستجوی فضای فرضیه ای FOCL.

برای یادگیری یک قانون، FOCL فرضیه های ممکن را کلی به جزئی و به صورت افزایشی مورد بررسی قرار می دهد. دو نوع عملگر خاص سازی فرضیه ی فعلی را انجام می دهند. یکی از این دو نوع تنها یک عبارت جدید به قانون اضافه می کند (خطوط متوسط شکل). عملگر نوع دوم خاص سازی قانون فعلی را با اضافه کردن مجموعه ای از عبارات که به طور منطقی شرط کافی تابع هدف بر اساس تئوری قلمرو است را در نظر می گیرد (خطوط خط چین شکل). FOCL از میان این خاص سازی ها، با معیار کارایی شان بر روی داده ها خاص سازی ای انتخاب می کند. بنابراین تئوری قلمرو های ناکامل فقط زمانی که مدارک تاییدی داشته باشند تاثیر خواهند گذاشت. این مثال بر پایه داده ها و تئوری قلمروی استفاده شده در KBANN آورده شده است.

با وجود اینکه FOCL و FOIL هر دو horn clause های درجه اول را یاد می گیرند اما در اینجا بحث را به قوانین گزاره ای درجه اول یا همان horn clause های بدون متغیر محدود می کنیم. دوباره مفهوم هدف Cup، نمونه های آموزشی مربوطه و تئوری قلمرو شکل ۱۲.۳ را در نظر بگیرید. برای توصیف عملیات FOCL، ابتدا باید فرق بین عبارات آمده در تئوری قلمرو و فرضیه ها را مشخص کنیم. زمانی می گوییم یک عبارت عملیاتی (operational) است که اجازه داشته باشیم آنرا در توصیف یک فرضیه ی خروجی بکار ببریم. برای مثال در مثال Cup شکل ۱۲.۳ ما فقط اجازه داریم که از ۱۲ ویژگی ای که در نمونه های آموزشی آمده (مثل HasHandle، HandleOnTop) عبارات مبتنی بر این ۱۲ ویژگی بنابراین عبارات عملیاتی شمرده می شوند. در مقابل، عباراتی که مبتنی بر ویژگی واسطه ی میانی در تئوری قلمرو اند و مبتنی بر ویژگی اولیه ی نمونه ها نیستند عبارات غیر عملیاتی به شمار می آیند. نمونه ای از ویژگی های غیر عملیاتی در این مثال ویژگی Stable است.

در هر مرحله از جستجو ی کلی به جزئی FOCL فرضیه ی فعلی خود h را با دو عملگر زیر گسترش می دهد:

برای هر عبارت عملیاتی که جزو h نیست، یک جزئی سازی از قانون h با اضافه کردن این تک عبارت به شروط قانون بساز. FOIL نیز از این متد برای ایجاد جزئی سازی ممکن استفاده می کند. فلش های پر رنگ شکل ۱۲.۸ این نوع جزئی سازی را نشان می دهد.

مطابق با تئوری قلمرو شرطی منطقی و عملیاتی برای تابع هدف بساز. این عبارت جدید را به شروط قانون h اضافه کن و بلاخره با حذف شروط اضافی (برای پوشش نمونه ها) h را هرس کن. فلش خط چین در شکل ۱۲.۸ چنین جزئی سازی ای را نشان می دهد.

جزئیات عملگر دوم به شرح زیر است. FOCL ابتدا یکی از قوانین تئوری قلمرو که حکمش با تابع هدف یکی است را انتخاب می کند. اگر چندین قانون چنین حالتی داشته باشد قانونی انتخاب خواهد شد که بهره ی اطلاعات بیشتری بر روی تابع هدف دارد. برای مثال، با تئوری قلمرو و داده های جدول ۱۲.۳ فقط یک قانون چنین حالتی خواهد داشت:

Cup←Stable,Liftable,OpenVessel

شروط قانون انتخابی شرطی منطقی کافی برای تابع هدف را تشکیل می دهد. هر عبارت غیر عملیاتی در این شرط کافی با استفاده از تئوری قلمرو جایگزین می شود، بدین معنا که حکم قوانین با شروطشان جایگزین می شود. برای مثال، قانون Stable←BottomIsFlat معادل جایگزین کردن عبارت عملیاتی BottomIsFlat به جای عبارت غیر عملیاتی Stable استفاده می شود. این فرایند بازکردن[1] تئوری قلمرو تا جایی که شرط کافی با عبارات عملیاتی بیان شود ادامه پیدا می کند. اگر چندین قانون از تئوری قلمرو برای یک عبارت توضیح آورده باشند انکه بهره ی اطلاعات بیشتری دارد برای جایگزینی استفاده می شود. واضح است که شرط کافی با داده ها و تئوری قلمرو موجود در مثال Cup به صورت زیر است:

BottomIsFlat, HasHandle, Light, HasConcavity, ConcavityPointsUp

در مرحله ی آخر در ایجاد خاص سازی ممکن، این شرط کافی هرس می شود. بدین صورت که هر عبارتی که حذفش باعث کاهش دقت دسته بندی بر روی نمونه های آموزشی نمی شود حذف خواهد شد. این مرحله برای تشخیص جزئی سازی بیش از حد[2] ایجاد شده است زیرا که

[1] unfolding
[2] overspecialization

تئوری قلمرو های ناکامل ممکن است عبارات نامربوط در بر داشته باشند. در مثال حاضر، حذف عبارت HasHandle باعث افزایش کارایی می شود. بنابراین شرط کافی به صورت زیر خواهد بود:

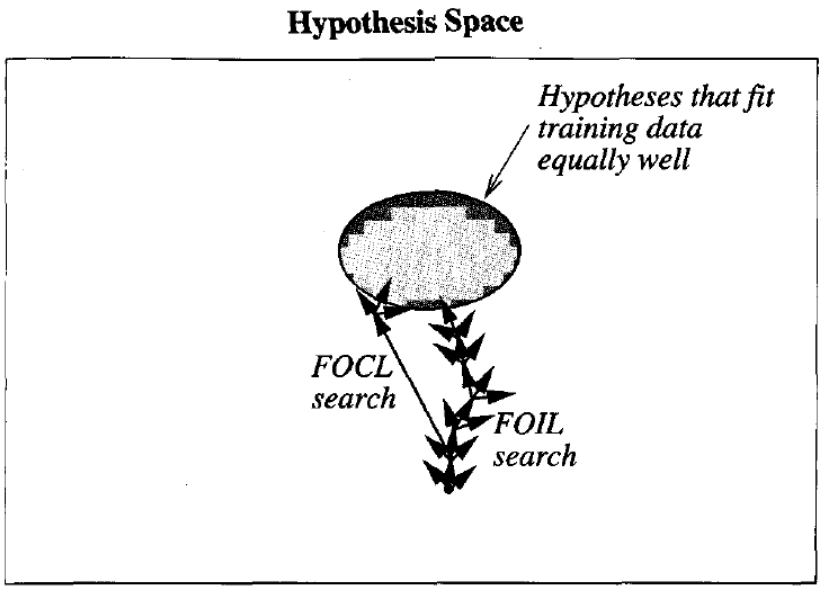
BottomIsFlat, Light, HasConcavity, ConcavityPointsUp

حال این مجموعه از عبارات به شروط فرضیه فعلی اضافه می شود. توجه داشته باشید که فرضیه حاصل با فلش خط چین در شکل ۱۲.۸ نشان داده شده است.

یکی از جزئی سازی های فرضیه ی فعلی با استفاده از دو عملگر بالا ایجاد شده است. بین این فرضیه ها فرضیه ای که بیشترین بهره ی اطلاعات را داشته باشد انتخاب خواهد شد. در مثال نشان داده شده در شکل ۱۲.۸ جزئی سازی انتخاب شده در مرحله ی اول درخت جستجو همان خاص سازی حاصل از تئوری قلمرو است. در ادامه ی جستجو سیستم به دنبال جزئی سازی بیشتر شروط تئوری قلمرو می گردد. این قسمت از جستجو قسمت استقرایی جستجو است که بازنگری در شروط اشتقاقی از تئوری قلمرو را ممکن می سازد. در مثال حاضر، تئوری قلمرو فقط در مرحله ی اول جستجو تاثیر می گذارد. با این وجود همیشه این اتفاق نمی افتد. اگر تئوری قلمرو به اندازه ی کافی کارا نباشد در مرحله ی اول از روش استقرایی استفاده می شود و تاثیر تئوری قلمرو به مراحل بعدی موکول می شود . به طور خلاصه اینکه FOCL، horn clause ها را به فرم زیر یاد می گیرد،

$$c \leftarrow o_i \wedge o_b \wedge o_f$$

در این رابطه c مفهوم هدف است و o_i عطفی اولیه از عبارات عملیاتی است که در مرحله به مرحله توسط عملگر اول به قانون اضافه شده است (در مراحل قبلی)، o_b عطفی از عبارات عملیاتی است که در یک مرحله بر اساس تئوری قلمرو اضافه می شود و o_f نیز عطفی از عبارات عملیاتی است که در یک مرحله توسط عملگر اول اضافه می شود. هر یک از این سه مجموعه ممکن است تهی باشد.



شکل ۱۲.۹ جستجوی فضای فرضیه ای FOCL.
FOCL مجموعه ای عملگر های جستجوی FOIL را تغییر می دهد. با وجود اینکه FOIL فقط اضافه کردن یک عبارت را در هر مرحله در نظر می گیرد، اما FOCL اضافه کردن چندین عبارت استخراجی از تئوری قلمرو در یک مرحله را نیز در نظر می گیرد.

بحث بالا استفاده از تئوری قلمرو گزاره ای در ایجاد خاص سازی های ممکن فرضیه در طی جستجوی کلی به جزئی برای پیدا کردن horn clause مناسب را توصیف می کند. این الگوریتم می تواند به راحتی برای نمایش درجه اول تعمیم داده شود (قوانینی که متغیر نیز دارند). فصل ۱۰ به طور کامل الگوریتم FOIL و نحوه ی تعمیم ایجاد horn clause های درجه اول بدون متغیر به ایجاد horn clause های درجه اول با متغیر را توضیح داده است. برای تعمیم عملگر دوم برای تطبیق با تئوری قلمرو هایی با horn clause های درجه اول، در بازکردن تئوری قلمرو باید متغیر ها را در نظر گرفت. این کار را می توان با روش فرایند regression در جدول ۱۱.۳ انجام داد.

۱۲.۵.۲ نکات

FOCL از تئوری قلمرو برای افزایش جزئی سازی های ممکن در نظر گرفته شده در هر مرحله از جستجوی horn clause ها استفاده می کند. شکل ۱۲.۹ جستجوی فضای فرضیه ای الگوریتم FOCL و الگوریتم استقرایی محض FOIL را مقایسه می کند. جزئی سازی متناسب با تئوری قلمرو در FOCL مشابه تغییرات بزرگ (macro) در جستجوی FOIL که در آن چندین عبارت همزمان و در یک گام به قانون اضافه می شود. این فرایند را می توان به دید ترفیع فرضیه ای که ممکن است در ادامه ی جستجو مورد بررسی قرار گیرد در نظر گرفت ---. اگر تئوری قلمرو درست باشد، داده های آموزشی نیز مطابق با آن خواهد بود، پس این خاص سازی انتخاب خواهد شد. حال اگر تئوری قلمرو درست نباشد، ارزیابی تمامی فرضیه ها صورت گرفته و مسیر دیگری برای ادامه ی جستجو انتخاب خواهد شد.

به طور خلاصه FOCL از هر دو روش ایجاد جزئی سازی ممکن با روش استقرایی و مبتنی بر تئوری قلمرو در هر مرحله از جستجو استفاده می کند. الگوریتم بین جزئی سازی های ممکن فقط بر اساس معیاری که بر اساس تجربه (نمونه های آموزشی) است انتخاب می کند. بنابراین تئوری قلمرو یادگیر را بایاس می کند اما انتخاب بین این بایاس و سیستم استقرایی را بر عهده ی تجربه (روش استقرایی) می گذارد. بایاسی تئوری قلمرو ایجاد می کند ترجیح انتخاب horn clause های مشابه horn clause های عملیاتی، منطقی و کافی استخراجی از تئوری قلمرو است. این بایاس با بایاس برنامه ی استقرایی محض FOIL، که ترجیح فرضیه های کوتاهتر است، ترکیب شده و بایاس این سیستم را ایجاد می کند.

FOCL نشان داده است که در تعدادی از کاربرد ها که تئوری قلمرو ناکامل در دسترس است دقت بهتری در تعمیم نسبت به الگوریتم استقرایی محض FOIL دارد. برای مثال، (Pazzani and Kibler 1992) با این روش مفهوم "چینش های مجاز صفحه ی شطرنج" را بررسی کردند. با مجموعه ای ۶۰ تایی از نمونه های آموزشی که حاوی ۳۰ نمونه ی مجاز و ۳۰ نمونه ی غیرمجاز بود، FOIL ۸۶ درصد روی مجموعه ی مجزایی کارایی داشت. در حالی که FOCL با همان نمونه های آموزشی و تئوری قلمرویی با ۷۶ کارایی، ۹۴ درصد کارایی نهایی داشت، که این خطای مقدار از نصف میزان خطای FOIL کمتر است. برای مثال، با مجموعه ای از ۵۰۰ نمونه ی آموزشی خرابی تلفن از شرکت NYNEX، FOIL کارایی ۹۰ درصد داشت درحالی که FOCL با تئوری قلمرویی با کارایی ۹۵ درصد به کارایی نهاییی ۹۸ درصد روی همان مجموعه ی تست رسید.

۱۲.۶ آخرین پیشرفت ها

متد های معرفی شده در این فصل فقط روش های ساده ی ممکن در ترکیب یادگیری تحلیلی و استقرایی است. در حالی که هر یک از این متد ها نشان داده اند که کارایی بهتری نسبت به روش های استقرایی محض دارند، اما هیچ یک از این روش ها در روی مجموعه ی متنوعی از

قلمرو ها[1] تست نشده یا کارایی خوبی نداشته اند. بحث ترکیب روش های استقرایی و تحلیلی هنوز یکی از بحث های قابل تحقیق و بررسی است.

۱۲.۷ خلاصه و منابع برای مطالعه بیشتر

نکات اصلی این فصل شامل موارد زیر می باشد:

دانش قبلی تقریبی، یا تئوری قلمرو ها در بسیاری از مسائل یادگیری عملی در دسترس اند. روش های استقرایی محض مثل درخت تصمیم و شبکه های عصبی از چنین تئوری قلمروهایی استفاده نمی کنند و بنابرانی فقط زمانی که مقدار داده های به اندازه کافی زیاد باشد درست عمل می کنند. روشهای تحلیلی محض مثل Prolog-EBG از تئوری قلمرو ها استفاده کرده اما فرضیه های غلتی با داشتن دانش قبلی ناکامل ایجاد می کنند. متدهایی که مخلوطی از یادگیری استقرایی و تحلیلی اند می توانند از مزیت هر دو متد بهره ببرند: کاهش پیچیدگی نمونه ای و قدرت تصحیح دانش قبلی نادرست.

یکی از روشها، الگوریتم های ترکیب یادگیری استقرایی و تحلیلی با توجه به تاثیر تئوری قلمرو بر جستجوی فضای فرضیه ای است. در این فصل به متدهایی پرداختیم که از تئوری قلمروی ناکامل برای (۱) ایجاد فرضیه اولیه ی جستجو استفاده می کند، (۲) گسترش مجموعه ی عملگر های جستجو که بازنگری روی فرضیه فعلی می کنند استفاده می کنند و (۳) هدف جستجو را تغییر می دهند انجام دادیم.

یکی از سیستم هایی که از تئوری قلمرو برای مقدار دهی اولیه ی فرضیه استفاده می کند الگوریتم KBANN است. این الگوریتم از تئوری قلمرو بیان شده با دسته قوانین گزاره ای تحلیلی برای ایجاد شبکه ی عصبی اولیه که معادل تئوری قلمرو باشد استفاده می کند. سپس این تئوری قلمرو به صورت استقرایی توسط الگورتیم Backpropagation، برای بهبود کارایی روی داده های آموزشی بازبینی می شود. نتیجه ی حاصل شبکه ای بایاس شده توسط تئوری قلمروی ابتدایی خواهد بود که وزنهایش با روش استقرایی روی داده های آموزشی بازبینی شده اند.

TangentProp از دانش قبلی نمایش داده شده با مشتقات تابع هدف نیز استفاده می کند. در بعضی زمینه ها، مثل پردازش تصویر، این روش، روشی عادی برای نشان دادن دانش قبلی است. TangentProp این دانش قبلی را با تغییر هدف جستجوی شیب نزول در فضای فرضیه تاثیر می دهد.

EBNN از تئوری قلمرو برای تغییر هدف جستجو در فضای وزنهای ممکن یک شبکه ی عصبی استفاده می کند. این روش تئوری قلمرو به شکل شبکه های عصبی یادگرفته شده ی پیشین معادل برای ایجاد شبکه ی عصبی ---. ---

FOCL از تئوری قلمرو برای افزایش اعضا مجموعه ی فرضیه های ممکن در نظر گرفته شده در هر مرحله از جستجو استفاده می کند. این روش از تئوری قلمروی تقریبی نمایش داده شده با horn clause ها برای یادگیری مجموعه ای از horn clause ها که تابع هدف را تخمین می زنند استفاده می کند. FOCL از الگوریتمی پوشش ترتیبی استفاده کرده و برای یادگیری هر horn clause از جستجویی کلی به جزئی استفاده می کند. تئوری قلمرو برای تغییر مجموعه ی فرضیه های خاص تر ممکن در نظر گرفته شده در هر مرحله از این جستجو به کار می رود. فرضیه های ممکن سپس بر اساس کاراییشان روی داده های آموزشی

[1] domain

سنجیده می شوند. با این روش، FOCL جستجوی حریصانه، کلی به جزئی و استقرایی FOIL را با زنجیر قانون (rule-chaining) و استدلال تحلیلی را ترکیب می کند.

اینکه چگونه به بهترین وجه ممکن دانش را با مشاهدات جدید ترکیب کنیم همچنان یکی از سوالات اصلی یادگیری ماشین باقی مانده است.

الگوریتم های بسیاری وجود دارند که هدفشان ترکیب یادگیری استقرایی و یادگیری تحلیلی است. برای مثال، متدهای یادگیری شبکه های باور بیزی که در فصل ۶ به آن پرداختیم روش دیگری برای بحث در اینجا خواهند بود---. منابع این فصل نمونه ها و منابع یادگیری بیشتر را دربر دارد.

تمرینات

۱۲.۱ مسئله ی یادگیری مفهوم GoodCreditRisk را بر روی نمونه های توصیفی با چهار ویژگی HasStudentLoan، HasSavingsAccount، IsStudent و OwnsCar را در نظر بگیرید. شبکه ی اولیه ی الگوریتم KBANN را شبکه ی حاصل از تئوری قلمروی زیر با تمامی ارتباط ها در نظر بگیرید.

GoodCredit←Employed, LowDept

Employed←¬IsStudent

LowDebt←¬HasStudentLoan,HasSavingsAccount

۱۲.۲ KBANN دسته ای از horn clause های گزاره ای را دریافت کرده و انرا به شبکه ی عصبی اولیه تبدیل می کند. کلاس قوانین گزاره ای "n از m" را در نظر بگیرید، این قوانین m عبارت شرط دارند و پارامتر مربوطه ی n را دربر می گیرند $(n \leq m)$. زمانی شرط قانون گزاره ای "n از m" که حداقل n گزاره از m گزاره ی شرط آن درست باشند. برای مثال،

Student ← LivesInDorm,Young,Studies; n=2

نشان می دهد که زمانی مفهوم Student درست است که حداقل دو گزاره از سه گزاره ی شرط بالا درست باشد.

الگوریتمی مشابه الگوریتم KBANN طراحی کنید که دسته ای از قوانین "n از m" را دریافت کرده و شبکه ای عصبی سازگار با تئوری قلمرو ایجاد کند.

۱۲.۳ تعمیم الگوریتم KBANN را برای دریافت تئوری قلمرویی که شامل قوانین درجه اول به جای horn clause هاست را در نظر بگیرید (horn clause هایی که متغیر نیز دارند، مشابه فصل ۱۰). اگر چنین کاری ممکن است الگوریتمی برای ساخت شبکه های عصبی معادل قوانین درجه اول ارائه کنید و اگر ممکن نیست مشکلات بازدارنده را بیان کنید.

۱۲.۴ این تمرین از شما می خواهد که شیب نزول را مشابه انچه در TangentProp استفاده شد استخراج کنید. فضای نمونه ای X را شامل اعداد حقیقی در نظر بگیرید، فضای فرضیه ای H را نیز توابع درجه دو روی x در نظر بگیرید. یعنی،

$$h(x) = w_0 + w_1 x + w_2 x^2$$

(a) قانون شیب نزول را برای مینیمم کردن معیار Backpropagation به کار ببرید؛ معیار Backpropagation مینیمم کردن مجموع خطای مربعی بین فرضیه و داده های آموزشی است.

(b) قانون شیب نزول دیگری که همان معیار را برای TangentProp مینیمم می کند را استخراج کنید. فقط تبـدیل s(α,x)=x+ α را در نظر بگیرید.

۱۲.۵ EBNN مشتقات آموزشی را از توضیحات حاصل از بررسی وزنها و تحریک شبکه های عصبی ساخته شده برای توضیح بدست مـی آورد. نمونه ی ساده ای را که در آن توضیحات با یک سیگموید با n ورودی ایجاد می شود را در نظر بگیریـد. رابطـه ای بـرای پیـدا کـردن مشـتق آموزشی $\frac{\partial \hat{f}(x)}{\partial x^j}|_{x=x_i}$ پیدا کنید، در این رابطه x_i یک نمونه ی آموزشی خاص ورودی به واحد، $\hat{f}(x)$ خروجی واحد سیگموید و x^j نشـان دهنده ی j امین ورودی واحد سیگموید است. می توانید از نماد گذاری x_i^j برای نشان دادن j امین ویژگی نمونه ی x_i استفاده کنید. راهنمایی: حل این مسئله مشابه استخراج قانون یادگیری برای backpropagation است.

۱۲.۶ دوباره مسیر یادگیری الگوریتم FOCL که در شکل ۱۲.۸ نشان داده شد را در نظر بگیرید. فرض کنید که فرضیه ی اولیه ی انتخاب شده فرضیه ی زیر باشد:

Cup←¬HasHandle

فرضیه های مرحله ی دوم ای که FOCL ایجاد می کند را پیدا کنید. فقط کافی است فرضیه های ایجـاد شـده توسـط عملگـر دوم جسـتجوی FOCL را پیدا کنید که از تئوری قلمرو استفاده می کند. فراموش نکنید که شروط کافی باید هرس شوند. از داده های جدول ۱۲.۳ برای آموزش استفاده کنید.

۱۲.۷ این فصل سه روش برای استفاده از دانش قبلی برای تاثیر بر جستجو فضای فرضیه های ممکن ارائه می کنـد. ایـده هـای خـود را بـرای اینکه چگونه می توان این سه روش را کامل کرد ارائه کنید. ایا می توانید الگوریتم خاصی ارائـه کنیـد کـه دو روش از ایـن سـه روش را بـرای نمایش فرضیه ای خاصی به کار ببرد؟ مزیت ها و مزرت های این ترکیب چه خواهد بود؟

۱۲.۸ دوباره سوال مطرح شده در قسمت ۱۲.۲.۱ را در نظر بگیرید. چه معیاری برای انتخاب میان فرضیه ها زمانی که هم داده و هم دانش قبلی در دسترس است استفاده شود؟ در این باره دیدگاه خود را بیان کنید.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

Telegram: @Farabar_BI
www.farabar.net

فصل سیزدهم: یادگیری تقویتی

یادگیری تقویتی به مسئله‌هایی می‌پردازد که در آن یک عامل مستقل حالت‌هایی را درک کرده و مطابق با آن ادراک اعمال بهینه ای را برای رسیدن به اهدافش را انجام می‌دهد. این مسئله‌ی بسیار جامع است و شامل مسائل یادگیری کنترل ربات‌های متحرک، یادگیری بهینه سازی کارخانه‌ها، و یادگیری بازی‌های صفحه ای[1] می‌شود. هر گاه که عامل عملی را در محیطش انجام می‌دهد، یک معلم متناسب با حالت و عمل انجام شده به وی پاداش می‌دهد یا وی را تنبیه می‌کند (پاداش منفی). برای مثال، معلم ممکن است در یک بازی برای برد پاداش مثبت و برای باخت پاداش منفی و برای اعمال دیگر پاداش صفر را در نظر بگیرد. کار عامل، یادگیری از این پاداش‌ها (که گاهاً تأخیر نیز دارد) است تا در اعمال بعدی بیشترین میزان تابع تجمعی پاداش را بگیرد. در این فصل بیشتر بر روی الگوریتمی به نام یادگیری Q که پاداش‌های تأخیری را نیز در نظر می‌گیرد تمرکز می‌کنیم. این الگوریتم حتی زمانی که عامل هیچ اطلاعاتی در مورد محیط ندارد درست کار می‌کند. الگوریتم‌های یادگیری تقویتی رابطه‌ی نزدیکی با الگوریتم‌های برنامه نویسی پویا[2] دارند، که کاربرد بسیاری در مسائل بهینه سازی دارد.

۱۳.۱ معرفی

فرض کنید که رباتی یادگیر داریم. این ربات، یا عامل[3]، دسته حسگری برای مشاهده‌ی حالت[4] محیط دارد، و دسته ای از اعمال[5] را می‌تواند برای تغییر حالت انجام دهد. برای مثال، یک ربات متحرک ممکن است حسگرهایی چون دوربین و اعمالی چون "حرکت به سمت جلو" و

[1] board games
[2] dynamic programming
[3] agent
[4] state
[5] action

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

"چرخش" داشته باشد. هدف این ربات یادگیری خط مشی[1] یا متدی برای کنترل اعمال است که بتواند با استفاده از آن‌ها به اهداف خود برسد. برای مثال، ممکن است هدف ربات اتصال به شارژر در هنگام کمبود شارژ باشد.

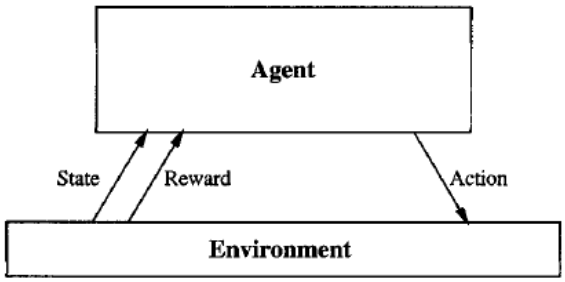
در این فصل به چگونگی یادگیری استراتژی کنترل بهینه این عامل‌ها می‌پردازیم. فرض می‌کنیم که می‌توان اهداف عامل را با تابعی حقیقی مقدار به نام "تابع پاداش"[2] مشخص کرد. این تابع به هر عمل در هر حالت عددی را نسبت می‌دهد. مثلاً برای مثال، برای اتصال به شارژر در حالتی شارژ باتری کم می‌تواند مقدار مثبتی (مثلاً 100+) و در برای اتصال به شارژر و در بقیه‌ی حالت‌ها مقدار صفر داشته باشد. تابع پاداش را می‌توان به در قسمتی از ربات و یا عاملی خارجی مثل معلمی که برای اعمال پاداش می‌دهد تعریف کرد. کار ربات انجام سری اعمالی و مشاهده‌ی نتیجه‌ی آن‌ها در محیط و یادگیری استراتژی کنترل است. استراتژی کنترل این است که در هر حالت اولیه ربات بتواند اعمالی را انتخاب کند تا به بیشترین پاداش برسد. این تعریف به طور خلاصه در شکل ۱۳.۱ آورده شده است.

همان طور که در شکل ۱۳.۱ نیز پیداست، مسئله‌ی یادگیری خط مشی‌ای برای حداکثر کردن پاداش تجمعی[3]، خیلی کلی است و مسائلی خارج از محدوده‌ی یادگیری ربات را نیز در بر می‌گیرد. در کل، مسئله یادگیری کنترل سری اعمال، در هر حالت است. این مسئله ممکن است، برای مثال، مسئله‌ی بهینه کردن اعمال یک تولید کننده برای حداکثر کردن سود کارخانه باشد، در این مثال تابع پاداش می‌تواند قیمت کالاهای تولید شده منهای مبالغ مصرفی باشد. یا ممکن است مسئله این باشد که شرکت تاکسی بی سیم برنامه‌ی تاکسی‌های خود را چگونه در یک شهر بزرگ برنامه ریزی کند و تابع پاداش کم شدن زمان منتظر ماندن مسافرین و سوخت مصرفی تاکسی‌ها است. در کل هدف یادگیری خط مشی‌ای برای هر نوع عاملی است که اعمالش بر روی محیط تأثیر می‌گذارد و تابعی تجمعی برای کیفیت هر عمل در دسترس دارد. به همراه این نوع مسائل معمولاً شرایطی در نظر گرفته می‌شود، مثلاً اینکه اعمال همیشه قطعی[4] هستند یا نه، یا اینکه عامل از قبل اطلاعاتی را در باره ی اعمالش دارد یا خیر.

[1] policy
[2] reward function
[3] cumulative
[4] deterministic

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \cdots$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{ , where } 0 \leq \gamma < 1$$

شکل ۱۳.۱ عاملی در ارتباط با محیطش. عاملی در محیطی است که حالتش یکی از اعضای S است.
عامل می‌تواند اعمال مجموعه‌ی A را انجام دهد. هر بار که عملی مثل a_t را در حالت s_t انجام می‌دهد مقدار حقیقی‌ای r_t را از تابع پاداش دریافت می‌کند که حاصل لحظه ای این عمل است. این فرایند سری‌ای از s_i و a_iها و r_iها را ایجاد می‌کند. هدف عامل این است که خط مشی‌ای به فرم $\pi: S \rightarrow A$ را یاد بگیرد که مجموع این پاداش با پاداش‌های آینده در سری توانی حداکثر شود.

توجه دارید که قبلاً نیز مسائل یادگیری سری اعمال را در این کتاب دیده بودیم. در بخش ۱۱.۴ درباره‌ی یادگیری توضیحی دسته قوانین برای کنترل جستجو را در حین حل مسئله بحث کردیم. مسئله‌ی مطرح برای عامل در آن بخش انتخاب بین گزینه های مختلف عمل در هر مرحله از جستجو برای یافتن حالت هدف بود. تفاوت تکنیک‌هایی که در اینجا به کار می‌بریم با تکنیک‌های بخش ۱۱.۴ در این است که مسائلی را در نظر می‌گیریم که اعمال ممکن است نتایج غیر قطعی داشته باشند و یادگیر نیز تئوری قلمرویی درباره‌ی غیر قطعی بودن نتایج اعمال ندارد. در فصل ۱ مسئله‌ی یادگیری انتخاب‌های اعمال بازی چکرز را مورد بحث قرار دادیم. در آنجا طراحی متد یادگیری‌ای بسیار مشابه متد های این فصل را طراحی کردیم. در واقع، یکی از موفق‌ترین کاربردهای الگوریتم‌های یادگیری تقویتی این فصل برای مسئله‌ی طرز-بازی[1]، بازی مشابهی است. (Tesauro 1995) برنامه‌ی TD-Gammon را که با روش یادگیری تقویتی برای رسیدن به بازیکن جهانی برای بازی تخته نرد آموزش دیده را معرفی می‌کند. این برنامه بعد ۱.۵ میلیون بازی خودساخته[2]، حال در حد بهترین بازیکن‌های جهانی شمرده می‌شود و در مقابل بسیاری از بازیکنان سطح بالا در مسابقات جهانی تخته نرد بازی می‌کند.

مسئله‌ی یادگیری خط مشی انتخاب اعمال از جنبه‌هایی شبیه مسائل تخمین توابع که در فصل‌های گذشته بررسی کرده‌ایم است. در این تشابه تابع هدفی که باید یاد گرفته شود، تابع $\pi: S \rightarrow A$ است که برای هر حالت s از مجموعه‌ی S، یک عمل مثل a از مجموعه‌ی A را نظیر می‌کند. با وجود این تشابه مسئله‌ی یادگیری تقویتی با مسائل تخمین توابع در بسیاری از جنبه‌ها متفاوت است:

[1] game-playing
[2] self-generated

www.farabar.net

Telegram: @Farabar_BI

- پاداش تأخیری[1]. هدف یادگیری، یادگیری تابع هدفی مثل π که هر حالت s را به عملی ربط دهد a= π(s). در فصول قبلی، همیشه فرض بر این بوده که تابع هدفی چون π را از نمونه های آموزشی‌ای به فرم <s, π(s)> یاد می‌گیریم. در حالی که در یادگیری تقویتی اطلاعات آموزشی به این فرم موجود نیست و به جای آن مربی[2] سری پاداش‌هایی لحظه ای نظیر حرکات را به عامل می‌دهد. پس عامل با مسئله‌ی نسبت دادن ارزش موقتی[3] مواجه است: تصمیم گیری برای اینکه کدام یک از اعمال در سری اعمال احتمالاً بعداً باعث پاداش خواهند داد.
- جستجوی محیط[4]. در یادگیری تقویتی، عامل با انتخاب اعمال بر توزیع نمونه های آموزشی تأثیر می‌گذارد. همین تأثیر سؤالی دیگر را مطرح می‌کند، کدام استراتژی در انتخاب آزمایش‌ها[5] آموزش بهینه تری را فراهم می‌کند؟ یادگیر دو انتخاب خواهد داشت، یکی اینکه حالت‌ها و اعمال جدید را جستجو کند (تا اطلاعات جدیدی بدست آورد) و دیگر اینکه به اطلاعاتی که تا به حال پیدا کرده اکتفا کند و از آن‌ها برای رسیدن به بیشترین پاداش استفاده کند[6].
- حالت‌های نیمه معلوم. با وجود اینکه راحت‌تر است فرض کنیم حسگر های عامل می‌توانند تمام شرایط محیط را در یک حالت را معلوم کنند، اما با این حال در بسیاری از مثال‌های کاربردی حسگرها اطلاعات کاملی در مورد محیط به ما نمی‌دهند. مثلاً، در مثال ربات زمانی که از یک دوربین روبه جلو استفاده می‌شود اجسام پشت سر ربات توسط دوربین مشخص نمی‌شوند. در چنین شرایط بهتر است که در هنگام انتخاب عمل، عامل شرایط مشاهده شده‌ی قبلی را نیز در نظر بگیرد، شاید بهترین خط مشی انتخاب اعمالی باشد که میزان مشاهده‌ی ربات را از محیط معلوم کند.
- یادگیری مادام‌العمر[7]. بر خلاف تخمین توابع ، در یادگیری رباتیک گاهی نیاز است که ربات چندین کار مرتبط با هم را در همان محیط و با همان سنسورها یاد بگیرد. برای مثال ممکن است نیاز باشد که یک ربات متحرک علاوه بر وصل شدن به باتری، نحوه‌ی حرکت در راهروهای باریک و نحوه‌ی برداشتن خروجی پرینتر را نیز یاد بگیرد. این باعث می‌شود که آزمایش‌ها قبلی و دانش بدست آمده در حالت‌های قبلی را بتواند در یادگیری کارهای جدید به کار ببرد.

۱۳.۲ یادگیری کارها

در این قسمت دقیق‌تر و با دید ریاضی به مسئله‌ی یادگیری سری استراتژی‌های کنترل می‌پردازیم. توجه داشته باشید که راه‌های بسیاری برای این بررسی وجود دارد. برای مثال، ممکن است فرض کنیم که اعمال قطعی یا غیر قطعی هستند. یا ممکن است فرض کنیم که عامل می‌تواند حالت بعد از هر عمل را پیش بینی کند و یا در نقطه‌ی مقابل حالت‌ها غیر قابل پیش بینی هستند. یا حتی ممکن است فرض کنیم که عامل توسط یک معلم آموزش داده می‌شود که تمامی راه‌های بهینه را برای انجام اعمال نشان می‌دهد، یا در نقطه‌ی مقابل خود ربات باید با انجام

[1] delayed reward
[2] trainer
[3] temporal credit assignment
[4] exploration
[5] experimentation strategy
[6] exploitation
[7] life-long learning

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

اعمال کارها را یاد بگیرد. در اینجا ما فرمولی کلی را که از فرایند های تصمیم گیری مارکوو[1] بدست آمده ارائه می‌کنیم. این فرمول در شکل ۱۳.۱ آورده شده است.

در یک فرایند تصمیم گیری مارکوو (MDP) عامل مجموعه ای از حالت‌ها به نام S و مجموعه ای از اعمال به نام A را در اختیار دارد. در هر لحظه‌ی t، حسگر های عامل حالت s_t را مشخص می‌کنند و عامل عمل a_t را انجام می‌دهد. محیط نیز در مقابل پاداش $r_t = r(s_t, a_t)$ را به عامل می‌دهد و حالت $s_{t+1} = \delta(s_t, a_t)$ را ایجاد می‌کند. در اینجا دو تابع r و δ جزو محیط هستند و الزاماً برای عامل مشخص نیستند. در یک MDP توابع $r(s_t, a_t)$ و $\delta(s_t, a_t)$ فقط به حالت فعلی و عمل وابسته‌اند و به حالت‌ها و اعمال قبلی هیچ وابستگی‌ای ندارند. در این فصل فقط حالت‌هایی را که در آن S و A متناهی هستند را بررسی می‌کنیم. در کل، دو تابع r و δ ممکن است توابع قطعی‌ای نباشند اما در ابتدا فرض می‌کنیم که این توابع قطعی‌اند.

کار عامل یادگیری خط مشی‌ای چون π: S → A است که بتواند حرکت بعدی a_t را بر اساس حالت فعلی s_t بدست بیاورد؛ داریم که $\pi(s_t) = a_t$. اما چگونه می‌توانیم دقیقا مشخص کنیم که عامل کدام خط مشی را برای π یاد بگیرد؟ یکی از راه حل‌های بسیار ساده تعیین خط مشی بهینه، تعریف آن به صورتی است که تابع تجمعی پاداش در طول زمان را حداکثر کند. برای تعریف دقیق‌تر مقدار تجمعی $V^{\pi}(s_t)$ را که توسط خط مشی π از حالت اولیه‌ی s_t بدست می‌آید را به فرم زیر تعریف می‌کنیم:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad (13.1)$$

در این رابطه سری ی r_{t+i} پاداش‌هایی هستند که از خط مشی π و شروع از s_t بدست می‌آید ($a_t = (s_t)$ و $a_{t+1} = (s_{t+1})$ و...). در این رابطه $0 \le \gamma < 1$ ثابتی برای پاداش‌های تأخیری است. در کل، تأثیر پاداشی که i مرحله بعد از انجام عمل داده می‌شود توسط ضریب توانی γ^i کوچک می‌شود. توجه داشته باشید که اگر γ =0، فقط پاداش لحظه ای در نظر گرفته خواهد شد. با بیشتر کردن مقدار γ تأثیر نسبی پاداش‌های تأخیری در تابع بیشتر می‌شود.

کمیت $V^{\pi}(s_t)$ که در رابطه‌ی ۱۳.۱ تعریف شده گاهی پاداش تجمعی تخفیفی[2] برای خط مشی π در حالت s نیز نامیده می‌شود. منطقی است که پاداش‌های آینده را تخفیف دهیم، زیرا که در بسیاری از موارد هدف ما بر سریع‌تر رسیدن به پاداش است. با این وجود، در بعضی موارد از کل پاداش دریافتی نیز استفاده می‌شود. برای مثال، finite horizon reward به فرم $\sum_{i=0}^{h} r_{t+i}$ تعریف می‌شود که پاداش‌ها را تا h پله‌ی بعدی مورد نظر قرار می‌دهد. یا متوسط پاداش[3] به صورت $\lim_{h\to\infty} \frac{1}{h}\sum_{i=0}^{h} r_{t+i}$ تعریف می‌شود که متوسط کل پاداش دریافتی در طول عمر ربات را در نظر می‌گیرد. در این فصل ما خود را محدود به رابطه‌ی 13.1 می‌کنیم. برای پاداش متوسط به (Mahadevan 1996) مراجعه کنید.

[1] Markov decition processes
[2] discounted cumulative reward
[3] average reward

www.farabar.net

Telegram: @Farabar_BI

حال به جایی رسیده‌ایم که کار یادگیری عامل را دقیق مشخص کنیم. باید خط مشی‌ای را پیدا کنیم تا برای تمامی حالت‌های s مقدار $V^{\pi}(s_t)$ ماکزیمم شود. چنین خط مشی‌ای را خط مشی بهینه[1] می‌نامیم و با π^* نشان می‌دهیم.

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

برای ساده سازی نمایش، تابع $V^{\pi^*}(s)$ را به صورت $V^*(s)$ نمایش می‌دهیم. $V^*(s)$ بیشترین مقدار ممکن پاداش تجمعی تخفیفی را برای هر حالت s می‌دهد؛ به عبارت دیگر، این مقدار، مقدار خط مشی بهینه برای حالت s است.

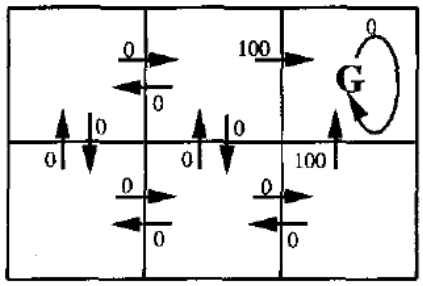
برای درک این مفاهیم، قسمت اول شکل ۱۳.۲ را که محیطی گسسته را نشان داده در نظر بگیرید. شش مربع نشان داده شده در شکل نشان دهنده‌ی شش حالت یا موقعیت برای عامل هستند. هر فلش در شکل یک عمل ممکن که عامل می‌تواند برای تغییر حالت خود انجام دهد را نشان می‌دهد. هر عدد روی فلش مقدار پاداش r(s,a) را که عامل پس از انجام هر عمل می‌گیرد را نشان می‌دهد. توجه داشته باشید که تمامی پاداش‌های لحظه ای در این مثال جز برای عملی که به خانه‌ی G ختم می‌شود صفر است. می‌توان حالت G را به عنوان حالت هدف در نظر گرفت زیرا که فقط با ورود به خانه‌ی G عامل پاداش دریافت می‌کند. توجه داشته باشید که در این محیط خاص تنها اعمالی که برای حالت G در نظر گرفته شده باقی ماندن در همان حالت است. به همین دلیل حالت G را حالت جاذب[2] می‌نامیم.

با معلوم بودن تمامی حالت‌ها، اعمال و پاداش‌ها می‌توان به راحتی با تعیین مقدار γ خط مشی بهینه π^* و تابع $V^*(s)$ آن‌را مشخص کرد. در این حالت بیایید فرض کنیم که γ=0.9. نمودار وسطی یکی از خط مشی‌های بهینه را برای این شرایط مشخص می‌کند. مثل تمامی خط مشی‌ها، این خط مشی نیز در هر حالت فقط یک عمل را پیشنهاد می‌کند. جای تعجب ندارد که خط مشی بهینه کوتاه‌ترین راه را به سمت حالت G نشان می‌دهد.

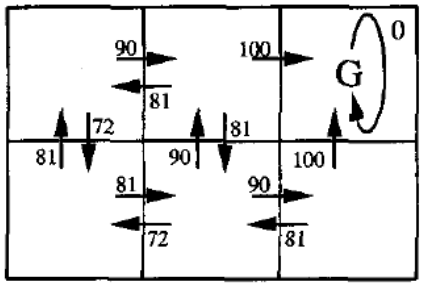
[1] optimal policy
[2] absorbing state

www.farabar.net

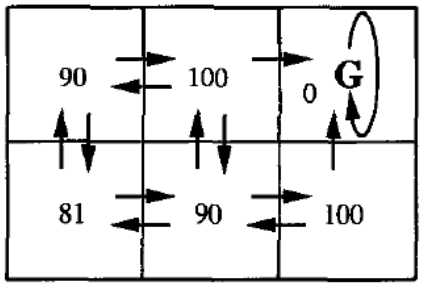
Telegram: @Farabar_BI
www.farabar.net



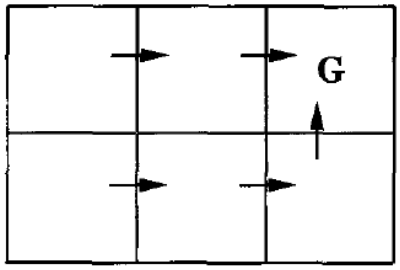
$r(s,a)$ (immediate reward) values



$Q(s,a)$ values



$V^*(s)$ values



One optimal policy

شکل ۱۳.۲ یک محیط قطعی برای تصور مفاهیم اولیه‌ی یادگیری Q.

هر مربع یک حالت و هر فلش یک عمل را نشان می‌دهند. تابع پاداش، r(s,a) در ورود به حالت G 100+ و در بقیه‌ی موارد صفر است. مقادیر $V^*(s)$ و Q(s,a) که از r(s,a) با $\gamma = 0.9$ بدست آمده است. خط مشی بهینه ای نیز با مقادیر Q ماکزیمم نیز در شکل آمده است.

شکل سمت راست در شکل ۱۳.۲ مقادیر V^* را برای هر حالت نشان می‌دهد. برای مثال، مربع گوشه‌ی پایین و راست شکل را در نظر بگیرید. مقدار V^* برای این مربع ۱۰۰ است زیرا که خط مشی برای این مربع فلش رو به بالا را انتخاب می‌کند و عامل پاداش لحظه ای ۱۰۰ را می‌گیرد و پس از آن نیز عامل در حالت جاذب می‌ماند و هیچ پاداش دیگری نیز نمی‌گیرد. مشابهاً، مقدار V^* برای مربع پایین و وسط ۹۰ است. این بخاطر این است که خط مشی بهینه ابتدا عمل به سمت راست رفتن را با پاداش لحظه ای صفر و سپس عمل بالا رفتن را با پاداش لحظه ای ۱۰۰ انجام می‌دهد. پس به سادگی برای این مربع با توجه به رابطه‌ی V^* می‌توان نوشت:

$$0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \cdots = 90$$

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

در این محیط خاص بعد از رسیدن به حالت جاذب G پاداش اعمال بعدی صفر خواهد بود و عامل دیگر پاداشی دریافت نمی‌کند.

۱۳.۳ یادگیری Q

چگونه عامل می‌تواند برای محیطی دلخواه خط مشی بهینه π^* را یاد بگیرد؟ یادگیری مستقیم $\pi^*: S \to A$ ممکن نیست زیرا که نمونه‌هایی آموزشی به شکل <s,a> نداریم و بجای آن تنها مقادیر پاداش‌ها را به فرم $r(s_i, a_i)$ برای i=0,1,2,3,... داریم. همان طور که بعداً نیز خواهیم دید با داشتن چنین اطلاعاتی یادگیری تابع تخمینی عددی بر روی حالت‌ها و اعمال ساده تر است. بعد از پیدا کردن تابع تخمین عددی، با استفاده از آن خط مشی بهینه را پیدا می‌کنیم.

چه تابع تخمینی را باید عامل یاد بگیرد؟ یکی از انتخاب‌های آشکار خود V^* است. عامل باید حالت s_1 را هر گاه که $V^*(s_1) > V^*(s_2)$ به حالت s_2 ترجیح دهد. البته خط مشی بین اعمال حق انتخاب دارد نه بین حالت‌ها. با این وجود می‌توان برای انتخاب بین اعمال نیز از V^* استفاده کرد. عمل بهینه در حالت s عملی مثل a است که مجموع پاداش‌های لحظه ای و V^* حالت پایانی‌اش ضرب در γ ماکزیمم باشد:

$$\pi^*(s) = arg\max_a \left[r(s,a) + \gamma V^*\big(\delta(s,a)\big)\right] \qquad (13.3)$$

(توجه داشته باشید که $\delta(s,a)$ نماد حالت بعدی برای عمل a از حالت s تعریف شد). بنابراین عامل با استفاده از V^* و اطلاعات کافی در مورد تابع پاداش‌های لحظه ای r و تابع حالت‌های پایانی δ می‌تواند خط مشی بهینه را مشخص کند. پس زمانی که عامل دو تابع δ و r را که محیط برای تغییرات استفاده می‌کند را بداند می‌تواند با استفاده از رابطه‌ی ۱۳.۳ عمل بهینه را برای حالت دلخواه s مشخص کند.

متأسفانه، با یادگیری V^* فقط زمانی می‌توان خط مشی بهینه را مشخص کرد که عامل دو تابع δ و r را بداند. چنین چیزی معادل دانستن نتایج لحظه ای (هم پاداش‌های لحظه ای و هم حالت‌های پایانی) برای هر زوج مرتب حالت عمل است. این فرض مشابه داشتن تئوری قلمروی کامل در یادگیری توضیحی (فصل ۱۱) است. در بسیاری از مسائل کاربردی، مثل کنترل ربات، پیش بینی دقیق نتیجه‌ی هر حرکت در هر حالت برای عامل و یا برنامه‌نویسش غیرممکن است. برای مثال، فرض کنید که بازوی بیل شکل رباتی می‌خواهد مقدار خاک برداری کند، و حالت نهایی مکان ذرات خاک بعد از خاک برداری است. در چنین شرایطی هم δ و هم r نامعلومند، و متأسفانه یادگیری V^* مفید نخواهد بود زیرا که عامل نمی‌تواند رابطه‌ی ۱۳.۳ را کامل کند. عامل باید از چه تابع تخمینی برای این تعریف مسئله‌ی کلی‌تر استفاده کند؟ تابع تخمین Q، که در بخش بعدی تعریف خواهد شد، جوابی برای این مسئله ارائه می‌کند.

۱۳.۳.۱ تابع Q

فرض کنید که تابع تخمینی Q(s,a) را به صورتی تعریف کرده‌ایم که مقدارش همیشه ماکزیمم ممکن تابع پاداش تجمعی تخفیفی برای هر حالت s و عمل a در گام اول است. به عبارت دیگر، مقدار عبارت Q همیشه جمع مقدار پاداش لحظه ای عمل a و پاداش خط مشی بهینه‌ی بعد از آن (با تخفیف γ) است:

$$Q(s,a) \equiv r(s,a) + \gamma V^*\big(\delta(s,a)\big) \qquad (13.4)$$

توجه داشته باشید که Q(s,a) دقیقا همان کمیتی است که در رابطه‌ی ۱۳.۳ برای انتخاب a در حالت s ماکزیمم شده است. بنابراین با بازنویسی رابطه خواهیم داشت:

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

$$\pi^*(s) = arg\max_a Q(s,a) \qquad (13.3)$$

اهمیت این بازنویسی در چیست؟ این رابطه نشان می‌دهد که اگر عامل تابع Q را به جای V^* یاد بگیرد بدون نیاز به دو تابع r و δ می‌تواند اعمال بهینه را پیدا کند. همان طور که در رابطه‌ی ۱۳.۵ نیز آمده است، کافی است با در نظر گرفتن حالت‌های مختلف برای a در حالت s عملی را انتخاب کنیم که Q(s,a) را ماکزیمم می‌کند.

در ابتدا ممکن است عجیب به نظر برسد که با ماکزیمم کردن مقداری موضعی مثل Q می‌توان با تکرار اعمال به بیشترین پاداش کلی رسید. و این بدین معناست که عامل بدون اینکه حتی در نظر بگیرد که بعد از این عمل چه حالت رخ خواهد داد می‌تواند عمل بهینه را پیدا کند. زیبایی یادگیری Q[1] در این است که تابع Q طوری تعریف شده که تمامی اطلاعات لازم درباره‌ی تابع پاداش تجمعی تخفیفی در آینده با انتخاب عمل a در حالت s را در خود داراست.

برای درک بهتر، شکل ۱۳.۲ مقادیر تابع Q را برای هر حالت در محیط مربعی تعریف شده نشان می‌دهد. توجه داشته باشید که مقدار Q برای هر زوج مرتب حالت عمل مجموع مقدار پاداش لحظه ای و مقدار تابع V^* را با تخفیف γ نشان می‌دهد. همچنین توجه داشته باشید که خط مشی بهینه نیز با مقادیر Q برای حالت‌های مختلف متناسب است.

۱۳.۳.۲ الگوریتمی برای یادگیری Q

یادگیری Q ارتباطی مستقیم با پیدا کردن خط مشی بهینه دارد. اما چگونه می‌توان Q را یاد گرفت؟

کلید حل این مشکل پیدا کردن راهی قابل اطمینان برای تخمین مقادیر آموزشی برای Q، از طریق تنها داده های موجود یعنی سری پاداش‌های r در طول زمان است. می‌توان با تخمین تکراری‌ای به چنین چیزی دست یافت. برای پی بردن به چگونگی این امر، بیایید به رابطه‌ی بین Q و V^* نگاهی دقیق‌تر بیندازیم،

$$V^*(s) = \max_{a'} Q(s,a')$$

که با توجه به رابطه‌ی ۱۳.۴ خواهیم داشت:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a') \qquad (13.6)$$

این تعریف بازگشتی Q پایه‌ی الگوریتم‌های حلقه ای برای تخمین Q است (Watkins 1989). برای توضیح الگوریتم، از نماد $\hat{Q}$ برای نمایش تخمین یا همان فرضیه‌ی یادگیر استفاده می‌کنیم، و نماد Q را برای تابع اصلی به کار می‌بریم. در این الگوریتم یادگیر فرضیه‌ی $\hat{Q}$ را با جدولی بزرگ که برای هر جفت حالت و عمل عددی آورده شده نشان می‌دهد. جدول مقادیر <s,a> مقادیر را برای $\hat{Q}(s,a)$ نگه داری می‌کند، فرضیه‌ی فعلی یادگیر که تخمینی از Q اصلی است. جدول در گام اول با مقادیر تصادفی پر می‌شود (اگر فرض کنیم جدول در گام اول با صفر پر می‌شود درک الگوریتم راحت‌تر خواهد بود). در هر مرحله عامل حالت s را مشاهده می‌کند و عملی چون a را انجام می‌دهد، سپس

[1] Q learning

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

پاداش ناشی از عمل r=r(s,a) و حالت پایانی s′=δ(s,a) را مشاهده می‌کند. سپس مقدار $\hat{Q}(s,a)$ داخل جدول را برای چنین حالتی به فرم زیر تغییر می دهد:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \qquad (13.7)$$

توجه دارید که عامل از مقدار فعلی $\hat{Q}$ در s′ برای تخمین $\hat{Q}$ در مرحله‌ی قبلی استفاده می‌کند. با وجود اینکه این رابطه به تخمین قبلی عامل از $\hat{Q}$ نیز وابسته است اما می‌توان گفت این قانون آموزش از رابطه‌ی ۱۳.۶ نشأت گرفته است. توجه داشته باشید که با وجود اینکه رابطه‌ی ۱۳.۶ Q را بر اساس دو تابع δ(s,a) و r(s,a) توصیف می‌کند، عامل برای پیاده سازی رابطه‌ی ۱۳.۷ هیچ نیازی به دانستن این دو رابطه ندارد. به جای آن عامل عمل مذکور را انجام می‌دهد و حالت جدید s′ و پاداش r را مشاهده می‌کند. پس می‌توان گفت که در این رابطه دو مقدار s′ و r این دو تابع را مدل سازی می‌کنند.

الگوریتمی که در بالا توضیح داده شد برای سیستم‌های تصمیم گیری مارکوو دقیق‌تر در جدول ۱۳.۱ آورده شده است. با استفاده از این الگوریتم تخمین عامل $\hat{Q}$ به مقدار واقعی Q میل می‌کند. با توجه به اینکه فرض کردیم سیستم یک مدل تصمیم گیری مارکوو است، تابع پاداش کران دار است و اعمال طوری انتخاب می‌شوند که هر جفت حالت و عمل چند دفعه یک بار مشاهده شود.

الگوریتم یادگیری Q

برای هر جفت s و a مقدار اولیه‌ی جدول $\hat{Q}(s,a)$ را صفر قرار بده

حالت فعلی s را مشاهده کن

حلقه‌ی زیر را تا بینهایت ادامه بده:

- عملی مثل a را انتخاب کن و آن‌را انجام بده.
- مقدار پاداش لحظه ای r را دریافت کن
- حالت جدید s′ را مشاهده کن
- مقدار جدول را با رابطه‌ی زیر تغییر بده

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

جدول ۱۳.۱

الگوریتم یادگیری Q با فرض اینکه پاداش‌ها و اعمال قطعی هستند. ثابت تخفیف $\mathbf{0 \le \gamma < 1}$ *در نظر گرفته شده است.*

۱۳.۳.۳ مثالی توضیحی

برای تصور بهتر یادگیری Q، عمل و حالت‌های نشان داده شده در شکل ۱۳.۳ را در نظر بگیرید. در اینجا یک عامل عملی را انجام داده و $\hat{Q}$ را با توجه به این عمل تغییر می‌دهد. عامل در این مثال به سمت راست حرکت کرده (عمل) و برای این عمل پاداش صفر را دریافت می‌کند.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

سپس با توجه به رابطه‌ی ۱۳.۷ مقدار تخمینی خود $\hat{Q}$ را برای زوج حالت و عملی که انجام داده تغییر می‌دهد. با توجه به رابطه مقدار جدید $\hat{Q}$ جمع پاداش لحظه ای عمل (صفر) و بیشترین مقدار $\hat{Q}$ برای حالت پایانی (۱۰۰) با تخفیف γ، ۰.۹ خواهد بود.

در هر تکرار عامل عملی را انجام داده و از حالت قدیمی به حالت جدید می‌رود، و یادگیری Q نیز در هر مرحله $\hat{Q}$ را از مرحله‌ی جدید به مرحله‌ی قبلی گسترش می‌دهد به طور همزمان میزان پاداش لحظه ای توسط عامل دریافت شده و در تغییر $\hat{Q}$ تاثیر خود را می‌گذارد.

فرض کنید که این الگوریتم را برای محیط مربعی و تابع پاداش نشان داده شده در شکل ۱۳.۲ به کار ببریم. از آنجایی که این محیط یک حالت جاذب دارد باید آموزش را در چند قسمت انجام دهیم. در هر قسمت عامل در حالتی تصادفی قرار می‌گیرد و تا رسیدن به حالت جاذب حق انتخاب اعمال را خواهد داشت. زمانی که عامل به حالت جاذب می‌رسد این قسمت از آموزش پایان می‌یابد و عامل دوباره به حالتی تصادفی فرستاده می‌شود تا قسمت بعدی آموزش انجام شود.

مقادیر $\hat{Q}$ چگونه در این مثال با استفاده از یادگیری Q تغییر می‌کنند؟ در ابتدای آموزش همه‌ی مقادیر $\hat{Q}$ صفر قرار داده می‌شوند و تا زمانی که عامل به حالت جاذب نرسیده و پاداشی غیر صفر دریافت نکرده این مقادیر صفر می‌مانند. با ورود به حالت جاذب، حالتی که عامل قبلاً در آن بوده مقدار $\hat{Q}$ خود را تغییر می‌دهد. در قسمت بعدی آموزش اگر عامل از حالتی که گفته شد رد شود مقدار غیر صفر $\hat{Q}$ در آن خانه باعث می‌شود تا خانه ای با فاصله‌ی دو از G نیز مقدار $\hat{Q}$ خود را پیدا کند و به همین ترتیب. اگر تعداد قسمت‌های آموزشی کافی باشد، اطلاعات لازم از طریق پاداش‌های غیر صفر در میان فضای جفت حالت عمل پخش خواهد شد و در آخر نیز به جدولی که در شکل ۱۳.۲ نیز نشان داده شده است ختم می‌شود.

در قسمت بعدی ثابت خواهیم کرد که الگوریتم ارائه شده در جدول ۱۳.۱ به سمت تابع Q میل خواهد کرد. اما ابتدا دو خاصیت کلی الگوریتم یادگیری Q را که در مورد تمامی MDP های قطعی با پاداش‌های غیر منفی صادق است را بررسی می‌کنیم. فرض کنید که تمامی مقادیر اولیه‌ی $\hat{Q}$ را صفر قرار داده‌ایم. اولین خاصیت این است که با این شرایط مقادیر $\hat{Q}$ هیچ‌وقت در طول آموزش کاهش نمی‌یابد. به عبارت دیگر اگر $\hat{Q}_n(s,a)$ نماد مقدار یاد گرفته شده‌ی $\hat{Q}(s,a)$ بعد از n تکرار فرایند (مثلاً بعد از اینکه عامل n جفت حالت و عمل را بررسی کرد) باشد، خواهیم داشت:

$$(\forall s,a,n)\ \hat{Q}_{n+1}(s,a) \geq \hat{Q}_n(s,a)$$

خاصیت کلی دوم این است که در طول آموزش همیشه مقدار $\hat{Q}$ بین دو مقدار صفر و مقدار حقیقی Q خواهد بود.

$$(\forall s,a,n)\ 0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

۱۳.۳.٤ همگرایی

آیا تقریب $\hat{Q}$ که در الگوریتم جدول ۱۳.۱ آورده شده به مقدار حقیقی Q میل خواهد کرد؟ جواب مثبت است، اما با شرایطی. ابتدا باید فرض کنیم که سیستم یک سیستم قطعی MDP است. دوم اینکه باید فرض کنیم که پاداش‌های لحظه ای کران دارند؛ یعنی اینکه مقداری ثابت مثل c موجود است که برای تمامی حالت‌های s و تمامی اعمال a داشته باشیم $|r(s,a)|<c$. سوم اینکه باید فرض کنیم که عامل هر جفت حالت و عمل را حداقل هر چند وقت یک بار بررسی می‌کند. به عبارت دیگر اگر عمل a در حالت s یک قانونی بود، در طول زمان عامل باید تا زمانی که طول سری اعمال به بینهایت برسد عمل a از حالت s با تناوبی غیر صفر انجام دهد. توجه دارید که این شرایط از جنبه ای محدود و

www.farabar.net

از جنبه ای بسیار کلی هستند. نسبت به مثال‌های بررسی شده این شرایط بسیار کلی هستند زیرا که با چنین شرایطی محیط‌های دلخواهی را می‌توان در نظر گرفت که پاداش‌های صفر یا مثبت یا منفی و تعداد دلخواهی جفت حالت و عمل داشته باشند. اما از طرفی دیگر این محدودیت وجود دارد که باید عامل بتواند در محیط هر جفت حالت و عمل را چند وقت یک بار بررسی کند. در کل این شرط، در محیط‌های بزرگ و پیوسته شرط بسیار قوی‌ای است. در آینده به نتایج قوی‌تر همگرایی خواهیم پرداخت. با این وجود شروط بررسی شده در این قسمت پایه های درک درستی یادگیری Q است.

نکته‌ی کلیدی اثبات همگرایی این است که پر خطاترین مقدار جدول $\hat{Q}$ هر زمان که بررسی می‌شود باید به نسبت ضریب γ خطای خود را تصحیح کند. زیرا که قسمتی از مقدار جدید $\hat{Q}$ به مقدار خطا دار $\hat{Q}$ و بقیه‌ی آن به مقدار بدون خطای پاداش r بستگی دارد.

قضیه‌ی ۱۳.۱. همگرایی یادگیری Q برای فرایند های تصمیم گیری مارکوو در حالت قطعی. یک عامل یادگیر را در یک MDP قطعی با پاداش‌های کران دار در نظر بگیرید: $(\forall s,a)|r(s,a)| \leq c$. که این عامل از قانون یادگیری Q که در رابطه‌ی ۱۳.۷ استفاده می‌کند. پس عامل مقادیر $\hat{Q}(s,a)$ را با مقادیر دلخواه کران دار مقدار دهی اولیه می‌کند و عامل تخفیف نیز γ است که $0 \leq \gamma < 1$. اگر $\hat{Q}_n(s,a)$ نماد فرضیه‌ی $\hat{Q}(s,a)$ پس از n بار تغییر باشد و هر جفت حالت و عمل نیز چند وقت یک بار بررسی شود، آنگاه زمانی که n→∞ برای تمامی مقادیر a و s، $\hat{Q}_n(s,a)$ به Q(s, a) میل خواهد کرد.

اثبات. از آنجایی که هر زوج حالت و عمل چند وقت یک بار بررسی می‌شوند، بازه های پیاپی‌ای را در نظر بگیرید که در آن تمامی زوج حالت و عمل چند وقت یک بار بررسی می‌شوند. در اینجا ثابت می‌کنیم که در چنین بازه ای خطای ماکزیمم برای تمامی مقادیر جدول $\hat{Q}$ حداقل متناسب با γ کاهش می‌یابد. اگر $\hat{Q}_n(s,a)$ جدول مقادیر فرضیه بعد از n بار تغییر باشد و Δ_n نیز حداکثر خطای این جدول باشد (به فرم زیر):

$$\Delta_n \equiv \max_{s,a} |\hat{Q}(s,a) - Q(s,a)|$$

در زیر از نماد s′ برای نمایش δ(s,a) استفاده خواهیم کرد. حال برای هر مقدار جدول $\hat{Q}_n(s,a)$ که در حلقه‌ی n+1 ام عوض می‌شود، اندازه‌ی خطا در $\hat{Q}_{n+1}(s,a)$ خواهد بود:

$$\begin{aligned}
\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| &= |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q(s,a)| \\
&= \gamma| \max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')| \\
&\leq \gamma \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\leq \gamma \max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')|
\end{aligned}$$

$$\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| \leq \gamma \Delta_n$$

خط دوم به سوم در روابط بالا از رابطه‌ی کلی‌ای زیر برای دو تابع دلخواه f_1 و f_2 ناشی شده:

Telegram: @Farabar_BI
www.farabar.net

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

در خط سوم به چهارم نیز توجه کنید که متغیر جدیدی به نام s'' را که متغیر ماکزیمم کننده است اضافه کرده‌ایم. این کار از این رو درست است که مقدار ماکزیمم بعد از اضافه کردن متغیر جدید بزرگ‌تر یا مساوی خواهد شد. توجه داشته باشید که با اضافه کردن این متغیر جدید تعریف Δ_n را در نامعادله می‌سازیم.

بنابراین تغییر $\hat{Q}_{n+1}$ برای هر مقدار s و a حداقل به نسبت γ از ماکزیمم خطای جدول $\hat{Q}_n$، Δ_n، کوچک‌تر است. از طرفی بزرگ‌ترین خطای اولین جدول یا همان Δ_0 نیز کران دار است زیرا تمامی مقادیر $\hat{Q}_0$ و Q(s,a) به ازای تمامی مقادیر a و s کران دارند. حال بعد از اولین بازه ای که تمامی s,a ها را بررسی می‌کند، بزرگ‌ترین خطای موجود در جدول $\gamma\Delta_0$ خواهد بود. بعد از k بار تکرار حداکثر خطا به $\gamma^k\Delta_0$ تقلیل خواهد یافت. از آنجایی که هر حالت هر چند وقت یک بار بررسی شده زمانی که n→∞ تعداد چنین تکرارهایی نیز به سمت بینهایت میل خواهد کرد و متعاقباً خواهیم داشت که $\Delta_n \to 0$.

۱۳.۳.۵ استراتژی‌های آزمایش

توجه دارید که الگوریتم جدول ۱۳.۱ نحوه‌ی انتخاب اعمال توسط عامل را مشخص نکرده. یکی از استراتژی‌های ساده‌ی موجود انتخاب عملی است که مقدار $\hat{Q}(s,a)$ را ماکزیمم کند و از اطلاعاتی که تا به حال بدست آورده برای بدست آوردن پاداش استفاده کند. با این وجود با انتخاب چنین استراتژی‌ای عامل ریسک اعتماد[1] را افزایش می‌دهد. یعنی اینکه در مراحل اولیه‌ی تخمین $\hat{Q}$ عملی مقدار زیادی در $\hat{Q}$ را پیدا کرده در حالی که Q همان عمل ماکزیمم نیست. از طرف دیگر طبق قضیه‌ی بالا برای اینکه $\hat{Q}$ به Q همگرا شود باید تمامی اعمال هر چند وقت یک بار انجام شوند. واضح است که با انتخاب ماکزیمم در هر حالت چنین شرطی برآورده نخواهد شد. به همین دلیل، معمول است که در یادگیری Q برای انتخاب اعمال از روشی احتمالی استفاده می‌کنند. مسلماً به اعمالی که $\hat{Q}$ بیشتر دارند احتمال بیشتری داده می‌شود اما به تمامی اعمال احتمالی غیر صفر باید داده شود. یکی از راه‌های انتخاب این احتمالات به شکل زیر است:

$$P(a_i|s) = \frac{k^{\hat{Q}(s,a)}}{\sum_j k^{\hat{Q}(s,a)}}$$

در این رابطه $P(a_i|s)$ احتمال انتخاب عمل a_i در حالت s است و k نیز ثابتی مثبت است که قدرت انتخاب گزینه‌ی بهتر را نشان می‌دهد. با افزایش مقدار k، اعمالی که $\hat{Q}$ بیشتر دارند احتمال بیشتری خواهند داشت، و باعث می‌شود تا عامل از دانسته‌هایش استفاده[2] کند تا اینکه به جستجوی[3] محیط بپردازد. در مقابل با کم شدن k احتمال اعمال دیگر افزایش می‌یابد و عامل بیشتر به سمت جستجو تمایل پیدا می‌کند. در بعضی موارد بد نیست تا مقدار k را متناسب با تعداد تکرار های الگوریتم عوض کنیم تا در مراحل اولیه یادگیری عامل به جستجو بپردازد و در مراحل آخر نیز از اطلاعات استفاده کند.

---

[1] overcommit
[2] expoit
[3] explore

www.farabar.net

Telegram: @Farabar_BI

www.farabar.net

۱۳.۳.۶ سری تغییرها

یکی از اهمیت‌های قضیه‌ی همگرایی بالا این نکته است که یادگیری Q نیازی به دنبال کردن خط مشی بهینه برای یادگیری آن ندارد. در واقع می‌توان تابع Q (و متعاقباً خط مشی بهینه) را با دنبال کردن یک خط مشی کاملاً تصادفی در هر مرحله و بررسی تمامی جفت حالت و عمل‌ها هر چند وقت پیدا کرد. همین نکته به ما اجازه می‌دهد که بدون از بین بردن همگرایی الگوریتم بتوانیم نحوه‌ی انتخاب اعمال را برای بهینه سازی یادگیری تغییر دهیم. برای تصور، دوباره یادگیری MDP را با همان حالت جاذب شکل ۱۳.۱ را در نظر بگیرید. فرض کنید که هنوز عامل را با قسمت‌های یادگیری آموزش نداده‌ایم. برای هر قسمت عامل را در حالتی تصادفی قرار می‌دهیم تا اعمالی را انجام دهد و جدول $\hat{Q}$ را تغییر دهد تا به حالت جاذب برسد. و بعد از آن نیز دوباره عامل در حالتی تصادفی قرار می‌گیرد. همان طور که قبلاً نیز گفته شد اگر تمامی مقادیر جدول $\hat{Q}$ را صفر قرار دهیم بعد از مرحله‌ی اول تنها یک مقدار جدول تغییر کرده و آن نیز مقدار حالت قبلی حالت جاذب است. توجه داشته باشید که اگر عامل همان سری اعمال را طی کند یکی دیگر از مقادیر جدول $\hat{Q}$ نیز غیر صفر خواهد شد. حال اگر در تمامی قسمت‌ها همین اعمال را تکرار کنیم مقادیر حالت‌ها به ترتیب از حالت جاذب به سمت حالت اولیه تک تک با سرعت یک حالت در قسمت غیر صفر خواهند شد. حال همان آموزش را با ترتیب برعکس برای هر قسمت در نظر بگیرید. یعنی اینکه از همان رابطه‌ی ۱۳.۷ برای محاسبه‌ی $\hat{Q}$ استفاده می‌کنیم اما این ترتیب را برعکس انجام می‌دهیم. با چنین شرایطی در یک قسمت می‌توان تمامی مقادیر تخمینی $\hat{Q}$ را در طول این مسیر دلخواه به سمت حالت جاذب پیدا کرد. چنین فرایندی مسلماً در تکرار های کمتری همگرا خواهد شد اما با این وجود حافظه‌ی بیشتری برای ذخیره‌ی کل قسمت قبل از آموزش آن قسمت لازم است.

استراتژی دوم مطرح برای بهبود سرعت همگرایی ذخیره سازی جفت حالت و عمل‌های قبلی با پاداش‌های لحظه‌هایشان تکرار تناوبی عملیات مربوطه است. با وجود اینکه در نگاه اول چنین کاری هدر دادن زمان به نظر می‌رسد اما این کار باعث تصحیح $\hat{Q}(s,a)$ می‌شود زیرا که در اجرای اول این مقدار از $\hat{Q}(s',a)$ نشات گرفته (s′=δ(s,a)). در حالی که خود مقدار $\hat{Q}(s',a)$ بعد از آن دچار تغییرات می‌شود، پس تکرار دوباره‌ی جفت <s,a> باعث بهبود مقدار $\hat{Q}(s,a)$ خواهد شد. در کل، میزانی که به مسیر های قدیمی می‌پردازیم در مقابل میزانی که مسیر های جدید ایجاد می‌کنیم بستگی به هزینه‌ی نسبی این دو عملیات در قلمرو[1] مسئله دارد. مثلاً در قلمرو رباتی که حرکت‌هایش بسیار کند است، تأخیر ایجاد شده در پیدا کردن جفت حالت و عمل‌های جدید در جهان واقعی هزینه‌ی چندین برابر تکرار همان مسیر های قبلی دارد. این تفاوت باعث می‌شود که گاهی یادگیری Q بعد از چندین هزار تکرار حلقه همگرا شود.

توجه داشته باشید که در تمام طول بحث بالا فرضمان بر این بود که توابع δ(s,a) و r(s,a) برای عامل مجهول است. اگر این دو تابع برای عامل معلوم باشند متد های موثر تری را می‌توان بکار برد. برای مثال اگر اجرای اعمال خارجی هزینه بر است عامل می‌تواند از اثر خارجی آن چشم پوشی کرده و آن‌را شبیه سازی کند و در این محیط شبیه سازی شده با انجام اعمال و پاداشی که خود با استفاده از r در نظر می‌گیرد خود را آموزش دهد. (Sutton 1991) ساختار Dyna را معرفی می‌کند که بعد از انجام هر مرحله از آموزش در جهان واقعی تعدادی عمل را در جهان شبیه سازی شده انجام می‌دهد. (Moore and Atkeson 1993) متدی به نام prioritized sweeping را معرفی می‌کنند که زمانی که در ادامه فرایند به حالت‌های امیدوار کننده جدید می‌پردازد و فقط زمانی که تغییر بزرگی رخ داد به حالت‌های قبلی باز می‌گردد. (Peng and Williams 1994) نیز روشی مشابه را توصیف می‌کنند. زمانی که توابع δ و r معلومند، تعداد زیادی از الگوریتم‌های کارامد را می‌توان برای مبحث برنامه نویسی پویا به کاربرد. (Kaelbling 1996) تعدادی از این الگوریتم‌ها را بررسی می‌کند.

---

[1] Domain

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

۱۳.٤ اعمال و پاداش‌های غیر قطعی

در قسمت‌های قبلی یادگیری Q را برای محیط‌های قطعی بررسی کردیم. حال می‌خواهیم فرض کنیم که محیط غیر قطعی است، یعنی تابع r(s,a) و تابع δ(s,a) ممکن است خروجی‌های احتمالی داشته باشند. برای مثال در برنامه‌ی بازی تخته نردی که Tesauro (1995) طراحی کرده، خروجی اعمال ذاتاً احتمالی است زیرا که هر حرکت به پرتاب تاس وابسته است. همچنین در مسئله‌ی رباتی که حسگرها و اعمال نویز دار هستند لازم است که رابطه‌ی بین اعمال و پاداش‌ها را غیر قطعی فرض کنیم. در چنین شرایطی، دو تابع r(s,a) و δ(s,a) را می‌توان به صورت توزیع‌های احتمالی روی دو فضای s و a در نظر گرفت و سپس خروجی تصادفی‌ای با توجه به این توزیع‌ها بدست آورد. زمانی که این توزیع‌های احتمال منحصراً به s و a وابسته باشند (مثلاً به حالت قبلی یا عمل قبلی وابسته نباشند)، آنگاه به کل سیستم، سیستم تصمیم گیری غیر قطعی مارکوو[1] می‌گوییم.

در این قسمت الگوریتم یادگیری Q را برای حالت غیر قطعی در محیط‌های غیر قطعی MDP تامیم می‌دهیم. برای این کار، دوباره الگوریتم را برای حالت قطعی بررسی کرده و در موارد لازم آن‌را عوض می‌کنیم.

در حالت غیر قطعی، در ابتدا باید برای عامل در نظر بگیریم که دیگر خروجی‌ها قطعی نیستند. پس بدیهی است که رابطه‌ی V^{π} را برای خط مشی π به صورت امید پاداش تجمعی تخفیفی بیان کنیم (زیرا که دیگر هیچ چیز قطعی نیست). پس داریم:

$$V^{\pi}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

که در این رابطه، همان طور که قبلاً نیز گفته شد، سری r_{t+i} پاداش‌های دریافتی خط مشی π با شروع از حالت s است. توجه داشته باشید که این همان تامیم رابطه‌ی ۱۳.۱ برای حالت غیر قطعی است.

همان طور که قبلاً نیز گفته شد، خط مشی بهینه π^* خط مشی‌ای مثل π است که مقدار $V^{\pi}(s)$ را برای تمامی حالت‌های s ماکزیمم می‌کند. در گام بعدی تعریف Q را که در رابطه‌ی ۱۳.۴ آمده تامیم می‌دهیم:

$$\begin{aligned} Q(s,a) &\equiv E[r(s,a) + \gamma V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma E[V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \qquad (13.8) \end{aligned}$$

که در آن $P(s'|s,a)$ احتمال رسیدن به حالت s' پس از انجام عمل a در حالت s است. توجه داشته باشید که ما در اینجا از $P(s'|s,a)$ برای بازنویسی امید مقدار $V^*(\delta(s,a))$ استفاده کرده ایم.

درست مثل روابط قبلی برای تعریف Q نیز خواهیم داشت:

[1] nondeterministic Markov decision process

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \qquad (13.9)$$

که این رابطه نیز تامیم یافته‌ی رابطه‌ی ۱۳.۶ است. به طور خلاصه، تعریف جدید Q(s,a) امید مقدار تعریف شده‌ی حالت قطعی است.

حال که تعریف Q را باز نویسی کردیم، باید به سراغ بازنویسی قانون آموزش برویم. قانون آموزش قبلی‌ای که از بازنویسی قانون قبلی (رابطه‌ی 13.7) بدست می‌آید برای همگرایی در شرایط غیر قطعی دچار مشکل می‌شود. فرض کنید، برای مثال، که r(s,a) تابعی غیر قطعی است که هر بار پاداش‌های متفاوتی برای زوج <s,a> می‌دهد. در چنین شرایطی، رابطه‌ی قبلی متناوباً مقدار $\hat{Q}(s,a)$ را حتی اگر مقدار $\hat{Q}(s,a)$ مقدار درست تابع Q باشد تغییر خواهد داد. به طور خلاصه، قانون آموزش قبلی در چنین شرایطی همگرا نمی‌شود. برای حل چنین مشکلی باید در قانون قدیمی تغییراتی را انجام داد، قانون جدید در هر مرحله باید به جای از بین بردن تخمین قبلی میانگینی وزن دار بین مقدار تخمینی جدید و مقادیر قبلی محاسبه کند. با تغییر رابطه به فرم زیر شرایط لازم برای همگرایی $\hat{Q}$ به Q فراهم می‌شود:

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')] \qquad (13.10)$$

که

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \qquad (13.11)$$

در رابطه‌ی بالا s و a حالت و عملی هستند که در تکرار n ام حلقه رخ می‌دهند و $visits_n(s,a)$ تعداد تمامی جفت حالت و عمل‌هایی است که تا تکرار n ام بررسی شده‌اند.

نکته‌ی کلیدی در این قانون این است که $\hat{Q}$ در این رابطه کند تر از قانون قبلی تغییر می‌کند. توجه داشته باشید که اگر مقدار α_n در رابطه‌ی ۱۳.۱۰ یک قرار داده شود به همان رابطه‌ی آموزش قدیمی می‌رسیم. با کمتر کردن مقدار α، این جمله میانگین $\hat{Q}$(s,a) و جدید تر خواهد بود. توجه دارید که با افزایش n در رابطه‌ی ۱۳.۱۱ α کاهش می‌یابد، پس تغییرات با پیشرفت آموزش به تدریج کمتر خواهد شد. با کم کردن α با سرعت مناسب در طول آموزش می‌توان به تابع Q میل کرد. انتخاب α_n ی که در بالا آمده با توجه به قضیه‌ی زیر فقط یکی از چندین شرط همگرایی است (Watkins and Dayan 1992).

قضیه‌ی ۱۳.۲. همگرایی یادگیری Q برای فرایند تصمیم گیری غیر قطعی مارکوو. فرض کنید که عامل یادگیری Q در یک محیط غیر قطعی MDP قرار گرفته است که پاداش‌ها نیز کران دارند $(\forall s,a)|r(s,a)| \le c$. عامل یادگیری Q با رابطه‌ی 13.10 و مقدار اولیه‌ی دلخواه کران دار اولیه‌ی جدول $\hat{Q}(s,a)$ و عامل تخفیف $0 \le \gamma < 1$ سعی می‌کند تا Q را تخمین بزند. اگر n(I,s,a) شماره‌ی تکراری باشد که عمل a از حالت s برای i امین بار اجرا می‌شود و اگر تمامی جفت حالت و عمل‌ها هر چند وقت یک بار بررسی شوند و نیز $0 \le \alpha_n < 1$ و داشته باشیم

$$\sum_{i=1}^{\infty} \alpha_{n(i,s,t)} = \infty, \sum_{i=1}^{\infty} [\alpha_{n(i,s,t)}]^2 < \infty$$

برای تمامی s و a ها اگر n→∞ با احتمال ۱ خواهیم داشت که $\hat{Q}(s,a) \to Q(s,a)$.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

با وجود اینکه ثابت می‌شود که یادگیری Q و دیگر الگوریتم‌های یادگیری تقویتی تحت شرایطی همگرا می‌شوند، اما در عمل چند هزار تکرار حلقه‌ی اصلی لازم است تا به میزان مطلوب همگرا شوند. برای مثال در بازی TD-Gammon که توسط (Tesauro) مطرح شده، که قبلاً به آن اشاره کردیم، ۱.۵ میلیون بازی مختلف وجود دارند که هر کدام نیز ده‌ها جفت حالت و عمل دارند.

۱۳.۵ یادگیری اختلاف ارزش‌ها

یادگیری Q با کم کردن تفاوت مقدار تخمینی و مقدار اصلی Q در هر تکرار حلقه، Q را یاد می‌گیرد. از این نظر یادگیری Q یک حالت خاص از دسته الگوریتم‌های کاهش اختلاف ارزش‌ها[1] است که از طریق کم کردن تفاوت بین تخمین عامل و تابع اصلی در چندین مرحله تابع هدف را تخمین می‌زنند. همان طور که قانون آموزش ۱۳.۱۰ تفاوت بین مقدار تخمینی $\hat{Q}$ برای حالت ابتدایی و حالت پایانی یک عمل را کم و زیاد می‌کند می‌توانیم الگوریتم‌هایی را طراحی کنیم تا اختلاف بین حالت‌های چند عمل قبل و حالت‌های چند عمل بعد را نیز کم و یا زیاد کند.

برای بررسی بیشتر، محاسبات رابطه‌ی یادگیری Q را که برای محاسبه‌ی مقدار $\hat{Q}(s_t, a_t)$ با استفاده از مقدار $\hat{Q}(s_{t+1}, a_{t+1})$ آورده شده را در نظر بگیرید (که در آن s_{t+1} نتیجه‌ی انجام عمل a_t در حالت s_t است). این رابطه را با نگاه یک مرحله ای را با $\hat{Q}^{(1)}(s_t, a_t)$ نیز نشان می دهند:

$$\hat{Q}^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a_t)$$

می‌توان بجای این نگاه یک مرحله ای به پاداش‌ها نگاه دو مرحله ای داشت:

$$\hat{Q}^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a_t)$$

یا در حالت کلی نگاهی n مرحله ای به پاداش‌ها داشت:

$$\hat{Q}^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a_t)$$

(Sutton 1988) متدی کلی برای ترکیب این روابط تخمینی جایگزین به نام TD(λ) معرفی می‌کند. ایده‌ی اصلی TD(λ)، استفاده از ثابت $0 \le \lambda \le 1$ برای ترکیب این جایگزین‌هاست:

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda)\left[\hat{Q}^{(1)}(s_t, a_t) + \lambda \hat{Q}^{(2)}(s_t, a_t) + \lambda^2 \hat{Q}^{(3)}(s_t, a_t) + \cdots\right]$$

به صورت بازگشتی خواهیم داشت که :

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})]$$

حال اگر λ=0 بگیریم به همان رابطه‌ی $\hat{Q}^{(1)}$ خواهیم رسید که فقط یک مرحله پاداش را در تخمین نظر می‌گیرد. با افزایش مقدار λ الگوریتم به سمتی می‌رود تا اختلاف تخمین را نیز در مراحل بعدی کم کند. در نهایت امر زمانی که λ=1 است فقط مقادیر r_{t+i} در نظر گرفته می‌شوند

[1] temporal difference algorithms

www.farabar.net

Telegram: @Farabar_BI

و تخمین فعلی نیز تأثیری در رابطه نخواهد داشت. توجه داشته باشید که اگر $\hat{Q} = Q$ باشد قانون ذکر شده برای Q^{λ} برای تمامی مقادیر $0 \leq \lambda \leq 1$ ایده آل خواهد بود.

انگیزه‌ی ایجاد TD(λ) این است که در بعضی شرایط آموزش اگر با عمق بیشتر نگاه کنیم آموزش موثر تر خواهد شد. برای مثال، زمانی که عامل از خط مشی بهینه پی روی می‌کند، با مقدار λ=1 تخمین Q^{λ}بدون توجه به اشتباهات موجود در $\hat{Q}$ عالی‌ای از مقادیر واقعی تابع Q به ما می‌دهد. از طرف دیگر اگر اعمال به صورت ناحیه ای بهینه انتخاب شوند، r_{t+i} های بدست آمده ممکن است با توجه به آینده گمراه کننده باشند.

(Peng and Williams 1994) بحثی گسترده تر را به همراه نتایج آزمایشی که کارایی Q^{λ} را در قلمروی مسئله‌ی خاصی نشان می‌دهد ارائه می‌کنند. (Dayan 1992) نیز نشان می‌دهد که تحت شرایط خاصی روشی مشابه TD(λ) برای تمامی مقادیر $0 \leq \lambda \leq 1$ به تابع V^* میل می‌کند. (Tesauro 1995) از روش TD(λ) در برنامه‌ی TD-Gammon برای بازی تخته نرد[1] استفاده می‌کند.

۱۳.۶ تعمیم روی نمونه‌ها

شاید یکی از محدودکننده‌ترین فرض‌هایی که تا به حال در مورد یادگیری Q کردیم این بود که Q به صورت تابعی جدولی در نظر گرفته می‌شد که برای هر ورودی خاص (مثلاً یک زوج حالت و عمل) یک خروجی خاص داشت. پس الگوریتم‌هایی که تا به حال به کار بردیم چیزی شبیه یک حافظه معمولی[2] انجام می‌دهند و هیچ تلاشی برای تخمین مقادیر دیگر Q نمی‌کنند. این فرض در اثبات همگرایی نیز خود را نشان داده است، تنها زمانی الگوریتم همگرا خواهد شد که هر جفت حالت و عمل بررسی شوند (آن هم هر چند وقت یک بار!). چنین فرضی در فضاهای بزرگ و نامحدود فرضی کاملاً غیر واقعی است، یا حداقل، اجرای آن هزینه و وقت زیادی می‌برد. همین باعث شده تا در اکثر سیستم‌های کاربردی ترکیبی از یادگیری Q و تخمین توابع در فصل‌های گذشته شد مورد استفاده قرار می‌گیرد.

ترکیب الگوریتم‌های تخمین توابع مثل Backpropagation و یادگیری Q بسیار ساده است، به راحتی می‌توان از مقادیر جدول $\hat{Q}$ را برای آموزش شبکه ای عصبی استفاده کرد. برای مثال، می‌توان با کد کردن حالت s و عمل a و ورود آن‌ها به شبکه و گرفتن مقدار $\hat{Q}$ از خروجی آن و آموزش شبکه با استفاده از نمونه های جدول $\hat{Q}$ و روابط ۱۳.۷ و ۱۳.۱۰ شبکه ای برای تخمین $\hat{Q}$ ساخت. روش دیگر که در کاربرد موفق‌تر بوده، آموزش شبکه های مجزایی برای هر عمل با ورودی حالت و خروجی $\hat{Q}$ است. یکی دیگر از راه‌های مرسوم آموزش شبکه ای است که حالت را به عنوان ورودی می‌گیرد و برای هر عمل نیز یک مقدار $\hat{Q}$ خروجی می‌دهد. توجه دارید که در فصل اول در صفحه‌ی بازی checkers تابع ارزیابی را با استفاده از LMS با تابعی خطی تخمین زدیم.

در عمل، سیستم‌های یادگیری تقویتی موفقی را می‌توان با جایگزینی الگوریتم‌های تخمین به جای جدول ایجاد کرد. در برنامه‌ی TD-Gammon که ساخته‌ی Tesauro است برای بازی تخت نرد از شبکه ای عصبی و الگوریتم Backpropagation و قانون TD(λ) استفاده شده است. (Zhang and Dietterich 1996) از ترکیبی مشابه از Backpropagation و TD(λ) برای برنامه ریزی

[1] backgammon
[2] Rote learning

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

مغازه ای[1] استفاده کرده‌اند. (Crites and Barto 1996) روشی تقویتی با شبکه‌ی عصبی برای برنامه ریزی یک آسانسور را مورد استفاده قرار دادند. (Thrun 1996) نیز از یادگیری Q بر اساس خوشه یابی دسته حالت‌ها برای مسئله کنترل ساده‌ی ربات استفاده کرده است.

با وجود موفقیت در این سیستم‌ها، یادگیری تقویتی در کارهایی دیگر با شکست روبرو شده است، تابع همگرا نشده است. مثال‌هایی از این شکست‌ها در (Boyan and Moore 1995) و (Baird 1995) و (Gordon 1995) آمده است. توجه دارید که قضایای همگرایی مطرح شده در این فصل فقط برای زمانی برقرار است که $\hat{Q}$ با جدولی از داده‌ها نمایش داده شود. برای درک بهتر، فرض کنید به جای جدول از شبکه ای عصبی بجای جدول $\hat{Q}$ استفاده شود. توجه دارید که اگر یادگیر شبکه را برای یادگیری بهتر Q برای نمونه‌ی خاصی چون $< s_i, a_i >$ تغییر دهد، ممکن است تخمین‌های $\hat{Q}$ دیگر زوج مرتب‌ها را نیز دست‌خوش تغییر کند. زیرا که ممکن است این تغییرات خطای $\hat{Q}$ را در تخمین دیگر زوج مرتب‌ها افزایش دهد، و ویژگی لازم بنا قضیه‌ی اثبات همگرایی دیگر تضمین شده نیست. بررسی‌های تئوری یادگیری تقویتی با تعمیم توابع تخمینی در (Gordon 1995) و (Tsitsiklis 1994) آورده شده است. (Baird 1995) متد های شیب نزول را که خطای کل نمونه های آموزشی را در نظر می‌گیرند را برای رفع این مشکلات (عدم همگرایی) پیشنهاد می‌کند (این روش خطای باقیماندهی بلمن[2] نیز نامیده می‌شود).

۱۳.۷ رابطه با برنامه نویسی پویا

متد های یادگیری تقویتی همچون یادگیری Q رابطه ای بسیار نزدیک با تحقیقاتی در برنامه نویسی پویا برای حل مسائل تصمیم گیری مارکوو دارند. در چنین مسائلی فرض می‌کنند که عامل اطلاعات کاملی درباره‌ی توابع δ(s,a) و r(s,a) که محیط را تعریف می‌کنند دارد. بنابراین، با فرض اینکه محیط کاملاً قابل شبیه سازی است و نیازی به تعامل مستقیم با محیط نیست اصولاً این سؤال مطرح می‌شود چگونه با کمترین تلاش محاسباتی به خط مشی بهینه برسیم؟ ویژگی طولانی کننده‌ی یادگیری Q این بود که فرض می‌کرد عامل هیچ علمی درباره‌ی توابع δ(s,a) و r(s,a) ندارد و باید به جای حرکت در شبیه سازی باید در محیط واقعی به حرکت و مشاهده عکس‌العمل محیط بپردازد. در شرایطی که ذکر شد اولویت اول ما معمولاً کم کردن تعداد حرکاتی است که عامل در جهان واقعی انجام می‌دهد تا به خط مشی قابل قبولی همگرا شود، کم کردن تعداد تکرار حلقه‌ی محاسبات در اولویت‌های بعدی است. دلیل چنین اولویت بندی‌ای این است که در بسیاری از زمینه های کاربردی مثل مسائل تولید، هزینه‌ی اعمال در جهان واقعی پول و زمان است که از هزینه‌ی مصرفی برای انجام محاسبات بسیار بیشتر است. سیستم‌هایی که در محیط واقعی عمل می‌کنند و نتایج اعمال را مشاهده می‌کنند اصطلاحاً سیستم‌های online نامیده می‌شوند در حالی که سیستم‌هایی که در محیط‌های شبیه سازی عمل می‌کنند سیستم‌های offline نامیده می‌شوند.

ارتباط بین روش‌های قبلی و یادگیری تقویتی که در اینجا ذکر شد توسط رابطه‌ی بلمن (Bellman) آشکار می‌گردد، این رابطه پایه‌ی بسیاری از روش‌های برنامه نویسی پویا برای حل MDP هاست. رابطه‌ی بلمن در زیر نوشته شده:

$$(\forall s \in S) V^*(s) = E[r(s, \pi(s)) + \gamma V^*(\delta(s, \pi(s)))]$$

[1] job-shop schedualing
[2] Bellman residual errors

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

به رابطه‌ی نزدیک بین رابطه‌ی بلمن و رابطه ای که قبلاً برای خط مشی بهینه تعریف کردیم (رابطه‌ی ۱۳.۲) توجه داشته باشید. بلمن (Bellman 1957) نشان داد که خط مشی بهینه‌ی π^* در رابطه‌ی بالا صدق می‌کند و هر رابطه ای که در رابطه‌ی بالا صدق کند نیز خط مشی ی بهینه است. از جمله اولین بررسی‌ها درباره‌ی برنامه نویسی پویا می‌توان الگوریتم کوتاه‌ترین مسیر بلمن–فورد (Bellman 1958; Ford and Fulkerson 1962) را نام برد که یاد می‌گرفت تا کوتاه‌ترین مسیر به سمت هدف در یک گراف را برای هر گره گراف بر اساس طول مسیر های گره های همسایه‌اش تخمین می‌زد. در این الگوریتم این فرض که یال‌های گراف و گره هدف معلومند معادل فرض ما در معلوم بودن دو تابع δ(s,a) و r(s,a) است. (Barto 1995) و مقالات دیگر نیز درباره‌ی رابطه‌ی نزدیک برنامه نویسی پویا و یادگیری تقویتی بحث کرده‌اند.

۱۳.۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی مطرح شده در این فصل به شرح زیر است:

- یادگیری تقویتی مسئله‌ی یادگیری استراتژی کنترل برای عامل‌های خودکار[1] را حل می‌کند. این نوع یادگیری فرض می‌کند که داده های آموزشی به فرم سیگنال پاداش حقیقی مقداری به ازای هر جفت حالت و عمل به یادگیر داده می‌شود. هدف عامل یادگیری خط مشی‌ای که کل پاداش دریافتی را مستقل از نقطه‌ی شروع حداکثر کند.
- الگوریتم‌های یادگیری تقویتی‌ای که در این فصل آورده شده‌اند، برای تعریف مسئله‌ی معروفی به نام فرایند تصمیم گیری مارکوو ارائه شده است. در فرایند تصمیم گیری مارکوو، نتیجه‌ی حاصل از اعمال یک عمل به یک حالت فقط به حالت و عمل انجام شده وابسته است (و به اعمال قبلی و حالات قبلی وابستگی ندارد). تعریف مسئله‌ی فرایند تصمیم گیری مارکوو مسائل زیادی را از جمله بسیاری از مسائل کنترل ربات، اتوماسیون کارخانه و مسائل برنامه ریزی را در بر می‌گیرد.
- یادگیری Q یکی از فرم‌های یادگیری تقویتی است که عامل در آن تابعی بر روی حالات و اعمال را یاد می‌گیرد. در کل، تابع ارزیابی Q(s,a)، امید ماکزیمم پاداش تخفیفی تجمعی قابل دریافت برای عامل با اعمال عمل a به حالت s تعریف می‌شود. الگوریتم یادگیری Q این مزیت را دارد که حتی زمانی که عامل دانش قبلی‌ای در مورد تأثیر عملش بر محیط ندارد هم قابل استفاده است.
- ثابت می‌شود که یادگیری Q به شرط آنکه فرضیه یادگیر از $\widehat{Q}(s,a)$ جدولی از داده های خام <s,a> باشد به تابع Q درست میل می‌کند. این قضیه هم در حالت قطعی و هم در حالت احتمالی MDP درست است. در عمل یادگیری Q حتی در مسائلی با معتدل‌ترین اندازه ممکن است پس از هزاران حلقه همگرا شود.
- یادگیری Q عضوی از کلاس گسترده تری از الگوریتم‌ها به نام الگوریتم‌های کاهش اختلاف ارزش‌ها محسوب می‌گردد. در کل، الگوریتم‌های کاهش اختلاف ارزش‌ها با کاهش تناوبی اختلافات بین تخمین عامل و واقعیت یاد می‌گیرند.
- یادگیری تقویتی بسیار نزدیک به برنامه نویسی پویا در حل فرایند تصمیم گیری مارکوو است. تفاوت کلیدی در این است که برنامه نویسی پویا فرض می‌کند که اطلاعات لازم از $\delta(s,a)$ و $r(s,a)$ را دارد. در مقابل یادگیری تقویتی در کل فرض می‌کند که یادگیر از داشتن چنین موهبتی محروم است.

[1] autonomous

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

موضوع متداولی که زیرساخت اکثر کارهای یادگیری تقویتی است، کم کردن اختلاف بین ارزیابی‌های حالت‌های موفق در هر حلقه است. بعضی از تلاش‌های اولیه روی چنین متدی توسط (Samuel 1959) انجام گرفت. برنامه‌ی یادگیری بازی چکرز وی سعی دارد تا تابع ارزیابی‌ای با استفاده از ارزیابی حالت‌های انتهایی برای ایجاد مقادیر آموزشی برای حالت‌های اولیه ایجاد کند. تقریباً در همان موقع، الگوریتم‌های Bellman-Ford، برای کوتاه‌ترین مسیر با هدف معلوم توسط (Bellman 1958; Ford and Fulkerson 1962) ایجاد گشتند، این الگوریتم‌ها مقادیر متغیرهای فاصله تا هدف را از گره‌ها به همسایه‌هایشان سرایت می‌دهند. تحقیق روی کنترل بهینه برای حل فرایند های مارکوو به متد های مشابه ای رسید (Bellman 1961; Blackwell 1965). متد bucket brigade (Holland 1986) نیز برای یادگیری سیستم‌های دسته بندی از متدی مشابه که امتیاز را در بین پاداش‌های تأخیری پخش می‌کند استفاده می‌کند. (Barto et al. 1983) روشی برای اختصاص امتیاز زمانی ارائه کرده که در نهایت به مقاله (Sutton 1988) ختم شد که TD(λ) را معرفی کرد و همگرایی آن را برای λ=0 اثبات کرد. (Dayan 1992) نیز این نتیجه را به مقادیر دلخواه λ تعمیم داد. (Watkins 1989) یادگیری Q را برای رسیدن به خط مشی بهینه زمانی که توابع پاداش و انتقال نامعلومند ارائه داد. اثبات همگرایی برای بسیاری از حالات مختلف این متدها ارائه گردیده است. علاوه بر اثباتی که در این فصل ارائه گردید، می‌توانید به (Baird 1995; Bersekas 1987; Tsitsiklis 1994; Singh and Sutton 1996) مراجعه کنید.

یادگیری تقویتی همچنان زمینه‌ی تحقیق فعالی است. برای مثال، (McCallum 1995) و (Littman 1996) تعمیم یادگیری تقویتی را به تعریف مسئله ای با متغیر های حالت غیر قابل مشاهده را بحث کردند، این تعریف مسئله خارج از تعریف مسئله‌ی فرایند تصمیم مارکوو است. تحقیق زیادی برای تعمیم این متدها روی مسائل بزرگ‌تر و کاربردی‌تر انجام می‌شود. (Maclin and Shavlik 1996) روشی ارائه کردند که یادگیر تقویتی بتواند از مربی توصیه های ناکامل دریافت کند، این روش بر اساس تعمیمی از KBANN است (فصل ۱۲). (Lin 1992) نیز نقش آموزش را با سری اعمال پیشنهادی بررسی می‌کند. متدهایی نیز برای تعمیم و اعمال سلسله مراتبی اعمال توسط (Singh 1993) و (Lin 1993) ارائه شده است. (Dietterich and Flann 1995) انتگرال‌گیری متدهای مبتنی بر توضیحات را با یادگیری تقویتی بحث کرده و (Mitchell and Thrun 1993) کارایی الگوریتم EBNN (فصل ۱۲) را بر روی یادگیری Q بحث می‌کنند. (Ring 1994) یادگیری پیوسته یادگیر را روی کارهای چندگانه بررسی کرده است.

تحقیقات جدید روی یادگیری تقویتی در (Kaelbling et al. 1996)؛ (Barto 1992)؛(Barto et al. 1995) و (Dean et al. 1993) آمده است.

تمرینات

۱۳.۱ خط مشی بهینه‌ی دیگری برای مسئله‌ی نشان داده شده در شکل ۱۳.۲ بیابید.

۱۳.۲ جهان قطعی نشان داده شده زیر را با حالت جاذب هدف G در نظر بگیرید. در این جا تمامی فلش‌هایی که عدد ۱۰ دارند پاداش ۱۰ و بقیه اعمال پاداش ۰ دارند.

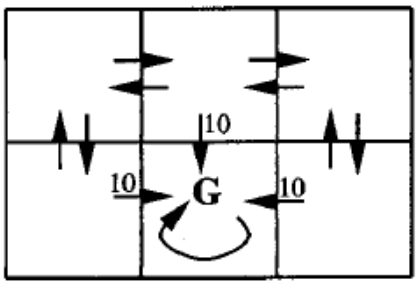
V^* (a) را برای تمامی حالات شکل مشخص کنید. برای تمامی حرکت‌های ممکن Q(s,a) را محاسبه کنید. در انتها نیز خط مشی بهینه ای را ارائه کنید. فرض کنید که γ=0.8.

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

(b) تغییری در r(s,a) پیشنهاد کنید که مقادیر Q(s,a) را تغییر دهد اما در خط مشی بهینه تأثیری نداشته باشد. تغییری در r(s,a) پیشنهاد کنید که Q(s,a) را تغییر داده اما تأثیری بر $V^*(s,a)$ نداشته باشد.

(c) حال می‌خواهیم یادگیری Q را به این جهان با فرض اینکه مقادیر اولیه‌ی $\hat{Q}$ صفرند اعمال کنیم. فرض کنید که عامل از گوشه پایین سمت چپ شروع و ساعت‌گرد به حرکت خود ادامه می‌دهد تا به حالت جاذب برسد. چه مقادیری از $\hat{Q}$ تغییر خواهد کرد و این تغییرات چه هستند. حال همین سؤال را برای تکرار همین دور برای بار دوم و سوم جواب دهید.



۱۳.۳ بازی دوز[1] را در مقابل بازیکنی که به صورت تصادفی بازی می‌کند را در نظر بگیرید. در عمل فرض کنید که حریف با توزیع یکنواخت یکی از خانه های خالی را انتخاب می‌کند، مگر اینکه مجبور به انتخاب خانه‌ی دیگر باشد (که به وضوح خانه درست را انتخاب خواهد کرد).

(a) مسئله یادگیری استراتژی بهینه‌ی بازی دوز را در این شرایط برای یادگیری Q را دقیق بیان کنید. در این فرایند تصمیم گیری غیر قطعی مارکوو مجموعه های حالات اعمال و پاداش چه هستند؟

(b) آیا برنامه‌ی شما در صورتی که حریف بهینه بازی کند نیز موفق خواهد بود؟

۱۳.۴ توجه داشته باشید که در بسیاری از مسائل MDP به سادگی می‌توان دو خط مشی مختلف مثل π_1 و π_2 را پیدا کرد که اگر عامل از حالت s_1 شروع به حرکت کند خط مشی π_1 بهینه تر باشد و اگر از حالت s_2 شروع کند π_2 بهینه تر باشد. به عبارت دیگر، $V^{\pi_1}(s_1) > V^{\pi_2}(s_1)$ اما $V^{\pi_2}(s_2) > V^{\pi_1}(s_2)$. توضیح دهید که چرا همیشه خط مشیی (مثل π^*) وجود دارد که برای تمامی حالات اولیه ی s $V^{\pi}(s)$ ماکزیمم است. به عبارت دیگر، توضیح دهید که چرا یک MDP همیشه اجازه می‌دهد که خط مشی‌ای مثل π^* باشد که $(\forall \pi, s) V^{\pi^*}(s) \geq V^{\pi}(s)$.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

استفاده از دانسته‌ها برای بدست آوردن پاداش	Expoit
اعمال	Action
بازی‌های صفحه ای	board games
برنامه نویسی پویا	dynamic programming

[1] tic-tac-toe

www.farabar.net

Telegram: @Farabar_BI
www.farabar.net

پاداش تأخیری	delayed reward
پاداش تجمعی تخفیفی	discounted cumulative reward
پاداش لحظه ای	imidiate reward
تابع پاداش	reward function
تجمعی	cumulative
ثابت تخفیف	discount factor
جستجوی محیط	Exploration
جستجوی محیط	Explore
خط مشی	Policy
خط مشی بهینه	optimal policy
فرایند تصمیم گیری غیر قطعی مارکوو	nondeterministic Markov decision process
عامل	Agent
غیر قطعی	Nondeterministic
قلمرو	Domain
قطعی	Deterministic
الگوریتم‌های کاهش اختلاف ارزش‌ها	temporal difference algorithms
متوسط پاداش	average reward
محیط	Envirement
مربی	Trainer
نسبت دادن ارزش موقتی	temporal credit assignment
حالت	State
حالت جاذب	absorbing state

www.farabar.net

ضمیمه

نمادگذاری

در زیر خلاصه ای از نمادگذاری این کتاب آورده شده است:

[a,b): کروشه و پرانتز برای نشان دادن بازه ها به کار می روند، کروشه ها نشان دهنده ی این است که بازه شامل مقدار مرز است و پرانتز نشان دهنده ی این است که بازه مرز را شامل نمی شود.

برای مثال [1,3) یعنی $1<x\leq 3$.

$\sum_{i=1}^{n} x_i$: مجموع $x_1 + x_2 + \cdots + x_n$.

$\prod_{i=1}^{n} x_i$: حاصلضرب $x_1 . x_2 . \ldots . x_n$.

$\vdash$: نماد نتیجه گیری منطقی. برای مثال $A \vdash B$ یعنی که B از A نتیجه گیری منطقی می شود.

$\succ_g$: نماد کلی تر بودن. برای مثال $h_i \succ_g h_j$ نشان می دهد که h_i از h_j کلی تر است.

$\text{argmax}_{x\in X} f(x)$: مقدار x است که f(x) در آن ماکزیمم می شود. برای مثال،

$\hat{f}(x)$: تابعی که تابع f(x) را تخمین می زند.

δ: در یادگیری PAC، مرزی بر روی احتمال شکست است. در شبکه های عصبی نیز، جمله ی خطای مربوطه ی خروجی یک تک واحد است.

ϵ: مرز خطا فرضیه (در یادگیری PAC).

Telegram: @Farabar_BI
www.farabar.net

η: ضریب یادگیری در شبکه های عصبی و متد های یادگیری مربوطه.

μ: میانگین توزیع احتمال.

σ: انحراف معیار توزیع احتمال.

$\nabla E(\vec{w})$: گرادیان E نسبت به بردار $\vec{w}$.

C: کلاس توابع هدف ممکن

D: داده های آموزشی

$\mathcal{D}$: توزیع احتمال روی فضای نمونه ای.

E[x]: مقدار امید x.

$E(\vec{w})$: مجموع خطاهای مربعی شبکه ی عصبی با بردار وزنهای $\vec{w}$.

Error: خطای فرضیه گسسته مقدار یا پیشبینی.

H: فضای فرضیه ای.

h(x): پیشبینی فرضیه ی h برای نمونه ی x.

P(x): احتمال x.

Pr(x): احتمال اتفاق x.

p(x): چگالی توزیع احتمال x.

Q(s,a): تابع Q در یادگیری تقویتی.

$\Re$: مجموعه ی اعداد حقیقی.

VC(H): بعد Vapnik-Chervonenkis فضای فرضیه ای H.

$VS_{H,D}$: فضای ویژه؛ مجموعه ای از فرضیه هایی از H که با D سازگارند.

w_{ij}: در شبکه های عصبی وزن از گره i به گره j است.

X: فضای نمونه ای.

www.farabar.net