Chương 5
Machine Learning cơ bản

DL là một phạm trù đặc biệt của ML. Để hiểu sâu DL ta phải có một nền tảng ML cơ bản. Chương này sẽ **giới thiệu** ngắn gọn các nguyên lý chung quan trọng nhất trong ML được áp dụng trong phần còn lại của cuốn sách. Với những bạn mới làm quen hoặc muốn tìm hiểu sâu hơn về ML, chúng tôi khuyên bạn nên tìm đọc một số cuốn sách giáo khoa ML với mức độ bao phủ kiến thức nền tảng toàn diện hơn, chẳng hạn như [Murphy, 2012] hoặc [Bishop, 2006]. Nếu đã quen thuộc với những kiến thức ML cơ bản, bạn có thể chuyển ngay đến phần 5.11, ở đó sẽ trình bày một số kỹ thuật ML truyền thống đã góp phần thúc đẩy sự phát triển của các thuật toán DL.

Ta sẽ bắt đầu bằng việc định nghĩa một *learning algorithm* là gì, và xem xét một ví dụ: thuật toán *hồi quy tuyến tính* (*linear regression*). Tiếp đó ta sẽ mô tả các thách thức trong việc tạo ra mô hình khớp (*fit*) với dữ liệu huấn luyện sẽ *khác biệt* thế nào so với các thách thức trong việc tìm các mô thức giúp tổng quát hoá dữ liệu mới. Hầu hết các thuật toán ML đều có các thiết lập gọi là các *siêu tham số* (hyper-parameters) mà ta phải xác định bên ngoài *learning algorithm*; ta sẽ trình bày về cách thiết lập những siêu tham số này bằng cách sử dụng dữ liệu bổ sung. Về cơ bản, ML là một dạng của lĩnh vực thống kê ứng dụng, một lĩnh vực đang ngày càng hướng tâm điểm vào ứng dụng trong việc sử dụng máy tính để ước lượng thống kê các hàm phức tạp, và giảm trọng tâm vào việc chứng minh khoảng tin cậy xung quanh các hàm này. Do đó, chúng tôi sẽ trình bày hai phương pháp tiếp cận trọng tâm của thống kê: *các ước lượng tần suất* (*frequentist estimators*) và *suy luận Bayes* (*Bayesian inference*). Các thuật toán ML có thể được phân loại thành *học có giám sát* (*supervised learning*) và *học không giám sát* (*unsupervised learning*); chúng tôi sẽ mô tả hai cách học này và đưa ra một vài ví dụ đơn giản cho mỗi loại. Hầu hết các thuật toán DL được dựa trên một thuật toán tối ưu hóa gọi là *stochastic gradient descent*. Ngoài ra, chúng tôi cũng mô tả cách kết hợp các thuật toán thành phần, như thuật toán tối ưu, hàm mất mát, mô hình (*model*) và tập dữ liệu (*dataset*) để xây dựng một thuật toán ML. Cuối cùng, trong phần 5.11, chúng tôi mô tả một số yếu tố làm hạn chế khả năng của ML truyền thống trong việc khái quát hoá dữ liệu, cũng là động lực

góp phần thúc đẩy sự phát triển các thuật toán DL nhằm vượt qua những trở ngại này.

5.1 Learning algorithms

Một thuật toán học máy là thuật toán có khả năng học từ dữ liệu. Nhưng ta đang ám chỉ học ở đây là học cái gì?

[Mitchell, 1997] đưa ra một định nghĩa ngắn gọn: “Một chương trình máy tính được cho là học hỏi từ kinh nghiệm E (*experience*) đối với một tập tác vụ T (*task*) và phương pháp đo lường hiệu năng P (*performance*), nếu hiệu năng của nó khi thực hiện tác vụ T, được đo bằng phương pháp P, được cải thiện với kinh nghiệm được rút ra từ E". Người ta có thể hình dung ra rất nhiều loại kinh nghiệm E, tác vụ T, và các phương pháp đo đạc P, nhưng trong khuôn khổ cuốn sách này, ta không tìm cách định nghĩa một cách chặt chẽ những gì có thể được sử dụng cho mỗi thành tố này. Thay vào đó, trong các phần sau, ta sẽ cố gắng cung cấp các mô tả trực quan và các ví dụ về các loại tác vụ, phương pháp đo lường hiệu năng và các kinh nghiệm có thể được sử dụng để xây dựng các thuật toán ML.

5.1.1 Tác vụ T

ML cho phép ta xử lý những tác vụ quá phức tạp được giải quyết bằng các chương trình cố định do con người tự viết và thiết kế. Từ quan điểm khoa học và triết học, ML là một lĩnh vực thú vị, hiểu về ML đòi hỏi ta cần hiểu biết về các nguyên tắc nền tảng của trí thông minh.

Trong định nghĩa khá hình thức này của từ “tác vụ”, quá trình tự học không phải là một tác vụ. Học là phương tiện để giúp ta đạt được khả năng thực hiện các tác vụ. Chẳng hạn, nếu ta muốn một robot có thể đi bộ, thì việc đi bộ là tác vụ. Ta có thể lập trình để robot học cách đi bộ, hoặc có thể cố gắng trực tiếp, tự tay viết một chương trình đặc tả cách đi.

Các tác vụ trong ML thường được mô tả như là cách hệ thống ML xử lý một *mẫu dữ liệu* (*example*). Trong đó, một mẫu dữ liệu là một tập hợp các *đặc trưng* (*feature*) đã được định lượng từ một số đối tượng hoặc sự kiện mà ta muốn hệ thống ML xử lý. Ta thường biểu diễn một mẫu dữ liệu bằng một vector $\boldsymbol{x} \in \mathbb{R}^n$

trong đó mỗi phần tử x_i của vector tương ứng với một feature khác nhau. Ví dụ: các feature của hình ảnh thường là giá trị của các pixel trong ảnh.

Có nhiều loại tác vụ có thể được giải quyết bằng ML. Trong đó, các tác vụ phổ biến nhất bao gồm:

- **Phân loại**: Trong loại tác vụ này, chương trình máy tính được yêu cầu xác định các mẫu dữ liệu đầu vào thuộc nhóm nào trong số k nhóm cho trước. Để giải quyết nhiệm vụ này, thuật toán học thường cần tạo ra một hàm $f : \mathbb{R}^n \to \{1, ..., k\}$. Khi $y = f(\boldsymbol{x})$, mô hình sẽ gán một đầu vào được mô tả bởi vector $\boldsymbol{x}$ vào một nhóm được xác định bởi mã số y. Ngoài ra, tác vụ phân loại còn có các biến thể khác, chẳng hạn khi f cho ra phân phối xác suất trên các lớp. Một ví dụ khác về phân loại là nhận dạng đối tượng, trong đó đầu vào là một hình ảnh (thường được mô tả như một tập hợp các giá trị độ sáng pixel), và đầu ra là một mã số xác định đối tượng có trong hình ảnh. Chẳng hạn, robot Willow Garage PR2 có khả năng hoạt động như một người bồi bàn có thể nhận ra các loại đồ uống khác nhau và mang ra cho khách [Good-fellow et al., 2010]. Mô hình nhận dạng đối tượng tốt nhất hiện nay được dựa trên DL [Krizhevsky et al., 2012; Ioffe và Szegedy, 2015]. Nhận dạng đối tượng là công nghệ cơ bản cho phép máy tính nhận diện khuôn mặt [Taigman et al., 2014], có thể được sử dụng để tự động gắn thẻ mọi người trong bộ sưu tập ảnh và giúp máy tính tương tác tự nhiên hơn với người dùng của họ.

- **Phân loại các dữ liệu bị thiếu**: Phân loại sẽ trở nên khó khăn hơn nếu hệ thống không được bảo đảm rằng mọi phép đo trong vector đầu vào luôn được cung cấp. Để giải quyết tác vụ phân loại, thuật toán học chỉ cần xác định một hàm *duy nhất* ánh xạ một vector đầu vào tới một nhóm ở đầu ra. Tuy nhiên, khi một số đầu vào có thể bị thiếu, thay vì xác định một hàm phân loại duy nhất, thuật toán học phải học một *tập* các hàm, trong đó mỗi hàm tương ứng với việc phân loại $\boldsymbol{x}$ với mỗi tập hợp con khác nhau của các đầu vào bị thiếu. Tình huống này phát sinh thường xuyên trong chẩn đoán y khoa, bởi vì có nhiều loại xét nghiệm y tế đắt đỏ. Một cách để định nghĩa một tập các các hàm lớn như vậy là học một phân phối xác suất trên tất cả các biến có liên quan, sau đó giải quyết tác vụ phân loại bằng cách lấy xác suất biên trên các biến bị thiếu. Với n biến đầu vào, ta có thể có tất cả 2^n hàm phân lớp cần thiết khác nhau cho mỗi tập hợp các đầu vào bị thiếu,

nhưng chương trình máy tính chỉ cần học một hàm duy nhất mô tả phân phối xác suất chung. Công trình của Goodfellow và các cộng sự (2013b) là một ví dụ về một mô hình xác suất đa tầng được áp dụng cho một tác vụ như vậy theo cách này. Nhiều tác vụ khác được mô tả trong phần này cũng có thể được khái quát hóa để xử lý các đầu vào bị thiếu; phân loại với đầu vào bị thiếu chỉ là một ví dụ về những gì ML có thể làm.

- **Hồi quy**: Trong loại tác vụ này, chương trình máy tính được yêu cầu dự đoán một giá trị số khi cho trước một vài điểm đầu vào. Để giải quyết tác vụ này, thuật toán học được yêu cầu xuất ra một hàm $f : \mathbb{R}^n \to \mathbb{R}$. Kiểu tác vụ này tương tự với tác vụ phân loại, chỉ khác nhau ở định dạng đầu ra. Một ví dụ về tác vụ hồi quy là dự đoán số tiền bồi thường dự kiến mà một người mua bảo hiểm sẽ được trả (được sử dụng để thiết lập phí bảo hiểm), hoặc dự đoán giá chứng khoán trong tương lai. Những loại dự đoán này cũng được sử dụng cho các giao dịch dựa trên thuật toán.

- **Phiên mã**: Trong loại tác vụ này, hệ thống ML được yêu cầu quan sát một biểu diễn tương đối không có cấu trúc của một số loại dữ liệu và ghi lại thông tin đó thành dạng văn bản rời rạc. Ví dụ, trong nhận dạng ký tự quang học, chương trình máy tính được xem một ảnh có chứa hình ảnh văn bản và được yêu cầu trả lại văn bản này dưới dạng chuỗi ký tự (chẳng hạn ở định dạng ASCII hoặc Unicode). *Google Street View* sử dụng DL để xử lý các số địa chỉ theo cách này [Goodfellow et al., 2014d]. Một ví dụ khác là nhận dạng giọng nói, trong đó chương trình máy tính được cung cấp một dạng sóng âm thanh và xuất ra chuỗi ký tự hoặc mã ID của từ mô tả các từ được nói trong bản ghi âm. DL là một thành phần quan trọng của các hệ thống nhận dạng giọng nói hiện đại được sử dụng ở rất nhiều các công ty lớn, bao gồm Microsoft, IBM và Google [Hinton et al., 2012b].

- **Dịch máy**: Trong một tác vụ dịch máy, đầu vào đã bao gồm một chuỗi các ký tự trong một số ngôn ngữ, và chương trình máy tính phải chuyển đổi chúng thành chuỗi ký tự trong một ngôn ngữ khác. Tác vụ này thường được áp dụng cho các ngôn ngữ tự nhiên, chẳng hạn như dịch từ tiếng Anh sang tiếng Pháp. DL gần đây đã bắt đầu có một ảnh hưởng quan trọng đối với dịch máy [Sutskever et al., 2014; Bahdanau et al., 2015].

- **Đầu ra có cấu trúc**: Các tác vụ loại này bao hàm bất kỳ tác vụ nào mà đầu ra là một vector (hoặc một cấu trúc dữ liệu khác chứa nhiều giá trị) với các mối quan hệ quan trọng giữa các phần tử với nhau. Đây là một nhóm tác vụ rộng, áp dụng các tác vụ phiên mã và phiên dịch được mô tả phía trên, cũng như nhiều tác vụ khác. Một ví dụ là phân tích cú pháp — giúp ánh xạ một câu ngôn ngữ tự nhiên tới một cây mô tả cấu trúc ngữ pháp của nó bằng cách gán nhãn các nút của cây thành các động từ, danh từ, trạng từ, v.v. Công trình của Collobert (2011) cho ta một ví dụ về việc áp dụng DL cho tác vụ phân tích cú pháp. Một ví dụ khác là phân vùng hình ảnh theo pixel, trong đó chương trình máy tính gán mỗi pixel trong một hình ảnh cho một nhóm cụ thể. Ví dụ, DL cũng có thể được sử dụng để chú thích vị trí của các con đường trong những bức ảnh chụp từ trên không [Mnih and Hinton, 2010]. Dạng của đầu ra không cần phản chiếu cấu trúc của đầu vào giống như trong các tác vụ kiểu chú thích này. Chẳng hạn, trong tác vụ chú thích hình ảnh, chương trình máy tính quan sát một hình ảnh và xuất ra một câu ngôn ngữ tự nhiên mô tả hình ảnh [Kiroset al., 2014a,b; Mao et al., 2015; Vinyals et al., 2015b; Donahue et al., 2014;Karpathy and Li, 2015; Fang et al., 2015; Xu et al., 2015]. Các tác vụ này được gọi là các *tác vụ đầu ra có cấu trúc* bởi ở đó chương trình phải xuất ra nhiều giá trị được liên kết chặt chẽ với nhau. Ví dụ, các từ được tạo bởi chương trình sinh phụ đề cho hình ảnh phải tạo thành một câu hợp lệ.

- **Phát hiện bất thường**: Trong loại tác vụ này, chương trình máy tính cần phải rà soát một tập các sự kiện hoặc đối tượng và đánh dấu một số trong số đó là bình thường hoặc bất thường. Một ví dụ về một tác vụ phát hiện bất thường là phát hiện gian lận thẻ tín dụng. Bằng cách mô hình hóa những thói quen mua hàng của bạn, một công ty phát hành thẻ tín dụng có thể phát hiện việc sử dụng thẻ sai mục đích. Nếu có kẻ trộm đánh cắp thông tin thẻ tín dụng hoặc thẻ tín dụng của bạn, việc mua hàng của tên trộm thường sẽ đến từ một phân phối xác suất khác hẳn phân phối xác suất các mặt hàng mà thông thường bạn mua. Nhờ đó công ty thẻ tín dụng có thể ngăn chặn gian lận bằng cách tạm khóa tài khoản ngay sau khi thẻ đó đã được sử dụng cho một giao dịch mua hàng được xem là bất thường. Xem khảo sát các phương pháp phát hiện bất thường trong bài báo của Chandola và các cộng sự (2009).

- **Tổng hợp và lấy mẫu**: Trong loại tác vụ này, thuật toán ML được yêu cầu tạo ra các mẫu dữ liệu mới tương tự như các mẫu dữ liệu trong dữ liệu huấn luyện. Tổng hợp và lấy mẫu thông qua ML có thể hữu ích cho các ứng dụng truyền thông bởi việc tạo ra lượng lớn nội dung bằng tay sẽ tốn kém, nhàm chán hoặc yêu cầu quá nhiều thời gian. Ví dụ, các trò chơi video có thể tự động tạo họa tiết cho các đối tượng lớn hoặc các phong cảnh, thay vì yêu cầu một hoạ sĩ ghi nhãn theo cách thủ công từng pixel [Luo et al., 2013]. Trong một số trường hợp, ta cần quy trình lấy mẫu hoặc tổng hợp để sinh ra một kiểu đầu ra cụ thể cho đầu vào. Chẳng hạn, trong một tác vụ tổng hợp giọng nói, ta đưa vào một câu dạng văn bản và yêu cầu chương trình phát ra một dạng sóng âm thanh có chứa một phiên bản tiếng nói của câu đó. Đây là một loại tác vụ đầu ra có cấu trúc, nhưng cần thêm điều kiện rằng không có đầu ra chính xác duy nhất cho mỗi đầu vào, và rõ ràng rằng ta sẽ mong muốn một lượng lớn biến thể ở đầu ra, để đầu ra có vẻ tự nhiên và giống thật hơn.

- **Gán giá trị khuyết**: Trong loại tác vụ này, thuật toán ML được đưa vào một mẫu $\boldsymbol{x} \in \mathbb{R}^n$ mới, nhưng bị thiếu đi một số đầu vào x_i của $\boldsymbol{x}$. Và thuật toán phải đưa ra dự đoán về các giá trị của các đầu vào bị khuyết.

- **Khử nhiễu**: Trong những tác vụ kiểu này, thuật toán ML được cho một mẫu dữ liệu bị nhiễu $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ sinh ra từ một tiến trình gặp sai sót với mẫu dữ liệu sạch $\boldsymbol{x} \in \mathbb{R}^n$. Thuật toán học cần phải dự đoán mẫu sạch $\boldsymbol{x}$ từ phiên bản bị nhiễu của nó $\tilde{\boldsymbol{x}}$, hoặc tổng quát hơn, là dự đoán phân phối xác suất có điều kiện $p(\boldsymbol{x}|\tilde{\boldsymbol{x}})$.

- **Ước lượng mật độ** hay **ước lượng hàm khối của xác suất**: Trong bài toán ước lượng mật độ, thuật toán ML được yêu cầu xác định một hàm $p_{\text{model}} : \mathbb{R}^n \rightarrow R$, trong đó $p_{\text{model}}(\boldsymbol{x})$ có thể xem là hàm mật độ xác suất (nếu $\boldsymbol{x}$ là liên tục) hoặc một hàm khối xác suất (nếu $\boldsymbol{x}$ là rời rạc) từ không gian lấy mẫu. Để giải quyết tốt bài toán này (ta sẽ chỉ ra cách để đánh giá độ tốt này trong phần thảo luận về chỉ tiêu đánh giá P), thuật toán cần học được cấu trúc của dữ liệu mà nó đã quan sát. Nó phải biết ở đâu thì các điểm dữ liệu co cụm lại sát cạnh nhau và ở đâu thì dữ liệu ít có khả năng xuất hiện. Phần lớn các tác vụ được mô tả ở trên đều yêu cầu thuật toán ML ít nhất cũng phải nắm bắt được, theo nghĩa ngầm định, cấu trúc phân phối xác suất của dữ liệu. Ước lượng mật độ cho phép ta nắm bắt phân phối đó một cách rõ

ràng. Về nguyên tắc, từ đó ta cũng có thể thực hiện tính toán trên phân phối này để giải quyết các tác vụ khác một cách tốt hơn. Chẳng hạn, nếu ta đã thực hiện ước lượng mật độ và thu được một phân phối xác suất $p(\boldsymbol{x})$, ta có thể sử dụng phân phối đó để giải quyết bài toán xác định giá trị khuyết thiếu. Nếu một giá trị x_i bị khuyết, và ta đã có tất cả các giá trị khác, ký hiệu là $\boldsymbol{x}_{-i}$, thì ta biết rằng nó tuân theo phân phối $p(x_i|\boldsymbol{x}_{-i})$. Trong thực tế, ước lượng mật độ không phải lúc nào cũng cho phép ta giải quyết tất cả các tác vụ liên quan trên, bởi trong nhiều trường hợp, các phép toán trên phân phối $p(\boldsymbol{x})$ có thể quá nặng về mặt tính toán.

Dĩ nhiên, có thể có nhiều tác vụ và loại tác vụ khác nữa. Các loại tác vụ mà chúng tôi liệt kê ở đây chỉ nhằm cung cấp cho bạn đọc một số ví dụ về những tác vụ mà ML có thể giải quyết, chứ không nhằm mục đích định nghĩa một cách phân loại cứng nhắc cho các tác vụ này.

5.1.2 Đánh giá hiệu năng P

Để đánh giá khả năng của một thuật toán ML, ta cần xây dựng một phương thức đo đạc một cách định lượng hiệu năng của nó. Thông thường, phương pháp đo đạc hiệu năng P này là chuyên biệt cho một loại tác vụ T được thực hiện bởi hệ thống.

Đối với các tác vụ như phân loại, phân loại với đầu vào khiếm khuyết, và phiên mã, ta thường đo đạc *độ chính xác* (*accuracy*) của mô hình. Độ chính xác chỉ là tỷ lệ các mẫu dữ liệu mà mô hình dự đoán chính xác đầu ra của nó. Ta cũng có một độ đo tương đương gọi là *tỷ lệ lỗi* (*error rate*). Tỷ lệ lỗi là tỷ lệ các mẫu dữ liệu bị mô hình gán nhãn sai. Ta thường nhắc đến tỷ lệ lỗi như là tỷ lệ mất mát 0-1 kì vọng. Tỷ lệ mất mát 0-1 (0-1 loss) được tính bằng 0 nếu như mẫu được phân loại chính xác và 1 trong trường hợp ngược lại. Đối với những tác vụ như ước lượng mật độ, thì các phép đo về độ chính xác, tỷ lệ lỗi hoặc các loại mất mát 0-1 thường không có nhiều ý nghĩa. Thay vào đó, ta cần sử dụng một thông số hiệu năng khác cho phép mô hình trả về một giá trị liên tục cho mỗi mẫu. Phương pháp tiếp cận phổ biến nhất là sử dụng trung bình logarit của xác suất mà mô hình gán cho một số mẫu dữ liệu.

Thông thường, ta sẽ quan tâm liệu thuật toán ML có chạy tốt trên những dữ liệu mà nó chưa gặp trước đó hay không? Bởi nó sẽ quyết định liệu mô hình có chạy

tốt nếu đem ra triển khai trên thực tế. Vì vậy nên ta cần đánh giá những thông số hiệu năng trên một *tập kiểm thử* (*test set*). Tập kiểm thử này phải hoàn toàn tách biệt với tập dữ liệu đã được sử dụng để huấn luyện hệ thống ML.

Việc lựa chọn phương pháp đánh giá hiệu năng có vẻ dễ dàng và khách quan, nhưng thực tế là không dễ để chọn một phương pháp đánh giá hiệu năng mà thực sự phản ánh hành vi mong muốn của hệ thống.

Trong một số trường hợp, khá khó khăn để xác định việc ta cần phải đo cái gì. Chẳng hạn, khi thực hiện một tác vụ phiên mã, liệu ta nên đo độ chính xác của hệ thống ở mức phiên mã cả chuỗi hoàn chỉnh, hay là một phương pháp đánh giá chi tiết hơn, có khả năng đưa ra điểm số cho từng phần dựa trên các phần tử của chuỗi được phiên mã chính xác? Khi thực hiện một tác vụ hồi quy, liệu ta nên phạt hệ thống nhiều hơn khi nó thường xuyên mắc những lỗi trung bình hay khi nó thỉnh thoảng mắc những lỗi rất lớn? Những lựa chọn này thường tùy thuộc vào ứng dụng mà bạn muốn triển khai.

Trong những trường hợp khác, giả sử ta biết rõ thứ cần đo, nhưng việc đo chúng, về cơ bản là không thực tế. Chẳng hạn, điều này thường xuất hiện trong việc ước lượng mật độ. Nhiều mô hình xác suất tốt nhất hiện nay chỉ có thể biểu diễn phân phối xác suất một cách ngầm định. Tính toán giá trị xác suất thực tế gắn với một điểm cụ thể trên không gian mẫu ở những mô hình này là không thể thực hiện được về mặt tính toán. Trong những trường hợp như vậy, người ta thường thiết kế ra một tiêu chí thay thế mà vẫn phản ánh mục tiêu ban đầu hoặc xây dựng một xấp xỉ đủ tốt cho tiêu chí mong muốn.

5.1.3 Kinh nghiệm E

Các thuật toán ML có thể được phân loại thành hai nhóm lớn là học *có giám sát* hoặc học *không giám sát*, tùy thuộc vào loại kinh nghiệm mà chúng sử dụng trong suốt quá trình học.

Hầu hết các thuật toán ML trong cuốn sách này có thể được hiểu là được phép áp dụng toàn bộ tập dữ liệu. Một tập dữ liệu là một tập bao gồm rất nhiều mẫu dữ liệu, như được định nghĩa trong mục 5.1.1. Đôi khi ta cũng gọi các mẫu này là những *điểm dữ liệu* (*data point*).

Một trong những tập dữ liệu cổ điển nhất được các nhà thông kê và các nhà nghiên cứu ML sử dụng là tập dữ liệu về hoa Diên Vĩ (Iris) [Fisher, 1936]. Đây là

tập dữ liệu đo đạc những phần khác nhau của 150 cá thể cây thuộc các loại Diên Vĩ khác nhau. Mỗi cá thể có một mẫu tương ứng. Các đặc trưng trong mỗi mẫu dữ liệu là kích thước của từng bộ phận của cây bao gồm: độ dài đài hoa, độ rộng của đài hoa cũng như độ dài và độ rộng của cánh hoa. Tập dữ liệu này cũng đồng thời ghi lại những cá thể này thuộc loài nào trong họ Diên Vĩ. Và trong tập dữ liệu này có 3 loại Diên Vĩ hiện diện.

Các thuật toán học không giám sát áp dụng trên một tập dữ liệu bao gồm nhiều đặc trưng, và chúng học những đặc tính hữu dụng ẩn sau cấu trúc của tập dữ liệu này. Ở khía cạnh DL, ta thường mong muốn làm sao đó có thể học được toàn bộ phân phối xác suất đã tạo ra tập dữ liệu đó, giống như trong ước lượng mật độ, hoặc ngầm định như trong các tác vụ tổng hợp hoặc khử nhiễu. Ngoài ra, một số thuật toán học không giám sát khác lại thực hiện một số tác vụ khác như phân cụm, giúp việc phân chia tập dữ liệu thành các cụm chứa các mẫu có sự tương đồng lớn với nhau.

Các thuật toán học có giám sát áp dụng trên một tập dữ liệu chứa nhiều đặc trưng, và mỗi mẫu dữ liệu được gắn với một *nhãn* (*label*) hay *đích* (target) nào đó. Ví dụ, tập dữ liệu hoa Diên Vĩ được chú thích bởi các loài trong họ Diên Vĩ mà mỗi mẫu thuộc về. Một thuật toán học có giám sát có thể học từ tập dữ liệu hoa Diên vĩ và tìm cách phân loại các cá thể này vào 3 loại hoa Diên Vĩ khác nhau, dựa trên các đo đạc của chúng.

Nói một cách nôm na, học không giám sát gắn với việc quan sát các mẫu khác nhau của một vector ngẫu nhiên $\mathbf{x}$, và tìm cách trực tiếp hoặc gián tiếp học được phân phối xác suất $p(\mathbf{x})$, hoặc một số đặc tính hữu ích của phân phối đó. Trong khi đó, học có giám sát gắn với việc quan sát một số mẫu của vector ngẫu nhiên $\mathbf{x}$ cùng với một giá trị hay một vector $\mathbf{y}$ gắn với nó, và tìm cách dự đoán giá trị của $\mathbf{y}$ từ $\mathbf{x}$, thường thông qua ước lượng xác suất $p(\mathbf{y}|\mathbf{x})$. Thuật ngữ **học có giám sát** bắt nguồn từ góc nhìn là giá trị đích $\mathbf{y}$ được đưa ra bởi một giáo viên hay một hướng dẫn viên, để chỉ cho hệ thống ML cần phải làm gì. Ngược lại, trong học không có giám sát, không tồn tại hướng dẫn viên hay giáo viên nào, và thuật toán phải tự học để hiểu rõ dữ liệu mà không cần chỉ dẫn đó.

Học có giám sát và học không giám sát không phải là các thuật ngữ được định nghĩa một cách chuẩn tắc. Ranh giới giữa chúng nhiều khi bị xóa mờ. Nhiều công cụ trong ML có thể được dùng để giải quyết cả hai tác vụ này. Ví dụ, quy

tắc chuỗi trong xác suất chỉ ra rằng, cho một vector $\mathbf{x} \in \mathbb{R}^n$, đồng phân phối xác suất có thể được phân rã thành:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) \qquad (5.1)$$

Sự phân rã này có nghĩa là ta có thể giải những bài toán có bề ngoài là bài toán học không giám sát để mô phỏng $p(\mathbf{x})$, bằng cách chia nhỏ nó thành n bài toán học có giám sát. Mặt khác, ta có thể giải bài toán học có giám sát nhằm tính xác suất $p(y|\mathbf{x})$ bằng cách sử dụng các kỹ thuật trong học không giám sát, để tìm ra đồng phân phối $p(\mathbf{x}, y)$, và từ đó suy ra

$$p(y|\mathrm{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')} \qquad (5.2)$$

Mặc dù học có/không giám sát không hoàn toàn là những khái niệm chuẩn và phân biệt rõ ràng, chúng cũng phần nào giúp phân loại phần lớn những vấn đề mà ta gặp phải. Thông thường, người ta coi các bài toán hồi quy, phân lớp hay những bài toán có đầu ra có cấu trúc, là học có giám sát. Việc ước lượng mật độ để hỗ trợ các tác vụ khác, thường được xem là học không có giám sát.

Ngoài ra, cũng tồn tại một số biến thể khác từ hai loại mô hình trên. Ví dụ, trong *học bán giám sát* (*semi-supervised learning*), một số mẫu dữ liệu đã được gán nhãn, trong khi một số khác lại không. Trong *học đa mẫu* (*multi-instance learning*), toàn bộ một tập mẫu dữ liệu đã được gán nhãn là chứa hoặc không chứa một mẫu của một lớp nào đó, nhưng từng mẫu riêng biệt trong tập đó lại không được gán nhãn là chúng thuộc lớp nào. Một ví dụ gần đây về học đa mẫu với những mô hình DL [Kotzias et al, 2015].

Một số thuật toán ML không chỉ áp dụng trên một tập dữ liệu cố định. Ví dụ, các thuật toán *học tăng cường* (*reinforcement learning*) tự thân nó tương tác với một môi trường, và duy trì một vòng lặp phản hồi giữa hệ thống học và những kinh nghiệm của nó với môi trường. Những thuật toán như vậy vượt quá phạm vi của quyển sách này, nếu bạn đọc quan tâm có thể tham khảo Sutton and Barto (1998) hoặc Bertsekas and Tsitsiklis (1996) để biết thêm thông tin về học tăng cường. Ngoài ra, công trình của Mnih và các cộng sự (2013) cung cấp một số phương pháp tiếp cận học tăng cường từ góc nhìn của DL.

Hầu hết các thuật toán ML thường chỉ áp dụng trên một tập dữ liệu. Một tập dữ liệu có thể được mô tả theo nhiều cách khác nhau. Trong mọi trường hợp, một

tập dữ liệu là một tập các mẫu dữ liệu, mà bản thân mỗi mẫu này là một tập của các đặc trưng.

Một cách thông dụng để mô tả một tập dữ liệu là sử dụng *ma trận thiết kế* (*design matrix*). Ma trận thiết kế là một ma trận chứa một mẫu trên mỗi dòng. Mỗi cột của ma trận tương ứng với mỗi đặc trưng khác nhau. Ví dụ, tập dữ liệu hoa Diên vĩ bao gồm 150 mẫu, với 4 đặc trưng cho mỗi mẫu. Điều này đồng nghĩa với việc ta có thể biểu diễn tập dữ liệu này dưới dạng một ma trận thiết kế $\boldsymbol{X} \in \mathbb{R}^{150 \times 4}$, trong đó $X_{i,1}$ là chiều dài của đài hoa của cây thứ i, $X_{i,2}$ là chiều rộng của đài hoa của cây thứ i, v.v. Ta sẽ mô tả hầu hết các thuật toán ML trong cuốn sách này ở khía cạnh chúng hoạt động như thế nào trên tập dữ liệu dưới dạng ma trận thiết kế.

Dĩ nhiên, để mô tả một tập dữ liệu dưới dạng ma trận thì mỗi mẫu dữ liệu phải được biểu diễn dưới dạng các vector, và các vector này phải có cùng kích thước với nhau. Điều này không phải lúc nào cũng khả thi. Thí dụ, nếu bạn có một tập các bức ảnh với kích thước chiều dài, chiều rộng khác nhau, thì bản thân mỗi ảnh sẽ chứa những số pixel khác nhau, và do đó, những ảnh này không thể được mô tả dưới dạng các vector cùng độ dài. Mục 9.7 và chương 10 sẽ mô tả cách để ta có thể xử lý những kiểu dữ liệu không đồng nhất như vậy. Trong những trường hợp như vậy, thay vì mô tả tập dữ liệu thành một ma trận với m dòng, ta sẽ mô tả nó như là một tập chứa m phần tử: $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(m)}$. Ký hiệu này không bắt buộc hai vector của hai mẫu bất kỳ $\boldsymbol{x}^{(i)}$ và $\boldsymbol{x}^{(j)}$ có cùng kích thước với nhau.

Trong học có giám sát, mẫu dữ liệu chứa một nhãn đi kèm với một tập các đặc trưng. Chẳng hạn, nếu ta muốn sử dụng một thuật toán ML để nhận diện vật thể trong các bức ảnh, ta cần chỉ ra những đồ vật nào xuất hiện trong từng bức ảnh. Ta có thể làm điều này với một mã số, ví dụ, số 0 để đánh dấu một người, số 1 để đánh dấu một chiếc ô tô, số 2 để đánh dấu một con mèo, và cứ như vậy. Thông thường, khi làm việc với một tập dữ liệu chứa một ma trận các đặc trưng của các quan sát $\boldsymbol{X}$, ta cũng gán kèm nó với một vector nhãn $\boldsymbol{y}$, với nhãn y_i là nhãn của mẫu thứ i.

Dĩ nhiên, đôi khi nhãn dữ liệu này có thể không chỉ đơn thuần là các con số. Ví dụ như khi ta muốn huấn luyện một hệ thống nhận diện giọng nói để dịch toàn bộ các câu, thì nhãn của mỗi câu là một chuỗi các từ liên kết với nhau.

Cũng như việc không có một định nghĩa chính thức nào về học có và không có giám sát, các tập dữ liệu hay các kinh nghiệm cũng không có một cách phân loại cứng nhắc nào. Những cấu trúc được mô tả ở đây cũng đã bao gồm hầu hết các trường hợp, nhưng bạn đọc vẫn hoàn toàn có thể thiết kế một kiểu cấu trúc dữ liệu mới để phù hợp với các ứng dụng mới.

5.1.4 Ví dụ: hồi quy tuyến tính

ta đã định nghĩa một thuật toán ML là một thuật toán có thể cải thiện hiệu năng của một chương trình máy tính trong một số tác vụ thông qua những kinh nghiệm thu được, nhưng đây vẫn là một định nghĩa khá mơ hồ và trừu tượng. Để mô tả một cách cụ thể hơn, ta sẽ trình bày một ví dụ về một thuật toán ML rất cơ bản: *hồi quy tuyến tính*. Ta sẽ nhắc lại ví dụ này nhiều lần để giới thiệu thêm về những khái niệm của ML giúp bạn đọc có thể hiểu hơn về hành vi của thuật toán này.

Giống như cái tên đã mô tả, hồi quy tuyến tính được sử dụng để giải các một bài toán hồi quy. Nói cách khác, mục đích của nó là xây dựng một hệ thống có thể nhận vào một vector đầu vào $\boldsymbol{x} \in \mathbb{R}^n$ và dự đoán giá trị của một số vô hướng $y \in \mathbb{R}$ ở đầu ra. Gọi $\hat{y}$ là giá trị mà mô hình của ta dự đoán y sẽ nhận. Ta định nghĩa đầu ra của hệ thống theo công thức:

$$\hat{y} = \boldsymbol{w}^\intercal \boldsymbol{x} \qquad (5.3)$$

trong đó $\boldsymbol{w} \in \mathbb{R}^n$ là một vector của các *tham số* (parameter).

Các tham số là các giá trị điều khiển hành vi của hệ thống. Trong trường hợp này, w_i là hệ số mà ta đem nhân với đặc trưng x_i trước khi cộng với đóng góp từ tất cả các đặc trưng. Ta có thể hiểu $\boldsymbol{w}$ như một tập các *trọng số* (weight) quyết định mức độ ảnh hưởng của đặc trưng đối với kết quả dự đoán. Nếu đặc trưng x_i nhận một giá trị trọng số w_i dương, đồng nghĩa với việc, tăng giá trị của đặc trưng đó sẽ làm tăng giá trị của kết quả dự đoán $\hat{y}$. Và ngược lại, nếu một đặc trưng có trọng số âm, thì việc tăng giá trị của đặc trưng này sẽ làm giảm giá trị ở kết quả dự đoán. Trọng số của một đặc trưng có giá trị lớn nghĩa là đặc trưng đó có mức độ ảnh hưởng lớn đến kết quả dự đoán. Và nếu giá trị trọng số này là 0, đặc trưng này không hề ảnh hưởng đến kết quả dự đoán.

Do đó, ta xây dựng được định nghĩa về tác vụ T như sau: dự đoán giá trị của y từ $\boldsymbol{x}$ bằng cách tính giá trị đầu ra $\hat{y} = \boldsymbol{w}^\intercal \boldsymbol{x}$. Tiếp theo, ta cần một định nghĩa về

phương pháp đánh giá hiệu năng, P.

Giả thiết rằng ta có một ma trận đầu vào với m mẫu không dùng để huấn luyện, mà chỉ dùng để đánh giá độ chính xác của mô hình dự đoán. Ta cũng có một vector của các đích hồi quy mang giá trị chính xác của y ứng với mỗi mẫu dữ liệu đầu vào. Bởi tập dữ liệu này chỉ được dùng cho việc đánh giá hiệu năng, nên ta gọi nó là tập kiểm thử. Ta ký hiệu ma trận đầu vào này là $\boldsymbol{X}^{(\text{test})}$ và vector giá trị các đích hồi quy là $\boldsymbol{y}^{(\text{test})}$.

Một cách để đo hiệu năng của mô hình này là tính giá trị *sai số bình phương trung bình* (mean squared error) khi áp dụng mô hình này lên tập kiểm thử. Gọi $\hat{\boldsymbol{y}}^{(\text{test})}$ là giá trị dự đoán của mô hình trên tập kiểm thử, thì giá trị sai số bình phương trung bình được tính theo công thức:

$$\text{MSE}_{\text{test}} = \frac{1}{m}\sum_i (\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})})_i^2 \qquad (5.4)$$

Về mặt trực quan, có thể thấy giá trị của sai số này giảm về 0 khi $\hat{\boldsymbol{y}}^{(test)} = \boldsymbol{y}^{(test)}$. Ngoài ra, ta cũng có thể thấy rằng:

$$\text{MSE}_{\text{test}} = \frac{1}{m}||\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}||_2^2, \qquad (5.5)$$

nên giá trị sai số tăng lên khi khoảng cách Euclide giữa giá trị dự đoán và giá trị thực tế của nhãn tăng lên.

Để xây dựng một thuật toán ML, ta cần xây dựng một thuật toán có thể cập nhật giá trị của các trọng số $\boldsymbol{w}$ sao cho sai số MSE_{test} giảm dần sau khi thuật toán thu được kinh nghiệm khi quan sát tập huấn luyện ($\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}$). Một cách rất trực quan để thực hiện điều này (chúng tôi sẽ chứng minh trong phần 5.5.1) là tối thiểu hóa giá trị sai số bình phương trung bình của mô hình trên tập huấn luyện, MSE_{train}.

Để cực tiểu hoá $\text{MSE}_{\text{train}}$, ta chỉ cần tìm giá trị của $\boldsymbol{w}$ sao cho đạo hàm của MSE bằng $\mathbf{0}$:

$$\nabla_{\boldsymbol{w}}\text{MSE}_{\text{train}} = 0 \qquad (5.6)$$

$$\Rightarrow \nabla_{\boldsymbol{w}}\frac{1}{m}||\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0 \qquad (5.7)$$

$$\Rightarrow \frac{1}{m}\nabla_{\boldsymbol{w}}||\boldsymbol{X}^{(\text{train})}w - \boldsymbol{y}^{(\text{train})}||_2^2 = 0 \qquad (5.8)$$

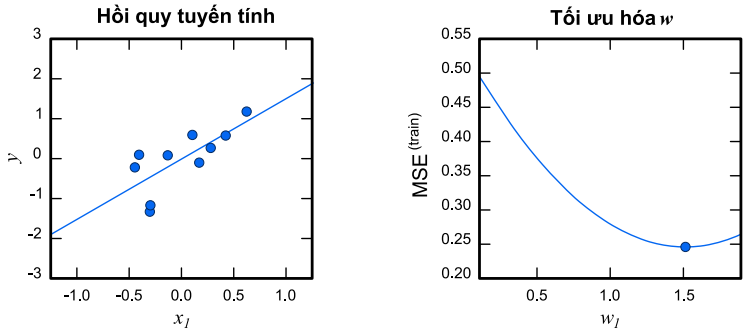
$$\Rightarrow \nabla_{\boldsymbol{w}}\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)^\intercal\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right) = 0 \qquad (5.9)$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{w}^\intercal \boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{X}^{(\text{train})} \boldsymbol{w} - 2\boldsymbol{w}^\intercal \boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{y}^{(\text{ train})} + \boldsymbol{y}^{(\text{train})^\intercal} \boldsymbol{y}^{(\text{train})} \right) = 0 \tag{5.10}$$

$$\Rightarrow 2\boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{X}^{(\text{train})} \boldsymbol{w} - 2\boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{y}^{(\text{train})} = 0 \tag{5.11}$$

$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{X}^{(\text{train})} \right)^{-1} \boldsymbol{X}^{(\text{train})^\intercal} \boldsymbol{y}^{(\text{train})} \tag{5.12}$$

Những phương trình có nghiệm cho bởi công thức 5.12 được gọi là các phương trình chuẩn (normal equation). Từ phương trình (5.12), ta có thể tạo ra một thuật toán ML đơn giản. Xem ví dụ về hồi quy tuyến tính trong hình 5.1.



> Hình 5.1: Trong một bài toán hồi quy tuyến tính, với tập huấn luyện chứa 10 điểm dữ liệu, mỗi điểm chứa 1 đặc trưng. Bởi chỉ có duy nhất một đặc trưng, vector trọng số $\boldsymbol{w}$ chứa chỉ duy nhất một thông số cần học là w_1. *(Hình bên trái)* Ta thấy, thuật toán hồi quy tuyến tính cố gắng học w_1 sao cho đường thẳng $y = w_1 x$ đi qua càng gần tất cả các điểm trong tập huấn luyện càng tốt. *(Hình bên phải)* Điểm in đậm màu xanh trên hình bên phải là giá trị w_1 tìm được từ nghiệm của phương trình chuẩn, và ta cũng có thể thấy là tại đó, giá trị lỗi **MSE** trên tập huấn luyện cũng là nhỏ nhất.

Cần lưu ý, thuật ngữ *hồi quy tuyến tính* thường được dùng để ám chỉ một mô hình phức tạp hơn một chút, với một tham số thêm vào – hệ số chặn (intercept) b . Trong mô hình này

$$\hat{y} = \boldsymbol{w}^\intercal \boldsymbol{x} + b \tag{5.13}$$

do đó ánh xạ từ các tham số tới các kết quả dự đoán vẫn là hàm tuyến tính, nhưng ánh xạ từ các vector đặc trưng tới các kết quả dự đoán lại là một hàm tịnh tiến. Có nghĩa là đồ thị biểu diễn các dự đoán của mô hình vẫn trông giống như một đường thẳng, nhưng không cần phải đi qua gốc toạ độ. Thay vì việc phải thêm số hạng tự do b vào, người ta có thể tiếp tục sử dụng mô hình với chỉ các trọng số w bằng cách nối thêm vào vector đặc trưng $\boldsymbol{x}$ một đầu vào x_0 luôn có giá trị bằng 1. Trọng số ứng với hệ số 1 sẽ đóng vai trò của số hạng tự do b.

Chúng tôi sẽ thường xuyên sử dụng thuật ngữ “tuyến tính” để ám chỉ những hàm tịnh tiến xuyên suốt cuốn sách này.

Hệ số chặn b thường được gọi là *hệ số tự do* (bias) của phép biến đổi tịnh tiến. Khái niệm này xuất phát từ góc nhìn rằng kết quả đầu ra của phép biến đổi sẽ nghiêng về phía b khi không có bất cứ đầu vào nào. Thuật ngữ này không giống với ý nghĩa của *độ chệch* (bias) trong thống kê, ám chỉ mức sai lệch thuật toán ước lượng thống kê khi nó ước lượng một đại lượng so với giá trị thực của đại lượng đó.

Hồi quy tuyến tính về cơ bản là một thuật toán rất đơn giản và còn nhiều hạn chế, nhưng nó cung cấp cho ta một ví dụ về cách thức mà một thuật toán ML hoạt động. Ở những phần tiếp theo, chúng tôi sẽ mô tả một số nguyên lý cơ bản ẩn sau việc xây dựng các thuật toán ML cũng như làm rõ cách mà những nguyên lý này có thể được sử dụng để xây dựng những thuật toán phức tạp hơn.

5.2 Capacity, Overfitting, Underfitting

Thách thức chính trong ML là xây dựng các thuật toán có khả năng hoạt động tốt không chỉ trên tập dữ liệu đã huấn luyện, mà còn trên cả các đầu vào *mới, chưa từng gặp trước đó*. Khả năng hoạt động tốt trên các dữ liệu đầu vào chưa được quan sát được gọi là khả năng *tổng quát hoá* (generalization).

Thông thường, khi huấn luyện một mô hình ML, ta được sử dụng một tập huấn luyện; ta có thể tính một độ đo lỗi nào đó của mô hình khi dự đoán trên tập huấn luyện, được gọi là *sai số huấn luyện* (training error), và ta mong muốn giảm sai số huấn luyện này. Điều làm ML khác biệt với tối ưu là ở chỗ ta cần *sai số tổng quát hoá* (generalization error), hay *sai số kiểm thử* (test error), càng thấp càng tốt. Sai số tổng quát hoá là giá trị kỳ vọng của sai số đối với đầu vào mới. Kỳ vọng ở đây được thực hiện trên không gian tất cả các đầu vào khả dĩ, được rút ra từ phân phối của các điểm đầu vào mà ta kỳ vọng hệ thống sẽ gặp phải trong thực tế.

ta thường ước lượng sai số tổng quát hoá của một mô hình ML bằng cách đo lường hiệu suất của nó trên một *tập kiểm thử* (test set) của các điểm dữ liệu được lựa chọn riêng biệt từ tập huấn luyện.

Trong ví dụ về hồi quy tuyến tính, chúng tôi đã huấn luyện mô hình bằng cách cực tiểu hoá sai số huấn luyện,

$$\frac{1}{m^{(\text{train})}} \|\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\|_2^2 \qquad (5.14)$$

nhưng trên thực tế ta cần quan tâm đến sai số kiểm thử, $\frac{1}{m^{(\text{test})}} \|\boldsymbol{X}^{(\text{test})}\boldsymbol{w} - \boldsymbol{y}^{(\text{test})}\|_2^2$.

Nhưng làm thế nào để tác động đến hiệu suất trên tập kiểm thử trong khi ta chỉ có thể quan sát tập huấn luyện? *Lý thuyết học thống kê* (statistical learning theory) sẽ cung cấp một số câu trả lời. Nếu tập huấn luyện và tập kiểm thử được thu thập một cách tuỳ ý, những gì ta có thể làm là rất hạn chế. Sẽ tốt hơn nhiều nếu ta được phép đưa ra một số giả định về cách thu thập dữ liệu huấn luyện và kiểm thử.

Dữ liệu huấn luyện và kiểm thử được sinh bởi một phân phối xác suất trên tập dữ liệu được gọi là *quá trình sinh dữ liệu*. Chúng tôi thường đưa ra một tập hợp các giả định được gọi chung là *giả định i.i.d.*, có nghĩa là các mẫu dữ liệu trong các tập dữ liệu là *độc lập* (independent) với nhau và tập huấn luyện cùng tập kiểm thử được *phân phối giống nhau* (identically distributed), cùng được chọn ra từ một phân phối xác suất. Giả định này cho phép ta mô tả quá trình sinh dữ liệu bằng một phân phối xác suất trên một mẫu dữ liệu. Sau đó, một phân phối tương tự được sử dụng để sinh ra toàn bộ mẫu huấn luyện và kiểm thử. Chúng tôi gọi phân phối cơ sở đó là *phân phối sinh dữ liệu*, ký hiệu là p_{data}. Khung xác suất này cùng với giả định i.i.d. cho phép ta nghiên cứu về mặt toán học mối quan hệ giữa sai số huấn luyện và sai số kiểm thử.

Một mối liên quan trực tiếp giữa sai số huấn luyện và sai số kiểm thử mà ta có thể thấy ngay đó là sai số huấn luyện kỳ vọng của một mô hình được lựa chọn ngẫu nhiên bằng với sai số kiểm thử kỳ vọng của mô hình đó. Giả sử ta có một phân phối xác suất $p(\boldsymbol{x}, y)$ và ta lấy mẫu từ nó nhiều lần để tạo ra tập huấn luyện và tập kiểm thử. Đối với một giá trị $\boldsymbol{w}$ cố định nào đó, sai số huấn luyện kỳ vọng bằng với sai số kiểm thử kỳ vọng, lí do là cả hai kỳ vọng được tạo ra bởi cùng một quá trình lấy mẫu dữ liệu. Sự khác biệt duy nhất giữa hai điều kiện là cái tên ta gán cho tập dữ liệu mà ta lấy mẫu.

Tất nhiên, khi sử dụng một thuật toán ML, ta không cố định các tham số rồi sau đó mới lấy mẫu cả hai tập dữ liệu cùng lúc. Trước tiên, ta sẽ lấy mẫu tập huấn luyện, sau đó sử dụng nó để lựa chọn các tham số sao cho sai số huấn luyện

giảm đi, rồi mới lấy mẫu tập kiểm thử. Trong quá trình này, sai số kiểm thử kỳ vọng sẽ lớn hơn sai số huấn luyện kỳ vọng. Các yếu tố quyết định để đánh giá một thuật toán ML có tốt hay không là khả năng của nó khi làm cho

1. Sai số huấn luyện nhỏ.
2. Khoảng cách giữa sai số huấn luyện và sai số kiểm thử là nhỏ.

Hai yếu tố trên tương ứng với hai thách thức chính trong ML: *underfitting* và *overfitting*. Underfitting xảy ra khi mô hình không thể đạt được một sai số đủ nhỏ trên tập huấn luyện. overfitting xảy ra khi khoảng cách giữa sai số huấn luyện và sai số kiểm thử là quá lớn.

ta có thể kiểm soát liệu một mô hình có nhiều khả năng overfitting hay underfitting hơn bằng cách thay đổi *dung lượng* (capacity) của nó. Nói nôm na, capacity của mô hình là khả năng khớp của mô hình đó với nhiều loại hàm số khác nhau. Các mô hình có capacity thấp có thể gặp khó khăn để khớp tập huấn luyện. Ngược lại, mô hình với capacity cao có thể gặp phải tình trạng overfitting bằng cách ghi nhớ một cách máy móc thuộc tính của tập huấn luyện, dẫn đến mô hình không có khả năng tổng quát trên tập kiểm thử.

Một cách để kiểm soát capacity của một thuật toán học là lựa chọn *không gian giả thuyết* (hypothesis space) của nó, là tập hợp các hàm số mà thuật toán học được phép chọn làm giải pháp. Chẳng hạn, thuật toán hồi quy tuyến tính có không gian giả thuyết là tập hợp tất cả các hàm tuyến tính của đầu vào của nó. Ta có thể khái quát hoá thuật toán hồi quy tuyến tính để nó có thể bao gồm các đa thức trong không gian giả thuyết của nó, thay vì chỉ gồm những hàm tuyến tính. Việc đó làm tăng capacity của mô hình.

ta đã quen thuộc với mô hình hồi quy tuyến tính đối với đa thức bậc 1, với giá trị mô hình dự đoán

$$\hat{y} = b + wx \qquad (5.15)$$

Bằng cách bổ sung x^2 làm một thuộc tính của mô hình hồi quy tuyến tính, ta có thể học một mô hình với hàm của x là bậc hai:

$$\hat{y} = b + w_1 x + w_2 x^2 \qquad (5.16)$$

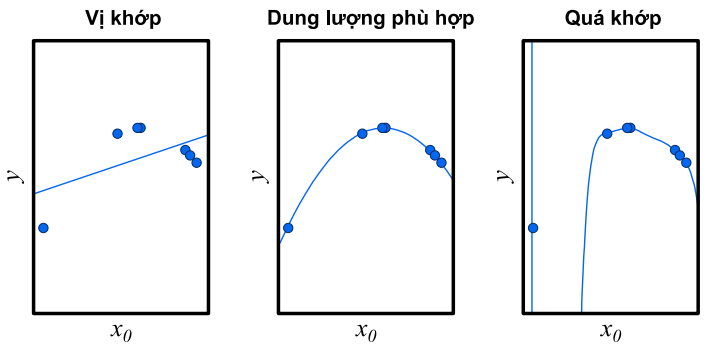
Mặc dù mô hình này sử dụng một hàm bậc hai cho đầu vào, nhưng đầu ra của nó vẫn là một hàm tuyến tính của các tham số, do đó ta vẫn có thể sử dụng các phương trình chuẩn để huấn luyện mô hình ở dạng công thức đóng. Ta có thể

tiếp tục bổ sung các luỹ thừa bậc cao hơn của x làm thuộc tính bổ sung, chẳng hạn, để thu được một đa thức bậc 9:

$$\hat{y} = b + \sum_{i=1}^{9} w_i x^i \qquad (5.17)$$

Các thuật toán ML thường sẽ hoạt động tốt nhất khi capacity của chúng phù hợp với độ phức tạp thực sự của tác vụ chúng cần thực hiện và lượng dữ liệu huấn luyện mà chúng được cung cấp. Các mô hình với capacity nhỏ sẽ không thể giải quyết các nhiệm vụ phức tạp. Ngược lại, các mô hình có capacity lớn có thể làm được việc đó, nhưng khi capacity của chúng lớn hơn những gì nhiệm vụ thực sự cần, chúng có thể bị overfitting.

Hình 5.2 biểu diễn nguyên tắc hoạt động trên. Ở đó chúng tôi so sánh các bộ dự đoán sử dụng lần lượt các hàm tuyến tính, bậc hai và bậc 9 khi cố gắng khớp với một bài toán mà ở đó các điểm dữ liệu thực tế tuân theo một hàm bậc hai. Hàm tuyến tính không thể mô tả được độ cong của đường biểu diễn dữ liệu, do đó underfitting xảy ra. Bộ dự đoán bậc 9 có khả năng mô tả dữ liệu rất chính xác, nhưng nó cũng có khả băng biểu diễn vô hạn các hàm đi qua các điểm dữ liệu huấn luyện, bởi vì nó có nhiều tham số hơn là số mẫu huấn luyện. Ta không có nhiều cơ hội để lựa chọn một lời giải có khả năng tổng quát tốt khi tồn tại quá nhiều lời giải. Trong ví dụ này, mô hình bậc hai hoàn toàn phù hợp với cấu trúc thực sự của nhiệm vụ và do đó, nó tổng quát hoá tốt đối với dữ liệu mới.



> Hình 5.2: Chúng tôi khớp ba mô hình với tập dữ liệu huấn luyện này. Dữ liệu huấn luyện được sinh bởi phép lấy mẫu x ngẫu nhiên và y được xác định bởi một hàm bậc hai. (*Hình bên trái*) Một hàm tuyến tính underfitting với dữ liệu này -- nó không thể mô tả đường cong của dữ liệu. (*Hình trung tâm*) Một hàm bậc hai khái quát hoá khá tốt với các điểm dữ liệu chưa gặp trước đó. (*Hình bên phải*) Một hàm đa thức bậc 9 lại bị overfitting khi thực hiện khớp dữ liệu. Ở đây, chúng tôi sử dụng giả nghịch đảo Moore-Penrose để

> giải các phương trình chuẩn vô định. Lời giải bậc 9 này giúp đồ thị đi qua tất cả các điểm dữ liệu huấn luyện một cách chính xác, nhưng không may là nó không giúp ta trích xuất cấu trúc thực sự của dữ liệu. Có thể thấy một vùng lõm sâu giữa hai điểm dữ liệu huấn luyện mà vùng này lại không thuộc hàm sinh dữ liệu. Giá trị của y cũng tăng rất mạnh về phía bên trái của dữ liệu, trong khi hàm sinh dữ liệu lại giảm ở vùng này.

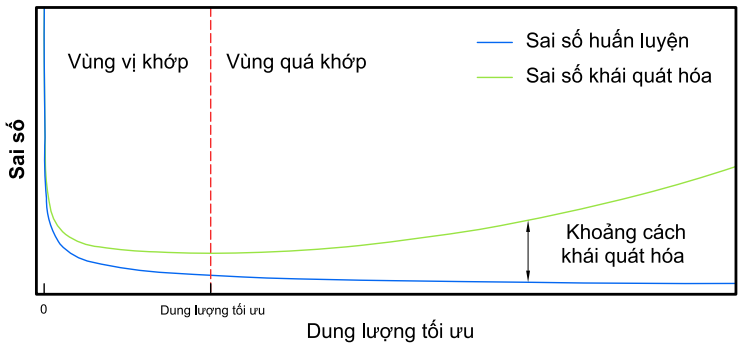
Đến đây, chúng tôi mới chỉ mô tả một cách để thay đổi capacity của một mô hình, bằng cách thay đổi số lượng đặc trưng đầu vào của nó đồng thời thêm vào các tham số mới có liên quan đến các đặc trưng đó. Trong thực tế, có rất nhiều cách để thay đổi capacity của một mô hình. Dung lượng của mô hình không chỉ được xác định bằng cách lựa chọn mô hình. Mô hình sẽ định ra họ các hàm số mà một thuật toán có thể học khi thay đổi các tham số nhằm mục đích giảm một mục tiêu huấn luyện. Họ các hàm số này còn được gọi là *dung lượng biểu diễn* (representational capacity) của mô hình. Trong nhiều trường hợp, việc tìm ra một hàm phù hợp nhất trong họ này là một bài toán tối ưu khó. Trong thực tế, thuật toán học không thực sự tìm ra hàm số tốt nhất, mà chỉ đơn thuần chọn ra hàm làm giảm sai số huấn luyện của mô hình một cách đáng kể. Có nghĩa là *dung lượng có hiệu lực* (effective capacity) của của một thuật toán học có thể thấp hơn dung lượng biểu diễn của họ mô hình.

Những ý tưởng hiện đại của ta về vấn đề cải thiện khả năng khái quát hóa các mô hình ML được đúc kết từ những quan điểm của các triết gia có niên đại từ thời Ptolemy. Ban đầu, nhiều học giả dẫn chứng một nguyên tắc phân tích được biết đến rộng rãi nhất cho đến nay là *dao cạo của Occam* (Occam's razor) (c. 1287–1347). Nguyên tắc này nói rằng trong số các giả thuyết cùng giải thích các quan sát đã biết rõ tốt như nhau, ta nên chọn giả thuyết “đơn giản nhất”. Ý tưởng này đã được định nghĩa một cách chặt chẽ và chính xác hơn vào thế kỷ hai mươi bởi những người sáng tạo lý thuyết học thống kê [Vapnik and Chervonenkis, 1971; Vapnik, 1982; Blumer et al., 1989; Vapnik, 1995].

Lý thuyết học thống kê cung cấp nhiều phương pháp khác nhau để định lượng hiệu năng mô hình. Trong số đó, phương pháp được biết nhiều nhất là *số chiều Vapnik-Chervonenkis* (Vapnik-Chervonenkis dimension), hoặc số chiều VC (VC dimemsion). Số chiều VC được sử dụng để đo lường hiệu năng của một bộ phân lớp nhị phân. Số chiều VC là giá trị lớn nhất có thể của m, sao cho tồn tại một bộ dữ liệu huấn luyện m điểm $\boldsymbol{x}$ mà bộ phân loại có thể gán nhãn tùy ý.

Việc định lượng capacity mô hình cho phép lý thuyết học thống kê đưa ra các dự đoán mang tính chất định lượng. Các kết quả quan trọng nhất của lý thuyết học thống kê chỉ ra rằng chênh lệch giữa sai số huấn luyện và sai số khái quát hóa bị chặn trên bởi một đại lượng mà nó sẽ tăng khi capacity của mô hình tăng nhưng sẽ giảm nếu tăng số lượng mẫu huấn luyện [Vapnik and Chervonenkis,1971; Vapnik, 1982; Blumer et al., 1989; Vapnik, 1995]. Những cận trên này cho ta cách giải thích tại sao thuật toán ML có thể hoạt động, nhưng chúng hiếm khi được sử dụng trong thực tế khi làm việc với những thuật toán DL. Điều này một phần vì các cận này thường khá lỏng lẻo, và một phần vì xác định capacity của các thuật toán học tập sâu là một bài toán khó. Xác định hiệu năng của một mô hình DL là bài toán đặc biệt khó bởi capacity có hiệu lực của mô hình bị giới hạn bởi khả năng của các thuật toán tối ưu, và ta có rất ít hiểu biết về lý thuyết của các bài toán tối ưu hoá không lồi liên quan đến DL.

ta cũng phải nhớ rằng trong khi các hàm đơn giản hơn có khả năng khái quát hóa (để khoảng cách giữa sai số huấn luyện và sai số kiểm thử là nhỏ) tốt hơn, ta vẫn phải chọn một giả thuyết đủ phức tạp để đạt được sai số huấn luyện thấp. Thông thường, sai số huấn luyện sẽ giảm cho đến khi nó tiệm cận với giá trị sai số tối thiểu có thể đạt tới khi tăng capacity của mô hình (giả định rằng sai số đo lường có giá trị tối thiểu). Thông thường, sai số khái quát hóa là một hàm của capacity mô hình có dạng chữ U. Điều này được minh hoạ ở hình vẽ 5.3.



Hình 5.3: Mối quan hệ thông thường giữa capacity mô hình và lỗi. Sai số trên tập huấn luyện và tập kiểm thử thay đổi theo hai xu hướng khác nhau. Ở phía bên trái của đường đồ thị, sai số huấn luyện và sai số khái quát hóa đều cao. Đây được gọi là trạng thái **underfitting**. Khi ta tăng capacity của mô hình lên, sai số huấn luyện cũng theo đó giảm dần, nhưng khoảng cách giữa sai số huấn luyện và sai số khái quát hóa lại tăng lên. Dần dần, lúc khoảng cách này vượt quá mức độ giảm ở sai số huấn luyện, ta rơi vào

> vùng **overfitting** - khi mà capacity mô hình là rất lớn, vượt quá capacity tối ưu.

Để đạt tới trường hợp cực đoan khi capacity lớn tuỳ ý, chúng tôi sẽ giới thiệu khái niệm về các *mô hình phi tham số*. Cho đến nay, ta mới chỉ biết đến các mô hình tham số, chẳng hạn như mô hình hồi quy tuyến tính. Các mô hình tham số sẽ học một hàm số được mô tả bởi các vector tham số với kích thước có hạn và cố định trước khi quan sát dữ liệu. Các mô hình phi tham số không có những giới hạn như vậy.

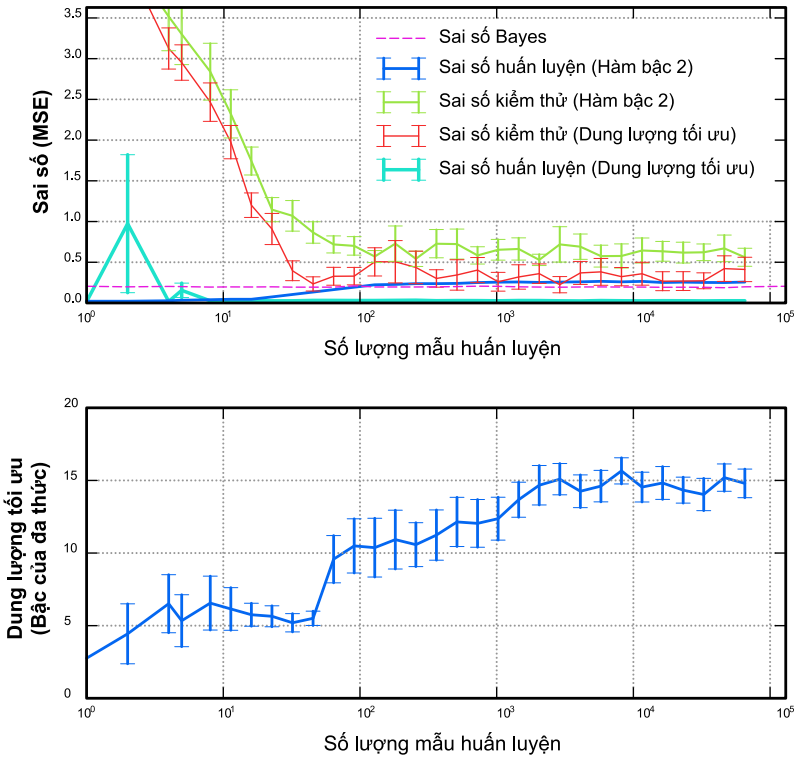
Đôi khi, các mô hình phi tham số chỉ là những thứ trừu tượng mang tính lý thuyết (chẳng hạn như một thuật toán tìm kiếm trên tất cả các phân phối xác suất có thể) mà không thể triển khai trong thực tế. Tuy nhiên, ta cũng có thể xây dựng các mô hình phi tham số thực tế hơn bằng cách đưa độ phức tạp của chúng trở thành một hàm số của kích thước tập huấn luyện. Một ví dụ về thuật toán như vậy là *hồi quy điểm gần nhất* (nearest neighbor regression). Không giống như hồi quy tuyến tính, có một vector trọng số với chiều dài cố định, mô hình hồi quy điểm gần nhất chỉ lưu trữ $\boldsymbol{X}$ và $\boldsymbol{y}$ từ dữ liệu huấn luyện. Khi được yêu cầu phân loại một điểm kiểm thử $\boldsymbol{x}$, mô hình tìm kiếm điểm gần nhất trong tập huấn luyện và trả về nhãn hồi quy tương ứng với điểm gần nhất này. Nói cách khác, $y^\intercal = y_i$ khi $i = \arg\min \|\boldsymbol{X}_{i,:} - \boldsymbol{x}\|_2$. Thuật toán này cũng có thể được khái quát hóa thành các thước đo khoảng cách khác với chuẩn L^2, chẳng hạn như các thước đo khoảng cách được học ra từ dữ liệu [Goldberger et al., 2005]. Nếu thuật toán được phép lấy trung bình giá trị y_i cho tất cả $\boldsymbol{X}_{i,:}$ khi có nhiều điểm cùng là gần nhất, thì thuật toán này có thể đạt được sai số huấn luyện nhỏ nhất (có thể lớn hơn 0, nếu hai đầu vào giống nhau được gắn với hai giá trị khác nhau) trên bất kỳ dữ liệu hồi quy nào.

Cuối cùng, ta có thể tạo một thuật toán học phi tham số bằng cách gói một thuật toán học có tham số bên trong một thuật toán khác có khả năng tăng số lượng tham số khi cần thiết. Ví dụ, ta có thể tưởng tượng một vòng lặp bên ngoài của quá trình học mà làm thay đổi bậc của đa thức được học bởi hồi quy tuyến tính trên một đa thức mở rộng của đầu vào.

Một mô hình lý tưởng là một *mô hình tiên tri* (oracle) có thể dễ dàng dự đoán được phân phối xác xuất thực sự sinh ra dữ liệu. Tuy nhiên, ngay cả một mô hình như vậy vẫn sẽ tồn tại những sai số, bởi vì trong phân phối vẫn có thể có một số nhiễu. Đối với học có giám sát, ánh xạ từ $\boldsymbol{x}$ đến y có thể mang bản chất ngẫu

nhiên, hoặc y có thể là một hàm tất định có liên quan đến các biến khác ngoài các biến đã được bao gồm trong $\boldsymbol{x}$. Sai số xuất hiện khi một mô hình tiên tri thực hiện các dự đoán từ phân phối thực sự $p(\boldsymbol{x}, y)$ được gọi là *sai số Bayes* (Bayes error).

Sai số huấn luyện và sai số khái quát hóa thay đổi khi kích thước tập huấn luyện thay đổi. Sai số tổng quát hóa kỳ vọng có thể không bao giờ tăng lên khi số lượng mẫu dữ liệu huấn luyện tăng lên. Đối với các mô hình phi tham số, càng nhiều dữ liệu càng cải thiện tính khái quát hóa của mô hình cho đến khi nó đạt được sai số thấp nhất có thể. Bất kỳ mô hình tham số cố định nào có capacity thấp hơn capacity tối ưu sẽ tiệm cận một giá trị sai số vượt quá sai số Bayes. Xem minh họa ở hình 5.4. Lưu ý rằng một mô hình có capacity tối ưu vẫn có thể có một khoảng cách lớn giữa sai số huấn luyện và sai số khái quát hóa. Trong tình huống này, ta có thể làm giảm khoảng cách nói trên bằng cách thu thập thêm mẫu huấn luyện.



Hình 5.4: Ảnh hưởng của kích thước tập dữ liệu huấn luyện đối với sai số tập huấn luyện và kiểm thử, cũng như capacity tối ưu của mô hình. Chúng tôi đã xây dựng một bài toán hồi quy tổng hợp dựa trên việc thêm một lượng nhiễu vừa phải vào một đa thức bậc 5, sinh ra chỉ một tập kiểm thử, sau đó sinh ra một số kích thước khác nhau cho tập huấn luyện. Đối với mỗi kích thước, chúng tôi tạo ra 40 tập huấn luyện khác nhau để vẽ một biểu đồ sai

> số với khoảng tin cậy 95%. (*Phía trên*) MSE trên dữ liệu huấn luyện và dữ liệu kiểm thử đối với hai mô hình khác nhau, một mô hình bậc hai và một mô hình với bậc được chọn để sai số kiểm thử nhỏ nhất. Cả hai mô hình đều được khớp ở dạng đóng. Đối với mô hình bậc hai, sai số huấn luyện tăng lên khi tăng kích thước của tập huấn luyện. Điều này là bởi các tập dữ liệu càng lớn thì mô hình càng khó khớp với dữ liệu. Đồng thời, sai số trên tập kiểm thử giảm vì có ít hơn các giả thuyết không chính xác trùng với giả thuyết trên dữ liệu huấn luyện. Mô hình bậc hai không có đủ capacity để giải quyết tác vụ này, do đó sai số kiểm thử của nó tiệm cận tới một giá trị cao. Sai số kiểm thử tại mô hình có capacity tối ưu sẽ tiệm cận tới sai số Bayes. Sai số huấn luyện có thể giảm xuống thấp hơn sai số Bayes, do thuật toán huấn luyện có khả năng ghi nhớ máy móc các mẫu dữ liệu trên tập huấn luyện. Khi kích thước của mẫu huấn luyện tăng đến vô cùng, sai số huấn luyện của bất cứ mô hình có capacity cố định nào (ở đây là mô hình bậc hai) sẽ tăng lên ít nhất bằng với sai số Bayes. (*Dưới*) khi dữ liệu của tập huấn luyện tăng lên, capacity tối ưu (ở đây thể hiện bằng bậc của bộ hồi quy đa thức tối ưu) cũng tăng, và đạt giá trị ổn định sau khi mô hình chạm đến một độ phức tạp đủ để giải quyết tác vụ ban đầu.

5.2.1 Luật bù trừ

Lý thuyết học cho rằng một thuật toán ML có thể khái quát tốt từ một tập hữu hạn các mẫu huấn luyện. Điều này có vẻ như mâu thuẫn với một số nguyên lý logic cơ bản. Lập luận quy nạp hoặc suy luận ra các quy tắc chung từ một tập hợp hữu hạn các mẫu, là không hợp lệ về mặt logic. Để suy luận một cách hợp lý một quy tắc mô tả mọi thành phần của một tập hợp, người ta phải có thông tin về các thành phần trong tập hợp đó.

Một phần, ML có thể tránh được vấn đề này bằng cách chỉ đưa ra các quy tắc xác suất, thay vì toàn bộ các quy tắc tất định được sử dụng trong lý luận thuần logic. ML hứa hẹn tìm ra các quy tắc *có thể* đúng với *hầu hết* các thành viên của tập dữ liệu mà chúng quan tâm.

Điều không may là ngay cả sử dụng các luật xác suất cũng không thể giải quyết được toàn bộ vấn đề. *Luật bù trừ* (no free lunch theorem) [Wolpert, 1996] trong ML chỉ ra rằng, tính trung bình trên tất cả các phân phối sinh dữ liệu có thể, mọi thuật toán phân lớp đều có cùng một tỷ lệ sai số khi phân loại các điểm chưa

từng được quan sát. Nói cách khác, theo một nghĩa nào đó, không có thuật toán ML nào luôn luôn tốt hơn hơn các thuật toán ML còn lại. Thuật toán phức tạp nhất ta có thể hình dung có cùng một hiệu suất trung bình (trên tất cả các tác vụ có thể) giống như các thuật toán chỉ đơn thuần dự đoán mọi điểm dữ liệu thuộc về cùng một lớp.

May mắn thay, những kết quả này chỉ đúng khi ta lấy trung bình trên *toàn bộ* các phân phối sinh dữ liệu . Nếu ta đưa ra những giả định về các loại phân phối xác suất mà ta gặp phải trong các ứng dụng thực tế, thì ta có thể thiết kế các thuật toán học hoạt động tốt trên những phân phối này.

Điều này có nghĩa rằng mục tiêu của nghiên cứu ML không phải là tìm kiếm một thuật toán học tập phổ quát hoặc thuật toán học tập tối ưu nhất trong mọi trường hợp. Thay vào đó, mục tiêu của ta là hiểu những loại phân phối nào liên quan đến "thế giới thực" mà một tác nhân AI trải nghiệm, và các loại thuật toán ML nào hoạt động tốt trên dữ liệu được lấy ra từ các loại phân phối sinh dữ liệu mà ta quan tâm.

5.2.2 Cơ chế kiểm soát

Luật bù trừ ngụ ý rằng ta phải thiết kế các thuật toán ML có thể hoạt động tốt trên một tác vụ cụ thể. Ta sẽ thực hiện điều điều đó bằng cách xây dựng một tập hợp các tuỳ chọn trong thuật toán học tập. Khi các tùy chọn này phù hợp với các bài toán học tập mà ta yêu cầu thuật toán giải quyết, thì nó hoạt động tốt hơn.

Từ đầu đến giờ, phương pháp duy nhất để sửa đổi thuật toán học tập mà ta đã thảo luận cụ thể là tăng hoặc giảm capacity biểu diễn của mô hình bằng cách thêm hoặc xóa bớt các hàm khỏi không gian giả thuyết của các giải pháp mà thuật toán học tập có thể lựa chọn. Chúng tôi đã đưa ra ví dụ cụ thể về việc tăng hoặc giảm bậc của đa thức cho một bài toán hồi quy. Quan điểm mà chúng tôi đã mô tả cho đến nay vẫn còn quá giản đơn.

Hành vi của thuật toán chịu ảnh hưởng mạnh mẽ không chỉ từ độ lớn của tập các hàm trong không gian giả thuyết của nó, mà còn bởi đặc điểm của các hàm đó. Thuật toán học tập mà ta đã nghiên cứu cho đến nay, hồi quy tuyến tính, có một không gian giả thuyết bao gồm tập hợp các hàm tuyến tính của đầu vào của nó. Các hàm tuyến tính này có thể hữu ích cho những bài toán mà mối quan hệ giữa đầu vào và đầu ra trong thực tế là gần tuyến tính. Những hàm này ít hữu ích

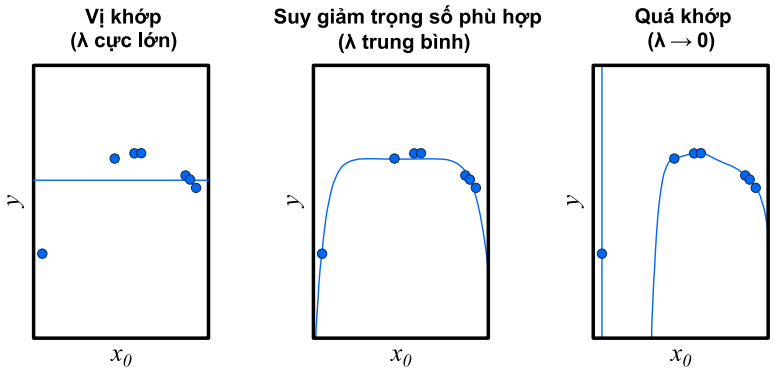
hơn đối với các bài toán phi tuyến tính. Ví dụ, hồi quy tuyến tính sẽ không hoạt động tốt nếu ta cố gắng sử dụng nó để dự đoán $\sin(x)$ từ x. Do đó, ta có thể kiểm soát hiệu suất của các thuật toán bằng cách chọn những loại hàm số nào mà ta cho phép thuật toán lấy các giải pháp từ đó, cũng như bằng cách kiểm soát số lượng các hàm này.

ta cũng có thể cho phép một thuật toán học tập ưu tiên một giải pháp hơn các giải pháp khác trong không gian giả thuyết của nó. Có nghĩa là cả hai hàm đều đủ điều kiện, nhưng một hàm sẽ được ưu tiên. Giải pháp không được ưu tiên sẽ chỉ được chọn nếu nó khớp với dữ liệu huấn luyện tốt hơn một cách đáng kể so với giải pháp ưu tiên.

Ví dụ, ta có thể sửa đổi tiêu chuẩn huấn luyện cho hồi quy tuyến tính để áp dụng *suy giảm trọng số* (weight decay). Để thực hiện hồi quy tuyến tính với suy giảm trọng số, ta cực tiểu hoá tổng của sai số bình phương trung bình trên tập huấn luyện và một tiêu chuẩn $J(\boldsymbol{w})$ thể hiện sự ưu tiên đối với các trọng số, để đạt giá trị bình phương của chuẩn L^2 nhỏ hơn.

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\text{train}} + \lambda \boldsymbol{w}^T \boldsymbol{w}, \qquad (5.18)$$

trong đó λ là một giá trị được chọn sẵn để kiểm soát độ ưu tiên đối với các trọng số nhỏ hơn. Khi $\lambda = 0$,ta không áp đặt ưu tiên, và λ càng lớn càng buộc các trọng số trở nên nhỏ hơn. Việc cực tiểu hoá $J(\boldsymbol{w})$ dẫn tới một sự đánh đổi (tradeoff) trong việc lựa chọn các trọng số, giữa việc hoặc khớp với dữ liệu huấn luyện hơn hoặc trở nên nhỏ hơn. Điều này mang lại cho ta những giải pháp có độ dốc nhỏ hơn, hoặc đặt trọng số trên ít đặc trưng hơn. Để minh họa về cách ta có thể điều khiển xu hướng của mô hình trở nên overfitting hoặc underfitting thông qua suy giảm trọng số, ta có thể huấn luyện mô hình hồi quy đa thức bậc cao với các giá trị khác nhau của λ. Xem hình 5.5 để thấy kết quả.



> Hình 5.5: ta khớp một mô hình hồi quy đa thức bậc cao với tập huấn luyện ví dụ hình 5.2. Hàm thực tế ứng với dữ liệu này là hàm bậc hai, nhưng ở

> đây ta chỉ sử dụng các mô hình với bậc 9. Ta thay đổi lượng suy giảm trọng số để tránh việc các mô hình bậc cao này gặp phải hiện tượng overfitting. (*Trái*) Với giá trị λ cực lớn, ta buộc mô hình học một hàm không có độ dốc. Lúc này underfitting xảy ra vì nó chỉ có thể biểu diễn một hàm hằng. (*Giữa*) Với giá trị λ trung bình, thuật toán học tập khôi phục một đường cong với hình dạng tổng quát phù hợp. Mặc dù mô hình có khả năng biểu diễn các hàm có hình dạng phức tạp hơn nhiều, nhưng việc suy giảm trọng số đã khiến nó sử dụng một hàm đơn giản được mô tả bởi các hệ số nhỏ hơn. (*Phải*) Với lượng suy giảm trọng số gần bằng 0 (nói cách khác, sử dụng giả ngịch đảo Moore-Penrose để giải bài toán vô định khi kiểm soát nhỏ), hàm đa thức bậc 9 trở nên overfitting, như ta đã thấy trong hình 5.2.

Tổng quát hơn, ta có thể kiểm soát một mô hình để học một hàm $f(\boldsymbol{x}; \boldsymbol{\theta})$ bằng cách thêm một đại lượng phạt gọi là *bộ kiểm soát* (regularizer) cho hàm chi phí. Trong trường hợp áp dụng suy giảm trọng số, bộ kiểm soát lúc này là $\Omega(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{w}$. Trong chương 7, ta sẽ thấy rằng có thể có nhiều bộ kiểm soát khác nhau.

Việc ưu tiên cho một hàm hơn một hàm khác là một cách tổng quát hơn để kiểm soát dung lượng của mô hình so với việc bao gồm hoặc loại trừ các thành phần ra khỏi không gian giả thuyết. Ta có thể coi việc loại trừ một hàm từ một không gian giả thuyết thể hiện một ưu tiên vô cùng mạnh chống lại hàm đó.

Trong ví dụ về suy giảm trọng số, chúng tôi đã thể hiện sự ưu tiên một cách rõ ràng với các hàm tuyến tính được xác định bởi các trọng số nhỏ hơn, thông qua một số hạng bổ sung trong biểu thức ta muốn cực tiểu hoá. Có nhiều cách khác, cả ngầm định lẫn rõ ràng, để thể hiện sự ưu tiên cho các giải pháp khác nhau. Các phương pháp tiếp cận này cùng được gọi là *cơ chế kiểm soát* (regularization). *Cơ chế kiểm soát là bất kỳ điều chỉnh nào ta thực hiện cho thuật toán học tập nhằm giảm sai số tổng quát hoá chứ không phải sai số huấn luyện của thuật toán đó*. Cơ chế kiểm soát là một trong những mối quan tâm trung tâm của ML, với tầm quan trọng không kém gì tối ưu hoá.

Quy luật bù trừ đã chỉ ra rằng không tồn tại thuật toán ML nào tối ưu, và đặc biệt là không có cơ chế kiểm soát nào là tốt nhất. Thay vào đó, ta phải chọn cơ chế kiểm soát phù hợp với từng tác vụ cụ thể ta giải quyết. Triết lý của DL nói chung và cuốn sách này nói riêng là: phần lớn các tác vụ (chẳng hạn như tất cả các tác

vụ sử dụng trí tuệ mà con người có thể làm) đều có thể được giải quyết một cách hiệu quả thông qua các cơ kiểm soát đa dụng.

5.3 Siêu tham số và tập xác thực

Hầu hết các thuật toán ML đều sử dụng các *siêu tham số* (hyperparameter), được thiết lập để ta có thể kiểm soát hành vi của thuật toán. Các giá trị của siêu tham số không được tự điều chỉnh bởi thuật toán (mặc dù ta có thể thiết kế một thủ tục học lồng nhau, mà trong đó một thuật toán học tập có thể học được bộ siêu tham số tốt nhất cho một thuật toán học tập khác).

Ví dụ về hồi quy đa thức ở hình 5.2 có một siêu tham số đơn đó là bậc của đa thức, có vai trò như một siêu tham số giúp kiểm soát *capacity*. Giá trị λ được sử dụng để kiểm soát mức độ suy giảm trọng số là một ví dụ khác về siêu tham số.

Đôi khi một cài đặt sẽ được chọn làm siêu tham số mà thuật toán học tập sẽ không học nó do cài đặt này rất khó tối ưu. Thường thì một cài đặt sẽ là một siêu tham số khi việc học siêu tham số đó từ tập huấn luyện là không thích hợp. Điều này được áp dụng với tất cả các siêu tham số kiểm soát capacity của mô hình. Lí do là nếu được học trên tập huấn luyện, các siêu tham số sẽ luôn chọn mô hình có capacity lớn nhất có thể, dẫn đến sự overfitting (tham khảo hình 5.3). Ví dụ, ta luôn có thể khớp tập huấn luyện tốt hơn với một đa thức bậc cao và với suy giảm trọng số $\lambda = 0$, thay vì một đa thức bậc thấp hơn có suy giảm trọng số là số dương.

Để giải quyết vấn đề này, ta cần một *tập xác thực* (validation set) của các mẫu dữ liệu mà thuật toán huấn luyện không thấy.

Trước đây, ta đã thảo luận về cách một tập kiểm thử được lấy riêng ra, bao gồm các mẫu dữ liệu đến từ cùng một phân phối như tập huấn luyện, có thể được sử dụng để ước tính sai số tổng quát hoá của một thuật toán học tập sau khi quá trình học đã hoàn tất. Điều quan trọng là các mẫu kiểm thử không được tham gia bằng bất kỳ cách nào trong quá trình đưa ra các lựa chọn của mô hình, bao gồm cả việc lựa các siêu tham số. Vì lý do này, không có mẫu dữ liệu nào từ tập kiểm thử có thể được sử dụng trong tập xác thực. Do đó, ta luôn phải xây dựng tập xác thực từ dữ liệu huấn luyện. Cụ thể, ta chia dữ liệu huấn luyện thành hai tập con riêng biệt. Một trong các tập con này được sử dụng để học các tham số. Tập hợp con còn lại chính là tập xác thực, được sử dụng để ước tính sai số tổng quát

hoá trong hoặc sau khi huấn luyện, cho phép các siêu tham số được cập nhật tương ứng. Tập dữ liệu con được sử dụng để học các tham số vẫn thường được gọi là tập huấn luyện, mặc dù cách gọi này có thể gây nhầm lẫn giữa chúng với nhóm dữ liệu lớn hơn được sử dụng cho toàn bộ quá trình huấn luyện. Tập dữ liệu con này được sử dụng như một sự hướng dẫn cho việc lựa chọn các siêu tham số được gọi là tập xác thực. Thông thường, người ta sử dụng khoảng 80 dữ liệu huấn luyện để huấn luyện và 20 để xác thực. Vì tập xác thực được sử dụng để "huấn luyện" các siêu tham số, sai số xác thực (validation error) sẽ thường nhỏ hơn sai số tổng quát hoá, dù lượng sai lệch giữa chúng thường nhỏ hơn lượng sai lệch so với sai số huấn luyện. Sau khi toàn bộ quá trình tối ưu siêu tham số được hoàn tất, sai số tổng quát hoá có thể được ước tính bằng cách sử dụng tập kiểm thử.

Trong thực tế, khi một tập kiểm thử được sử dụng nhiều lần để đánh giá hiệu suất của các thuật toán khác nhau trong nhiều năm, và đặc biệt là nếu ta xét tất cả các nỗ lực từ cộng đồng khoa học trong việc đánh bại hiệu suất tốt nhất hiện tại (state-of-the-art) đã được công bố đối với tập kiểm thử này trước đó, cuối cùng ta cũng sẽ có những đánh giá quá lạc quan với tập kiểm thử. Kết quả là các tập dữ liệu chuẩn có thể trở nên lỗi thời và không còn phản ánh hiệu suất thực sự của một hệ thống được huấn luyện. Rất may, cộng đồng có xu hướng chuyển sang các tập dữ liệu quy chuẩn mới (thường lớn hơn và tham vọng hơn).

5.3.1 Kiểm chứng chéo

Việc chia tập dữ liệu thành một tập huấn luyện cố định và một tập kiểm thử cố định có thể có vấn đề nếu tập kiểm thử là nhỏ. Một tập kiểm thử nhỏ mang đến sự bất định về mặt thống kê xung quanh sai số kiểm thử trung bình được ước tính, gây khó khăn để xác định liệu thuật toán A có hoạt động tốt hơn thuật toán B trong tác vụ đã cho hay không.

Khi tập dữ liệu có hàng trăm nghìn mẫu trở lên, đây không phải là vấn đề nghiêm trọng. Khi tập dữ liệu quá nhỏ, các thủ tục luân phiên nhau cho phép người dùng sử dụng toàn bộ các mẫu sử dụng trong ước lượng sai số kiểm thử trung bình, với cái giá là phải tăng chi phí tính toán. Những thủ tục này dựa trên ý tưởng lặp lại phép tính toán huấn luyện và kiểm thử trên các tập con được chọn ngẫu nhiên hoặc tách ra từ tập dữ liệu gốc. Phổ biến nhất trong số này là thủ tục xác thực chéo *k-gói*, được mô tả trong thuật toán 5.1, trong đó một phân hoạch của tập dữ

liệu được tạo thành bằng cách chia nó thành k tập con không chồng chéo nhau. Sau đó, sai số kiểm thử được ước tính bằng cách lấy sai số kiểm thử trung bình sau k phép thử. Trong phép thử i, tập hợp con thứ i của dữ liệu được sử dụng làm tập kiểm thử và phần còn lại của dữ liệu được sử dụng làm tập huấn luyện. Có một vấn đề là không tồn tại các hàm ước lượng phương sai không bị chệch về mặt thống kê (unbiased estimator) của các hàm ước lượng sai số trung bình đó [Bengio và Grandvalet, 2004], tuy nhiên, ta vẫn sử dụng các phép tính xấp xỉ để ước lượng.

Thuật toán 5.1 Thuật toán kiểm chứng chéo k-gói. Thuật toán này có thể được sử dụng nhằm ước lượng lỗi tổng quát của thuật toán học A khi tập dữ liệu $\mathbb{D}$ đã biết quá nhỏ để tách thành một tập huấn luyện/kiểm thử hay huấn luyện/xác thực để đưa ra ước lượng chính xác lỗi tổng quát, bởi vì trung bình của mất mát L trên tập dữ liệu kiểm thử có phương sai quá cao. Tập dữ liệu $\mathbb{D}$ bao gồm các phần tử trừu tượng $z^{(i)}$ (cho mẫu dữ liệu thứ i); phần tử này có thể là một cặp (đầu vào, mục tiêu) $z^{(i)} = \left(x^{(i)}, y^{(i)}\right)$ trong trường hợp học có giám sát, hoặc chỉ là một đầu vào $z^{(i)} = x^{(i)}$ trong trường hợp học không giám sát. Thuật toán trả về vector các lỗi $\boldsymbol{e}$ với mỗi mẫu dữ liệu trong $\mathbb{D}$, với ý nghĩa tương ứng là sai số tổng quát ước lượng. Các lỗi trên các mẫu dữ liệu riêng biệt có thể được sử dụng để tính toán khoảng tin cậy xung quanh giá trị trung bình (phương trình (5.47)). Mặc dù những khoảng tin cậy này không hẳn là hợp lệ về mặt lý thuyết khi ta thực hiện kiểm chứng chéo, nó vẫn được sử dụng phổ biến trong thực tế để công bố rằng thuật toán A là tốt hơn thuật toán B chỉ khi khoảng tin cậy của lỗi của thuật toán A nằm bên dưới và không giao cắt với khoảng tin cậy của thuật toán B.

> **Định nghĩa** $\mathrm{KFoldXV}(\mathbb{D}, A, L, k)$:
> **Yêu cầu:** $\mathbb{D}$, tập dữ liệu cho trước, với các phần tử $z^{(i)}$
> **Yêu cầu:** A, thuật toán học tập lấy tập dữ liệu đầu vào và xuất ra một hàm học được.
> **Yêu cầu:** L, hàm mất mát là một hàm ánh xạ từ hàm học f và một mẫu dữ liệu $z^{(i)} \in \mathbb{D}$ tới một số vô hướng $\in R$
> **Yêu cầu:** k, số lượng các gói
>
> Chia $\mathbb{D}$ thành k tập con riêng biệt $\mathbb{D}_i$ có hợp là $\mathbb{D}$

> **for** i from 1 to k **do**
> $f_i = A\left(\mathbb{D} \backslash \mathbb{D}_i\right)$
> **for** $z^{(j)}$ in $\mathbb{D}_i$ **do**
> $e_j = L\left(f_i, z^{(j)}\right)$
> end for
> end for
> Return $\boldsymbol{e}$

5.4. Ước lượng, độ chệch và phương sai

Lĩnh vực thống kê cung cấp cho ta nhiều công cụ để đạt được mục tiêu ML là giải quyết một tác vụ không chỉ trên tập huấn luyện mà còn trong khả năng khái quát hoá. Các khái niệm nền tảng như ước lượng tham số, độ chệch và phương sai rất hữu ích để mô tả một cách chặt chẽ các khái niệm về khái quát hóa, underfitting và overfitting.

5.4.1. Ước lượng điểm

Ước lượng điểm là nỗ lực nhằm đưa ra một dự đoán tốt nhất cho một số đại lượng mà ta quan tâm. Nói chung, đại lượng ta quan tâm có thể là một đơn tham số hoặc một vector các tham số trong mô hình, chẳng hạn như các trọng số trong ví dụ về hồi quy tuyến tính ở mục 5.1.4, nhưng nó cũng có thể là một hàm số.

Để phân biệt các ước lượng tham số với giá trị thực của chúng, chúng tôi quy ước ký hiệu một điểm ước lượng của một tham số $\boldsymbol{\theta}$ là $\hat{\boldsymbol{\theta}}$.

Gọi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ là tập hợp của m điểm dữ liệu độc lập và có cùng phân phối (i.i.d). Một *ước lượng điểm* (point estimator) hoặc một *thống kê* là một hàm số bất kỳ của dữ liệu:

$$\hat{\boldsymbol{\theta}}_m = g\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right) \qquad (5.19)$$

Định nghĩa không yêu cầu g trả về một giá trị gần với giá trị $\boldsymbol{\theta}$ thực tế, thậm chí là không yêu cầu khoảng giá trị của g là giống với tập các giá trị cho phép của $\boldsymbol{\theta}$.

Định nghĩa này của một ước lượng điểm là rất khái quát, và nó cho phép ta thiết kế các ước lượng một cách linh hoạt. Do đó, mặc dù gần như bất kỳ hàm số nào cũng đủ tiêu chuẩn để trở thành một ước lượng, nhưng một ước lượng tốt là một hàm mà đầu ra của nó gần với giá trị thực của $\boldsymbol{\theta}$ sinh bởi dữ liệu huấn luyện.

Ở thời điểm này, chúng tôi sử dụng quan điểm của trường phái tần suất về thống kê. Giả sử rằng giá trị tham số thực tế $\boldsymbol{\theta}$ là cố định nhưng chưa biết, trong khi ước lượng điểm $\hat{\boldsymbol{\theta}}$ là một hàm số của dữ liệu. Vì dữ liệu được xây dựng từ một quá trình ngẫu nhiên, nên bất kỳ hàm số nào của dữ liệu cũng là ngẫu nhiên. Do đó, $\hat{\boldsymbol{\theta}}$ là một biến ngẫu nhiên.

Ước lượng điểm cũng có thể bao gồm ước lượng mối quan hệ giữa các biến đầu vào và các biến mục tiêu. Ta gọi những kiểu ước lượng điểm đó là các ước lượng hàm (function estimator).

Ước lượng hàm Đôi khi ta muốn thực hiện việc ước lượng hàm (hoặc xấp xỉ hàm). Ở đó, ta có thể thử dự đoán một biến y khi cho trước một vector đầu vào $\boldsymbol{x}$. Ta giả thiết rằng có một hàm $f(\boldsymbol{x})$ mô tả mối quan hệ xấp xỉ giữa $\boldsymbol{y}$ và $\boldsymbol{x}$. Ví dụ, ta có thể giả thiết rằng $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$ với $\boldsymbol{\epsilon}$ là một phần không thể dự đoán được của $\boldsymbol{y}$ từ $\boldsymbol{x}$. Trong ước lượng hàm, ta quan tâm tới việc xấp xỉ f với một mô hình, hoặc hàm ước lượng $\hat{f}$. Ước lượng hàm thực sự giống với việc ước lượng một tham số $\boldsymbol{\theta}$; ước lượng hàm $\hat{f}$ đơn giản là một ước lượng điểm trong không gian hàm. Ví dụ về hồi quy tuyến tính (thảo luận trong mục 5.1.4) và hồi quy đa thức (thảo luận trong mục 5.2) đều minh họa các kịch bản mà ta có thể diễn giải như là ước lượng một tham số $\boldsymbol{w}$ hoặc ước lượng một hàm $\hat{f}$ ánh xạ từ $\boldsymbol{x}$ tới $\boldsymbol{y}$.

Bây giờ chúng tôi sẽ điểm qua các thuộc tính được nghiên cứu phổ biến nhất của các ước lượng điểm và thảo luận về những gì chúng cho ta biết về các ước lượng này.

5.4.2. Độ chệch

Độ chệch (bias) của một hàm ước lượng được định nghĩa như sau

$$\text{bias}\left(\hat{\boldsymbol{\theta}}_m\right) = \mathbb{E}\left(\hat{\boldsymbol{\theta}}_m\right) - \boldsymbol{\theta} \qquad (5.20)$$

trong đó kỳ vọng được lấy đối với dữ liệu (coi như là mẫu được lấy ra từ một biến ngẫu nhiên) và $\boldsymbol{\theta}$ là giá trị thực của $\boldsymbol{\theta}$ được sử dụng để định nghĩa phân phối sinh dữ liệu. Một ước lượng $\hat{\boldsymbol{\theta}}_m$ được coi là *không chệch* (unbiased) khi $\text{bias}\left(\hat{\boldsymbol{\theta}}_m\right) = 0$

, từ đó suy ra $\mathbb{E}\left(\hat{\boldsymbol{\theta}}_m\right) = \boldsymbol{\theta}$. Một ước lượng $\hat{\boldsymbol{\theta}}_m$ được coi là *tiệm cận không chệch* nếu $\lim_{m\to\infty} \text{bias}\left(\hat{\boldsymbol{\theta}}_m\right) = 0$, từ đó suy ra $\lim_{m\to\infty}\left(\hat{\boldsymbol{\theta}}_m\right) = \boldsymbol{\theta}$.

Ví dụ: Phân phối Bernoulli Xét một tập các mẫu $\{x^{(1)}, \ldots, x^{(m)}\}$ độc lập với nhau và cùng được phân phối theo phân phối Bernoulli với giá trị trung bình θ:

$$P\left(x^{(i)}; \theta\right) = \theta^{x^{(i)}}(1-\theta)^{\left(1-x^{(i)}\right)} \qquad (5.21)$$

Một ước lượng phổ biến cho tham số θ của phân phối này là trung bình của các mẫu huấn luyện:

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad (5.22)$$

Để xác định liệu hàm ước lượng này có bị chệch hay không, ta có thể thay thế phương trình 5.22 vào phương trình 5.20:

$$\begin{aligned}
\text{bias}\left(\hat{\theta}_m\right) &= \mathbb{E}\left[\hat{\theta}_m\right] - \theta && (5.23)\\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \theta && (5.24)\\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right] - \theta && (5.25)\\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{x^{(i)}=0}^{1}\left(x^{(i)}\theta^{x^{(i)}}(1-\theta)^{1-x^{(i)}}\right) - \theta && (5.26)\\
&= \frac{1}{m}\sum_{i=1}^{m}(\theta) - \theta && (5.27)\\
&= \theta - \theta = 0 && (5.28)
\end{aligned}$$

Kết quả là $\text{bias}\,(\theta) = 0$, nên ta nói rằng ước lượng $\hat{\theta}$ là không chệch.

Ví dụ: Ước lượng giá trị trung bình của phân phối Gauss

Xét một tập mẫu $\left\{x^{(1)}, \ldots, x^{(m)}\right\}$ độc lập với nhau và cùng được phân phối theo một phân phối Gaussian $p\left(x^{(i)}\right) = \mathcal{N}\left(x^{(i)}; \mu, \sigma^2\right)$

với $i \in \{1, \ldots, \mathrm{m}\}$. Hàm mật độ xác suất của phân phối Gauss được xác định bởi:

$$p\left(x^{(i)}; \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu\right)^2}{\sigma^2}\right) \qquad (5.29)$$

Ước lượng phổ biến cho tham số kì vọng của phân phối Gauss là *trung bình mẫu* (sample mean):

$$\hat{\mu}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad (5.30)$$

Để xác định độ chệch của trung bình mẫu, ta lại tính toán kỳ vọng của nó.

$$\begin{aligned}
\text{bias}\left(\hat{\mu}_m\right) &= \mathrm{E}\left[\hat{\mu}_m\right] - \mu && (5.31)\\
&= \mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \mu && (5.32)\\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left[x^{(i)}\right]\right) - \mu && (5.33)\\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mu\right) - \mu && (5.34)\\
&= \mu - \mu = 0 && (5.35)
\end{aligned}$$

Như vậy, ta thấy rằng trung bình mẫu là một hàm ước lượng không chệch cho kỳ vọng của phân phối Gauss.

Ví dụ: Các ước lượng của phương sai của một phân phối Gauss

Trong ví dụ này, ta so sánh hai ước lượng khác nhau của tham số phương sai σ^2 của phân phối Gauss. Ta muốn biết xem ước lượng nào là ước lượng bị chệch.

Ước lượng đầu tiên của σ^2 ta xét tới là *phương sai mẫu* (sample variance)

$$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2 \qquad (5.36)$$

trong đó $\hat{\mu}_m$ là trung bình mẫu. Ta sẽ tính:

$$\text{bias}\left(\hat{\sigma}_m^2\right) = \mathbb{E}\left[\hat{\sigma}_m^2\right] - \sigma^2 \qquad (5.37)$$

Ta bắt đầu bằng việc tính số hạng $\mathbb{E}\left[\hat{\sigma}_m^2\right]$:

$$\begin{aligned}
\mathbb{E}\left[\hat{\sigma}_m^2\right] &= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] && (5.38)\\
&= \frac{m}{m-1}\sigma^2 && (5.39)
\end{aligned}$$

Quay trở lại phương trình 5.37, ta kết luận rằng độ chệch của $\hat{\sigma}_m^2$ là $-\sigma^2/m$. Do đó, phương sai mẫu là một ước lượng có chệch.

Ước lượng *phương sai mẫu không chệch*

$$\tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2 \qquad (5.40)$$

đưa ra một hướng tiếp cận khác. Giống như cái tên, đây là một ước lượng không chệch. Có nghĩa là $\mathbb{E}\left[\tilde{\sigma}_m^2\right] = \sigma^2$:

$$\begin{aligned}
\mathbb{E}\left[\tilde{\sigma}_m^2\right] &= \mathbb{E}\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] && (5.41)\\
&= \frac{m}{m-1}\mathbb{E}\left[\tilde{\sigma}_m^2\right] && (5.42)\\
&= \frac{m}{m-1}\left(\frac{m-1}{m}\sigma^2\right) && (5.43)\\
&= \sigma^2 && (5.44)
\end{aligned}$$

Ta có hai ước lượng: một bị chệch, một không chệch. Rõ ràng, ta mong muốn tìm được những ước lượng không chệch, tuy nhiên chúng không phải lúc nào cũng là các ước lượng "tốt nhất". Thông thường, ta sẽ lựa chọn sử dụng các ước lượng lệch nhưng lại sở hữu những thuộc tính quan trọng khác.

5.4.3 Phương sai và sai số chuẩn

Một thuộc tính khác của hàm ước lượng mà ta có thể muốn xét tới là mức độ biến thiên của nó khi nó là một hàm của mẫu dữ liệu. Giống như khi tính toán kỳ vọng của một ước lượng để xác định độ chệch của nó, ta có thể tính toán phương sai của ước lượng đó. *Phương sai* (variance) của một ước lượng chỉ đơn giản là phương sai

$$\mathrm{Var}(\hat{\theta}) \qquad (5.45)$$

trong đó biến ngẫu nhiên là tập huấn luyện. Ngoài ra, căn bậc hai của phương sai được gọi là *sai số chuẩn* (standard error), ký hiệu là $\mathrm{SE}(\hat{\theta})$.

Phương sai, hoặc sai số chuẩn, của một ước lượng cung cấp cho ta một thước đo mức độ biến thiên của một ước lượng mà ta tính toán từ dữ liệu khi tái lấy mẫu bộ dữ liệu đó một cách độc lập từ quá trình sinh dữ liệu cơ sở. Cũng giống như việc ta muốn có một ước lượng với độ chệch nhỏ, ta cũng muốn ước lượng này có phương sai nhỏ.

Khi tính toán bất kỳ thống kê nào với một số lượng mẫu hữu hạn, kết quả ước lượng tham số của ta là bất định, theo nghĩa là có thể ta sẽ thu được các mẫu khác hẳn từ cùng một phân phối và do đó các thống kê của chúng cũng sẽ khác. Mức độ biến thiên kì vọng trong bất kì hàm ước lượng nào chính là một nguồn gây sai số mà ta muốn định lượng.

Sai số chuẩn của giá trị trung bình được tính như sau:

$$\mathrm{SE}(\hat{\mu}_m) = \sqrt{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}, \qquad (5.46)$$

trong đó σ^2 là phương sai thực sự của mẫu x^i. Sai số chuẩn thường được ước lượng bằng cách sử dụng một ước lượng của σ. Không may mắn là cả căn bậc hai của phương sai mẫu, lẫn căn bậc hai của ước lượng không chệch của phương sai đều không thể tạo ra một ước lượng không chệch cho độ lệch chuẩn. Cả hai cách tiếp cận này đều có xu hướng tạo ra một ước lượng có giá trị thấp hơn độ lệch chuẩn thực sự, nhưng chúng vẫn được sử dụng trong thực tiễn. Căn bậc hai ước lượng không chệch của phương sai sẽ gần với độ lệch chuẩn hơn. Đối với m lớn, đại lượng xấp xỉ này trở nên khá phù hợp.

Sai số chuẩn của giá trị trung bình rất hữu ích trong các thí nghiệm ML. Ta thường ước lượng sai số tổng quát hoá bằng cách tính trung bình mẫu của sai số trên tập kiểm thử. Số lượng mẫu trong tập kiểm thử quyết định độ chính xác của ước lượng này. Định lý giới hạn trung tâm (central limit theorem) cho ta biết rằng giá trị trung bình sẽ được phân bố theo một phân phối xấp xỉ chuẩn. Áp dụng định lý này, ta có thể sử dụng sai số chuẩn để tính xác suất mà kỳ vọng thực sự rơi vào bất kỳ khoảng được chọn nào. Chẳng hạn, khoảng tin cậy 95 phần trăm xung quanh giá trị trung bình $\hat{\mu}_m$ là

$$(\hat{\mu}_m - 1.96 \times \mathrm{SE}(\hat{\mu}_m),\ \hat{\mu}_m + 1.96 \times \mathrm{SE}(\hat{\mu}_m)), \qquad (5.47)$$

theo phân phối chuẩn với trung bình là $\hat{\mu}_m$ và phương sai $\mathrm{SE}(\hat{\mu}_m)^2$. Trong các thí nghiệm ML, người ta thường nói thuật toán A tốt hơn thuật toán B nếu cận trên của khoảng tin cậy 95 phần trăm cho sai số của thuật toán A nhỏ hơn cận dưới của khoảng tin cậy 95 phần trăm cho sai số của thuật toán B.

Ví dụ: Phân phối Bernoulli Một lần nữa ta xem xét một tập hợp các mẫu $\{x^{(1)}, ..., x^{(m)}\}$ được lấy độc lập từ cùng một phân phối Bernoulli (nhớ lại rằng $P(x^{(i)}; \theta) = \theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}$). Lần này ta quan tâm đến vấn đề tính toán phương sai của ước lượng $\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$.

$$\text{Var}(\hat{\theta}_m) = \text{Var}(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}) \tag{5.48}$$

$$= \frac{1}{m^2}\sum_{i=1}^{m} \text{Var}(x^{(i)}) \tag{5.49}$$

$$= \frac{1}{m^2}\sum_{i=1}^{m} \theta(1-\theta) \tag{5.50}$$

$$= \frac{1}{m^2} m\theta(1-\theta) \tag{5.51}$$

$$= \frac{1}{m}\theta(1-\theta) \tag{5.52}$$

Phương sai của ước lượng giảm xuống theo một hàm số của m, với m là số lượng mẫu trong tập dữ liệu. Đây là một tính chất chung của các ước lượng phổ biến mà ta sẽ quay lại khi thảo luận về tính nhất quán ở phần 5.4.5.

5.4.4 Đánh đổi giữa độ chệch và phương sai để cực tiểu hóa trung bình của bình phương sai số.

Độ chệch và phương sai đo lường hai nguồn sai số khác nhau trong một ước lượng. Độ chệch đo độ lệch kì vọng từ giá trị thực của hàm hoặc tham số. Mặt khác, phương sai cho phép đo độ lệch từ ước lượng kì vọng mà bất kỳ phép lấy mẫu cụ thể nào của dữ liệu có khả năng gây ra.

Điều gì sẽ xảy ra khi ta phải đưa ra một lựa chọn giữa hai ước lượng, một với độ chệch lớn hơn và một với phương sai lớn hơn? Chẳng hạn, giả sử ta muốn xấp xỉ hàm số được đưa ra trong hình 5.2, và ta chỉ được chọn lựa giữa một mô hình có độ chệch lớn và một mô hình có phương sai lớn. Ta sẽ chọn như thế nào?

Cách phổ biến nhất để đối mặt với sự đánh đổi này là sử dụng kiểm chứng chéo. Theo kinh nghiệm, kiểm chứng chéo là phương pháp rất thành công trên nhiều tác vụ thực tế. Ngoài ra, ta cũng có thể so sánh *sai số bình phương trung bình* (mean squared error - MSE) của các các ước lượng:

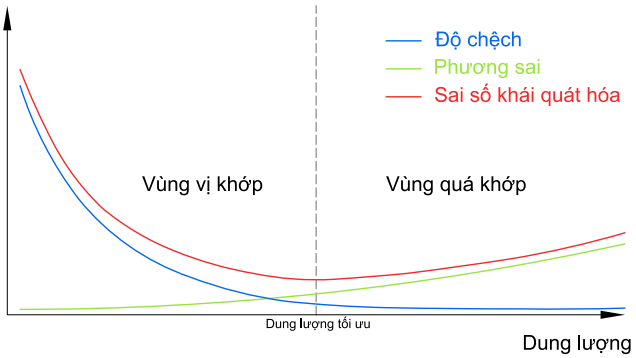
$$\text{MSE} = \text{E}[(\hat{\theta}_m - \theta)^2] \tag{5.53}$$

$$= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m) \tag{5.54}$$

MSE đo lường độ lệch kì vọng tổng thể - theo nghĩa bình phương của sai số - giữa ước lượng và giá trị thực của tham số θ. Từ phương trình 5.54, rõ ràng việc

đánh giá MSE kết hợp chặt chẽ cả độ chệch và phương sai. Các ước lượng ta mong muốn là các ước lượng có MSE nhỏ, và đó là những ước lượng có thể kiểm soát sự cân bằng giữa phương sai và độ chệch.

Sự liên hệ giữa độ chệch và phương sai có mối liên quan chặt chẽ tới các khái niệm capacity, overfitting và underfitting trong ML. Khi sai số tổng quát hoá được đo bởi MSE (trong đó độ chệch và phương sai là các thành phần có ảnh hướng tới sai số tổng quát hoá), việc tăng capacity có xu hướng làm tăng phương sai và giảm độ chệch. Điều này được minh họa trong hình 5.6, khi ta lại thấy đường cong hình chữ U của sai số tổng quát hoá với vai trò là một hàm của capacity.



> *Hình 5.6*: Khi capacity của mô hình tăng (trên trục x), độ chệch (đường chấm xanh dương) có xu hướng giảm và phương sai (đường gạch đứt màu lục) có xu hướng tăng, chúng phối hợp tạo nên đường cong chữ U biểu diễn sai số tổng quát hoá (đường đỏ đậm). Nếu ta tăng giảm capacity theo trục x , sẽ tồn tại giá trị capacity tối ưu, sao cho mô hình sẽ underfitting nếu capacity thấp hơn giá trị này, và sẽ overfitting nếu capacity cao hơn giá trị này. Mối quan hệ này tương tự như mối quan hệ giữa capacity, underfitting, overfitting, đã bàn đến trong phần 5.2 và hình 5.3.

5.4.5 Tính nhất quán

Cho đến thời điểm này, ta đã thảo luận về các thuộc tính của các ước lượng khác nhau cho một tập huấn luyện kích thước cố định. Thông thường, ta cũng quan tâm đến hành vi của một ước lượng khi số lượng dữ liệu huấn luyện tăng lên. Cụ thể, khi số lượng điểm dữ liệu m trong bộ dữ liệu tăng lên, ước lượng điểm của ta sẽ hội tụ đến giá trị thực của các tham số tương ứng. Nói một cách chặt chẽ hơn, ta sẽ muốn

$$\text{plim}_{m\to\infty}\hat{\theta}_m = \theta \qquad (5.55)$$

Ký hiệu plim biểu thị sự hội tụ theo xác suất, có nghĩa là đối với bất kỳ $\epsilon > 0$ nào,

$$P(|\hat{\theta}_m - \theta| > \epsilon) \to 0$$

khi $m \to \infty$. Điều kiện được mô tả bởi phương trình (5.55) được gọi là *tính nhất quán* (consistency). Đôi khi, nó được gọi là tính nhất quán yếu; tương ứng, tính nhất quán mạnh đề cập đến sự hội tụ *gần như chắc chắn* (almost sure) của $\hat{\theta}$ đến $\boldsymbol{\theta}$. *Hội tụ gần như chắc chắn* (almost sure convergency) của một chuỗi các biến ngẫu nhiên $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ tới một giá trị x xảy ra khi $p(\lim_{m\to\infty} \mathbf{x}^{(m)} = x) = 1$.

Tính nhất quán đảm bảo rằng độ chệch do hàm ước lượng gây ra sẽ giảm đi khi số lượng mẫu dữ liệu tăng lên. Tuy nhiên, điều ngược lại không đúng - một hàm ước lượng không chệch về mặt tiệm cận không có nghĩa là nó có tính nhất quán. Ví dụ, xét phép ước lượng tham số trung bình μ của một phân phối chuẩn $\mathcal{N}(x; \mu, \sigma^2)$, với một tập dữ liệu bao gồm m mẫu: $\{x^{(1)}, ..., x^{(m)}\}$. Ta có thể sử dụng mẫu đầu tiên $x^{(1)}$ của tập dữ liệu như một ước lượng không chệch: $\hat{\theta} = x^{(1)}$. Trong trường hợp đó, $\mathbb{E}(\hat{\theta}_m) = \theta$, vì vậy ước lượng là không chệch cho dù có bao nhiêu điểm dữ liệu được quan sát đi chăng nữa. Hiển nhiên, điều này mang ý nghĩa ước lượng là không chệch theo tiệm cận. Tuy nhiên, đây không phải là một ước lượng nhất quán vì nó không thỏa mãn trường hợp $\hat{\theta}_m \to \theta$ khi $m \to \infty$.

5.5 Maximum likelihood estimation

ta đã xem xét một số định nghĩa của các ước lượng phổ biến và phân tích các tính chất của chúng. Nhưng những ước lượng này từ đâu mà có? Thay vì đoán mò một vài hàm có thể tạo ra một ước lượng tốt, rồi sau đó phân tích độ chệch và phương sai của nó, ta cần có một số nguyên lý để từ đó có thể suy ra các hàm cụ thể mà chúng có thể là các ước lượng tốt cho các mô hình khác nhau.

Một trong những nguyên tắc phổ biến nhất đó là *ước lượng hợp lý cực đại* (maximum likelihood estimation).

Xét một tập hợp gồm m mẫu $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ được lấy độc lập từ một phân phối sinh dữ liệu thực sự nhưng chưa biết $p_{data}(\mathbf{x})$.

Đặt $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ là một họ phân phối xác suất trên cùng không gian được tham số hóa bởi $\boldsymbol{\theta}$. Nói cách khác, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ ánh xạ bất kỳ một mẫu $\boldsymbol{x}$ nào tới một số thực ước lượng xác suất thực sự $p_{\text{data}}(\boldsymbol{x})$.

Ước lượng hợp lý tối đa cho $\boldsymbol{\theta}$ sau đó được định nghĩa như sau:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{5.56}$$

$$= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{5.57}$$

Kết quả này là tích của nhiều nhiều xác suất, vì thế có thể gây ra bất tiện bởi nhiều lý do. Chẳng hạn, nó có xu hướng gây ra thấp luồng (underflow). Để thu được một bài toán tối ưu tương đương nhưng thuận tiện để giải quyết hơn, ta thấy rằng việc lấy logarit của hàm hợp lý không thay đổi giá trị tối đa của $\boldsymbol{\theta}$ nhưng lại dễ dàng biến đổi một tích thành một tổng:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{i}; \boldsymbol{\theta}) \tag{5.58}$$

Vì hàm $\arg\max$ không thay đổi khi ta tái cân chỉnh tỉ lệ hàm chi phí (tức là nhân với một tỉ số không đổi), nên ta có thể chia hàm chi phí cho m để thu được một biểu thức của giá trị kỳ vọng đối với phân phối thực nghiệm $\hat{p}_{\mathrm{data}}$ xác định bởi dữ liệu huấn luyện:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\hat{p}_{\mathrm{data}}} \log\, p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{5.59}$$

Một cách để diễn giải phương pháp ước lượng hợp lý cực đại là xem nó như một quá trình tối thiểu hóa sự sai khác giữa phân phối thực nghiệm, xác định bởi tập huấn luyện và phân phối của mô hình, với mức độ sai khác giữa hai phân phối được đo bằng độ phân kỳ KL. Độ phân kỳ KL được tính như sau:

$$D_{KL}(\hat{p}_{\mathrm{data}} || p_{\mathrm{model}}) = \mathbb{E}_{\mathbf{x}\hat{p}_{\mathrm{data}}}[\log\, \hat{p}_{data}(\boldsymbol{x}) - \log\, p_{\mathrm{model}}(\boldsymbol{x})] \tag{5.60}$$

> ***ND***: Độ phân kì KL được mô tả chi tiết hơn trong hình 2.10

Số hạng bên trái là một hàm chỉ ứng với quá trình sinh dữ liệu chứ không phải là mô hình. Có nghĩa là khi ta huấn luyện mô hình để cực tiểu hoá độ phân kỳ KL, ta chỉ cần cực tiểu hoá

$$-\mathbb{E}_{\mathbf{x}\hat{p}_{\mathrm{data}}}[\log\, p_{\mathrm{model}}(\boldsymbol{x})] \tag{5.61}$$

việc này cũng tương đương với tìm cực đại trong phương trình (5.59).

Cực tiểu hóa độ phân kỳ KL tương ứng với cực tiểu hoá entropy chéo giữa các phân phối. Nhiều tác giả sử dụng thuật ngữ “entropy chéo” để chỉ cụ thể *hàm đối của logarit hợp lý* (negative log-likelihood) của một phân phối Bernoulli hoặc phân phối softmax, nhưng đó là một sai lầm trong cách dùng thuật ngữ. Bất kỳ hàm mất mát nào có bao gồm một hàm đối của logarit hợp lý đều là một entropy chéo giữa phân phối thực nghiệm xác định bởi tập huấn luyện, và phân phối xác

suất xác định bởi mô hình. Ví dụ, sai số bình phương trung bình là entropy chéo giữa phân phối thực nghiệm và một mô hình Gauss.

Do đó, ta có thể coi ước lượng hợp lý cực đại là một nỗ lực khiến cho phân phối của mô hình khớp với phân phối thực nghiệm $\hat{p}_{\text{data}}$, và lý tưởng nhất là khớp với phân phối sinh dữ liệu thực sự p_{data}, nhưng trong thực tế ta không thể trực tiếp tiếp cập phân phối này.

Trong khi giá trị $\boldsymbol{\theta}$ tối ưu là giống nhau bất kể việc ta sẽ cực đại hoá hàm hợp lý hay cực tiểu hoá hóa độ phân kỳ KL, thì giá trị của các hàm mục tiêu là khác nhau. Ta thường coi cả hai là những quá trình tối thiểu hóa hàm chi phí. Do đó, cực đại hóa hàm hợp lý trở thành cực tiểu hóa hàm đối của logarit hợp lý (NLL - negative log likelihood), hay tương đương cực tiểu hóa entropy chéo. Quan điểm coi hợp lý cực đại như là phân kỳ KL cực tiểu trở nên hữu ích trong trường hợp này bởi vì ta đã biết phân kỳ KL có giá trị nhỏ nhất là 0. Hàm đối của logarit hợp lý trong thực tế có thể trở thành số âm khi $\boldsymbol{x}$ là giá trị thực.

5.5.1 Logarit hợp lý có điều kiện và sai số bình phương trung bình

Ta có thể dễ dàng tổng quát hóa *ước lượng hợp lý cực đại* (maximum likelihood estimator) để ước lượng một xác suất có điều kiện $P(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ để dự đoán $\mathbf{y}$ khi đã biết $\mathbf{x}$. Trong thực tế, bước biến đổi này được dùng rất phổ biến vì nó là cơ sở cho hầu hết các phương pháp học có giám sát. Nếu $\boldsymbol{X}$ biểu diễn tất cả các đầu vào và $\boldsymbol{Y}$ là tất cả các nhãn mà ta quan sát được, thì ước lượng hợp lý cực đại có điều kiện là

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \qquad (5.62)$$

Nếu các mẫu của ta là i.i.d, biểu thức trên có thể tách ra thành:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log\ P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \qquad (5.63)$$

Ví dụ: Hồi quy tuyến tính dưới góc nhìn hợp lý cực đại. Hồi quy tuyến tính, được giới thiệu ở phần 5.1.4, có thể xem là một quá trình hợp lý cực đại. Như đã nói, ta sử dụng hồi quy tuyến tính như là một thuật toán để học cách lấy dữ liệu đầu vào $\boldsymbol{x}$, từ đó sinh một đầu ra $\hat{y}$. Phép ánh xạ từ $\boldsymbol{x}$ tới $\hat{y}$ được chọn để cực tiểu hoá sai số bình phương trung bình, một tiêu chuẩn mà ta đã đưa ra ít nhiều

mang tính tùy ý. Bây giờ ta sẽ xem xét lại mô hình hồi quy tuyến tính từ góc nhìn của ước lượng hợp lý cực đại. Thay vì sinh ra một nhãn dự đoán $\hat{y}$, ta coi mô hình trên như là một cách tạo ra phân phối có điều kiện $p(y \mid \boldsymbol{x})$. Hình dung rằng với một tập huấn luyện lớn vô hạn, ta có thể quan sát thấy vài mẫu dữ liệu huấn luyện có giá trị đầu vào $\boldsymbol{x}$ giống nhau nhưng lại cho ra các giá trị y khác nhau. Mục tiêu của phương pháp học giờ đây là làm cho phân phối $p(y \mid \boldsymbol{x})$ khớp với tất cả các giá trị y khác nhau trên sao cho chúng đều tương thích với $\boldsymbol{x}$. Để suy ra thuật toán hồi quy tuyến tính ban đầu, ta định nghĩa $p(y \mid \boldsymbol{x}) = \mathcal{N}(y\,;\,\hat{y}(\boldsymbol{x},\,\boldsymbol{w}), \sigma^2)$. Với hàm $\hat{y}(\boldsymbol{x}; \boldsymbol{w})$ là một dự đoán của giá trị trung bình của phân phối chuẩn. Trong ví dụ này, ta giả sử rằng phương sai được cố định là một vài hằng số σ^2 đã chọn từ trước. Ta sẽ thấy rằng cách chọn dạng của hàm $p(y \mid \boldsymbol{x})$ khiến cho quá trình ước lượng hợp lý cực đại tương đương với thuật toán học tập ta đã xây dựng trước đó. Vì các mẫu dữ liệu được giả định là i.i.d., hàm logarit hợp lý có điều kiện (phương trình (5.63)) sẽ trở thành:

$$\sum_{i=1}^{m} \log p(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \qquad (5.64)$$

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^{m} \frac{||\widehat{y}^{(i)} - y^{(i)}||^2}{2\sigma^2} \qquad (5.65)$$

với $\hat{y}^{(i)}$ là đầu ra của hồi quy tuyến tính ứng với đầu vào $\boldsymbol{x}^{(i)}$ và m là số lượng mẫu huấn luyện. So sánh logarit hợp lý với sai số bình phương trung bình,

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^{m} ||\widehat{y}^{(i)} - y^{(i)}||^2 \qquad (5.66)$$

ta có thể thấy ngay rằng việc cực đại hoá giá trị logarit hàm hợp lý theo tham số $\boldsymbol{w}$ cho ra ước lượng của tham số $\boldsymbol{w}$ giống như khi cực tiểu hoá sai số bình phương trung bình. Hai tiêu chuẩn có các giá trị khác nhau nhưng lại có cùng nghiệm tối ưu. Điều này giải thích lí do ta có thể sử dụng MSE như một quá trình ước lượng hợp lý cực đại. Ta sẽ thấy ước lượng sử dụng hợp lý cực đại có nhiều tính chất hữu dụng.

5.5.2 Các tính chất của phương pháp hợp lý cực đại

Ưu điểm chính của ước lượng sử dụng hợp lý cực đại là nó tiệm cận với ước lượng tốt nhất, khi số mẫu $m \to \infty$, về mặt tốc độ hội tụ khi m tăng lên.

Với các điều kiện phù hợp, ước lượng sử dụng hợp lý cực đại có tính nhất quán (xem mục 5.4.5 ở trên), có nghĩa là khi số mẫu huấn luyện tiến tới vô cùng, ước lượng hợp lý cực đại của tham số sẽ hội tụ đến giá trị đúng của tham số. Các điều kiện cần thiết là:

- Phân phối thực sự p_{data} phải nằm ở trong họ mô hình $p_{\text{model}}(.;\boldsymbol{\theta})$. Nếu không, không ước lượng nào có thể xác định được p_{data}
- Phân phối thực sự p_{data} phải tương ứng chính xác với đúng một giá trị của $\boldsymbol{\theta}$. Nếu không, ước lượng hợp lý cực đại có thể tìm được p_{data}, nhưng sẽ không thể xác định giá trị $\boldsymbol{\theta}$ được dùng cho quá trình sinh dữ liệu.

Vẫn còn những phương pháp quy nạp khác bên cạnh ước lượng hợp lý cực đại, nhiều phương pháp trong số đó cũng thoả mãn tính chất của ước lượng nhất quán. Tuy nhiên, các ước lượng nhất quán khác nhau có thể có *tính hiệu quả thống kê* (statistical efficiency) khác nhau, có nghĩa là một ước lượng nhất quán có thể đạt được sai số tổng quát hoá thấp hơn với một số lượng mẫu m cố định, nói cách khác, ước lượng đó có thể chỉ cần một số lượng mẫu ít hơn để đạt được một mức sai số tổng quát hoá cố định.

Tính hiệu quả thống kê thường được nghiên cứu trong *trường hợp có tham số* (parametric case) (chẳng hạn như trong hồi quy tuyến tính), ở đó mục đích của ta là ước lượng giá trị của tham số (với giả định rằng giá trị thực của tham số là có thể xác định), chứ không phải giá trị của hàm. Một cách để đo lường sự tương quan giữa ước lượng của ta với với giá trị thực của tham số là tính kỳ vọng của sai số bình phương trung bình, tính bình phương của hiệu giữa tham số ước lượng và giá trị thực của tham số, với kỳ vọng lấy trên m mẫu huấn luyện từ phân phối sinh dữ liệu. Sai số bình phương trung bình này giảm khi m tăng, và với m lớn, cận dưới Cramér-Rao [Rao, 1945; Cramér, 1946] chỉ ra rằng không có ước lượng nhất quán nào có sai số bình phương trung bình thấp hơn ước lượng hợp lý cực đại.

Vì những lý do trên (tính nhất quán và tính hiệu quả), hợp lý cực đại thường được lựa chọn cho ước lượng trong ML. Khi số lượng mẫu đủ nhỏ để dẫn tới hiện tượng overfitting, các phương pháp kiểm soát như suy giảm trọng số có thể được dùng để thu được một phiên bản hiệu chỉnh của hợp lý cực đai có phương sai nhỏ hơn khi tập dữ liệu huấn luyện có giới hạn.

5.6 Thống kê Bayes

Cho đến nay, ta đã thảo luận về *thống kê tần suất* (frequentist statistics) và các phương pháp tiếp cận dựa trên việc ước lượng một giá trị đơn lẻ của $\boldsymbol{\theta}$, sau đó thực hiện tất cả các dự đoán tiếp theo dựa trên chỉ một ước lượng đó. Một cách tiếp cận khác là xem xét tất cả các giá trị có thể nhận của $\boldsymbol{\theta}$ khi đưa ra dự đoán. Ý tưởng đó chính là nội dung của *thống kê Bayes* (Bayesian statistics).

Như đã thảo luận trong phần 5.4.1, quan điểm của trường phái tần suất là: giá trị thực của tham số $\boldsymbol{\theta}$ là một số cố định nhưng chưa được biết trước, trong khi ước lượng điểm $\hat{\boldsymbol{\theta}}$ là một biến ngẫu nhiên bởi nó là hàm của một tập dữ liệu ngẫu nhiên.

Quan điểm Bayes về thống kê là khá khác biệt. Thống kê Bayes sử dụng xác suất để phán ánh mức độ chắc chắn trong các trạng thái của tri thức. Tập dữ liệu là những quan sát trực tiếp và do đó nó không ngẫu nhiên. Mặt khác, giá trị thực $\boldsymbol{\theta}$ của tham số là chưa biết hoặc bất định và vì vậy nó được biểu diễn như một biến ngẫu nhiên.

Trước khi quan sát dữ liệu, ta biểu diễn các tri thức về $\boldsymbol{\theta}$ bằng cách sử dụng *phân phối xác suất tiên nghiệm* (prior probability distribution) $p(\boldsymbol{\theta})$ (đôi khi được gọi đơn giản là "tiên nghiệm" - the prior). Nói chung, người thực hành ML thường chọn một phân phối tiên nghiệm khá rộng (tức là, có entropy cao) để phản ánh mức độ bất định cao của giá trị $\boldsymbol{\theta}$ trước khi quan sát bất kỳ dữ liệu nào. Ví dụ, người ta có thể giả định trước rằng $\boldsymbol{\theta}$ nằm trong một phạm vi hoặc một vùng không gian hữu hạn và được phân phối đều. Trong khi nhiều tiên nghiệm khác lại ưu tiên các giải pháp "đơn giản" hơn (chẳng hạn như các hệ số có độ lớn nhỏ hơn, hoặc một hàm gần như là một hằng số).

Xét một tập dữ liệu mẫu $\{x^{(1)}, ..., x^{m}\}$. Ta có thể xác định tác động của dữ liệu đối với dự đoán của ta về $\boldsymbol{\theta}$ bằng cách kết hợp các hàm hợp lý của dữ liệu $p(x^{(1)}, ...x^{(m)} \mid \boldsymbol{\theta})$ với tiên nghiệm của ta theo quy tắc Bayes:

$$p(\boldsymbol{\theta} \mid x^{(1)}, ...x^{(m)}) = \frac{p(x^{(1)}, ..., x^{(m)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(x^{(1)}, ..., x^{(m)})} \qquad (5.67)$$

Trong các tình huống mà ước lượng Bayes thường được sử dụng, tiên nghiệm thường là một phân phối tương đối đồng đều hoặc phân phối Gauss với entropy cao, và các dữ liệu được quan sát thường gây ra việc phân phối hậu nghiệm bị

mất mát entropy và tập trung xung quanh một số ít giá trị tham số có khả năng cao xảy ra.

Liên quan đến ước lượng hợp lý cực đại, ước lượng sử dụng Bayes mang lại hai điểm khác biệt quan trọng. Thứ nhất, không giống như cách tiếp cận hợp lý cực đại, đưa ra dự đoán bằng cách sử dụng một ước lượng điểm $\boldsymbol{\theta}$, phương pháp Bayes dự đoán bằng cách sử dụng phân phối đầy đủ đối với $\boldsymbol{\theta}$. Ví dụ, sau khi quan sát m mẫu, phân phối dự đoán trên mẫu dữ liệu tiếp theo, $x^{(m+1)}$, được cho bởi công thức:

$$p(x^{(m+1)}|x^{(1)}, ..., x^{(m)}) = \int p(x^{(m+1)} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid x^{(1)}, ..., x^{(m)})\, d\boldsymbol{\theta} \tag{5.68}$$

Ở đây, mỗi giá trị của $\boldsymbol{\theta}$ với mật độ xác suất dương góp phần vào việc dự đoán mẫu tiếp theo, với trọng số bằng chính mật độ hậu nghiệm. Sau khi quan sát $\{x^{(1)}, ..., x^{(m)}\}$, nếu ta vẫn còn không chắc chắn về giá trị của $\boldsymbol{\theta}$, thì sự không chắc chắn này được kết hợp trực tiếp vào bất kỳ dự đoán nào mà ta có thể thực hiện.

Trong phần 5.4, ta đã thảo luận về cách trường phái tần suất giải quyết sự bất định trong một ước lượng điểm cho trước của $\boldsymbol{\theta}$ thông qua việc đánh giá phương sai của nó. Phương sai của ước lượng là một sự đánh giá rằng ước lượng có thể thay đổi như thế nào khi ta thay đổi cách lấy mẫu của dữ liệu quan sát được. Lý giải cho câu hỏi về cách trường phái Bayes xử lý sự bất định trong ước lượng chỉ đơn giản là lấy tích phân trên ước lượng đó, việc này có xu hướng giúp ta tránh khỏi hiện tượng overfitting. Dĩ nhiên , phép lấy tích phân như vậy chỉ là một ứng dụng của các định luật xác suất, giúp phương pháp Bayes trở nên dễ diễn giải hơn, trong khi các công cụ mà trường phái tần suất sử dụng cho việc xây dựng một ước lượng lại dựa trên sự quyết định chỉ nhằm để tổng hợp mọi tri thức trong tập dữ liệu bằng một điểm ước lượng duy nhất.

Sự khác biệt quan trọng thứ hai giữa cách tiếp cận Bayes với việc ước lượng sử dụng cách tiếp cận hợp lý cực đại là do sự đóng góp của phân phối Bayes tiên nghiệm. Tiên nghiệm gây ảnh hưởng thông qua việc chuyển dịch mật độ khối xác suất về phía các vùng của không gian tham số được ưu tiên từ trước. Trong thực tế, tiên nghiệm thường được dùng để thể hiện sự ưu tiên dành cho các mô hình đơn giản hoặc trơn tru hơn. Các nhà phê bình theo hướng tiếp cận Bayes cho rằng tiên nghiệm là nguồn gốc của đánh giá chủ quan của con người gây tác động đến các dự đoán.

Các phương pháp Bayes thường tổng quát hoá tốt hơn đối với dữ liệu huấn luyện hữu hạn nhưng lại thường chịu chi phí tính toán cao khi số lượng mẫu huấn luyện là lớn.

Ví dụ: Hồi quy tuyến tính theo hướng Bayes Ở đây, ta xét phương pháp ước lượng Bayes để học các tham số của mô hình hồi quy tuyến tính. Trong hồi quy tuyến tính, ta cần học một ánh xạ tuyến tính từ một vector đầu vào $\boldsymbol{x} \in \mathbb{R}^n$ để dự đoán giá trị của một đại lượng vô hướng $y \in \mathbb{R}$. Dự đoán này được tham số hoá bởi vector $\boldsymbol{w} \in \mathbb{R}^n$:

$$\hat{y} = \boldsymbol{w}^\intercal \boldsymbol{x}. \qquad (5.69)$$

Cho trước một tập m mẫu huấn luyện $(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$, ta có thể biểu diễn dự đoán của y trên toàn tập huấn luyện dưới dạng:

$$\hat{\boldsymbol{y}}^{(\text{train})} = \boldsymbol{X}^{(\text{train})} \boldsymbol{w} \qquad (5.70)$$

Biểu diễn dưới dạng một phân phối Gauss có điều kiện đối với $\boldsymbol{y}^{(\text{train})}$, ta được:

$$p(\boldsymbol{y}^{(\text{train})} \mid \boldsymbol{X}^{(\text{train})}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}^{(\text{train})}; \boldsymbol{X}^{(\text{train})} \boldsymbol{w},\ \boldsymbol{I}) \qquad (5.71)$$

$$\propto \exp(\frac{-1}{2}(\boldsymbol{y}^{(\text{train})} - \boldsymbol{X}^{(\text{train})} \boldsymbol{w})^\intercal (\boldsymbol{y}^{(\text{train})} - \boldsymbol{X}^{(\text{train})} \boldsymbol{w})), \qquad (5.72)$$

trong đó, ta dựa trên công thức chuẩn của MSE với giả sử rằng phương sai của phân phối Gauss trên biến y là 1. Từ nay về sau, để kí hiệu đơn giản hơn, ta viết $(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$ đơn giản là $(\boldsymbol{X}, \boldsymbol{y})$.

Để xác định phân phối hậu nghiệm dựa trên vector tham số mô hình $\boldsymbol{w}$, trước tiên ta cần xác định được phân phối tiên nghiệm. Tiên nghiệm sẽ thể hiện dự đoán "ngây thơ" của ta về giá trị của các tham số, mặc dù đôi khi việc dự đoán tiên nghiệm về các tham số của mô hình là rất khó hoặc không có cơ sở. Trong thực tế, ta thường giả định bằng một phân phối khá rộng, thể hiện mức độ bất định cao đối với $\boldsymbol{\theta}$. Riêng với các tham số có giá trị thực, ta thường sử dụng một phân phối Gauss như một phân phối tiên nghiệm:

$$p(\boldsymbol{w}) = N(\boldsymbol{w};\ \boldsymbol{\mu}_0,\ \boldsymbol{\Lambda}_0) \propto \exp(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_0)^\intercal \boldsymbol{\Lambda}_0^{-1} (\boldsymbol{w} - \boldsymbol{\mu}_0)) \qquad (5.73)$$

với $\boldsymbol{\mu}_0$ và $\boldsymbol{\Lambda}_0$ lần lượt là vector kì vọng và hiệp phương sai của phân phối tiên nghiệm.[1]

> [1] Trừ những trường hợp đặc biệt, chúng tôi giả định ma trận hiệp phương sai $\boldsymbol{\Lambda}_0 = \text{diag}(\boldsymbol{\lambda}_0)$

Sau khi đã có chỉ định phân phối tiên nghiệm, ta có thể tiếp tục xác định phân phối *hậu nghiệm* (posterior) đối với các tham số mô hình:

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w}) \tag{5.74}$$

$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^\intercal(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})\right)\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_0)^\intercal\boldsymbol{\Lambda}_0^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_0)\right) \tag{5.75}$$

$$\propto \exp\left(-\frac{1}{2}(-2\boldsymbol{y}^\intercal\boldsymbol{X}\boldsymbol{w}+\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}+\boldsymbol{w}^\intercal\boldsymbol{\Lambda}_0^{-1}\boldsymbol{w}-2\boldsymbol{\mu}_0^\intercal\boldsymbol{\Lambda}_0^{-1}\boldsymbol{w})\right) \tag{5.76}$$

Đặt $\boldsymbol{\Lambda}_m = (\boldsymbol{X}^\intercal\boldsymbol{X}+\boldsymbol{\Lambda}_0^{-1})^{-1}$ và $\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\boldsymbol{X}^\intercal\boldsymbol{y}+\boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)$. Lúc này hậu nghiệm có thể viết lại dưới dạng một phân phối Gauss:

$$p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_m)^\intercal\boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_m)+\frac{1}{2}\boldsymbol{\mu}_m^\intercal\boldsymbol{\Lambda}_m^{-1}\boldsymbol{\mu}_m\right) \tag{5.77}$$

$$\propto \exp(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_m)^\intercal\boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_m) \tag{5.78}$$

Có thể bỏ qua tất cả các thành phần không chứa $\boldsymbol{w}$ bởi phân phối cần được chuẩn hóa để có tích phân bằng 1. Cách chuẩn hoá một phân phối Gauss đa biến đã được thể hiện tại phương trình (3.23).

Biểu thức hậu nghiệm đã gợi cho ta thấy đôi điều về hiệu quả của suy luận Bayes. Trong hầu hết trường hợp, $\boldsymbol{\mu}_0$ được đặt bằng $\mathbf{0}$. Nếu đặt $\boldsymbol{\Lambda}_0 = \frac{1}{\alpha}\boldsymbol{I}$, $\boldsymbol{\mu}_m$ sẽ cho cùng một ước lượng thu được của $\boldsymbol{w}$ khi ta áp dụng phương pháp hồi quy tuyến tính theo tần suất với mức phạt suy giảm trọng số $\alpha\boldsymbol{w}^\intercal\boldsymbol{w}$. Một điều khác biệt đó là ước lượng Bayes là vô định khi α được bằng không -- nghĩa là ta không được phép bắt đầu học suy luận Bayes với phân phối tiên nghiệm đối với $\boldsymbol{w}$ có độ rộng vô hạn. Một khác biệt quan trọng hơn nữa đó là ước lượng Bayes còn cho ta một ma trận hiệp phương sai, thể hiện toàn bộ các khả năng giá trị $\boldsymbol{w}$ có thể nhận, chứ không đơn thuần chỉ đưa ra ước tính của $\boldsymbol{\mu}_m$.

5.6.1 Ước lượng hậu nghiệm cực đại (MAP)

Trong khi phần lớn các cách tiếp cận cơ bản là đưa ra dự đoán bằng cách sử dụng một phân phối hậu nghiệm Bayes đầy đủ với tham số $\boldsymbol{\theta}$, nhưng nhiều khi ta vẫn cần một ước lượng điểm tối ưu duy nhất. Lý do chính cho điều này là các phép toán liên quan đến hậu nghiệm Bayes đối với các mô hình thú vị nhất lại có chi phí tính toán vô cùng lớn, và một ước lượng điểm cho phép ta xấp xỉ với chi phí chấp nhận được. Để thu được ước lượng điểm như vậy, thay vì chỉ đưa ra một ước lượng hợp lý cực đại, ta có thể tận dụng phương pháp Bayes bằng cách dùng phân phối tiên nghiệm tác động đến việc lựa chọn điểm ước lượng. Một

trong những cách thức phù hợp để thực hiện điều này là chọn ước lượng điểm có *hậu nghiệm cực đại* (maximum a posteriori - MAP). Ước lượng hậu nghiệm cực đại sẽ tìm ra một điểm mà tại đó xác suất hậu nghiệm đạt cực đại (hay là mật độ xác suất cực đại đối với trường hợp $\boldsymbol{\theta}$ liên tục):

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \tag{5.79}$$

Có thể thấy, ở vế phải của công thức, $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ chính là logarit của hàm hợp lý còn $\log p(\boldsymbol{\theta})$ tương ứng với phân phối tiên nghiệm.

Xét ví dụ một mô hình hồi quy tuyến tính với tiên nghiệm là một phân phối chuẩn của trọng số $\boldsymbol{w}$. Nếu tiên nghiệm này có dạng $\mathcal{N}(\boldsymbol{w}; \mathbf{0}, \frac{1}{\lambda})\boldsymbol{I}^2$, thì số hạng logarit của tiên nghiệm trong (5.79) sẽ trở thành đại lượng phạt suy giảm trọng số quen thuộc $\lambda \boldsymbol{w}^\top \boldsymbol{w}$, cộng thêm một đại lượng không phụ thuộc vào $\boldsymbol{w}$ và đại lượng này sẽ không ảnh hưởng đến quá trình huấn luyện. Do đó, suy diễn MAP bằng phương pháp Bayes với trọng số tuân theo tiên nghiệm có phân phối chuẩn sẽ tương đương với phương pháp sử dụng suy giảm trọng số.

Tương tự với suy diễn Bayes đầy đủ, ưu điểm của suy diễn MAP Bayes là nó có thể tận dụng được những thông tin mà tiên nghiệm mang lại, đây là những thông tin không thể tìm thấy trong tập huấn luyện. Những thông tin bổ sung này giúp giảm phương sai của ước lượng điểm MAP (so với ước lượng hợp lý cực đại). Tuy nhiên, cái giá phải trả là độ chệch (bias) sẽ tăng lên.

Ngoài ra, có nhiều phương pháp ước lượng sử dụng cơ chế kiểm soát, chẳng hạn như ước lượng hợp lý cực đại sử dụng suy giảm trọng số, có thể được diễn giải dưới dạng xấp xỉ MAP trong suy diễn Bayes. Quan điểm này có thể áp dụng khi cơ chế kiểm soát bao gồm việc cộng thêm vào hàm mục tiêu một đại lượng tương ứng với $\log(\boldsymbol{\theta})$. Tuy nhiên, không phải mọi mức phạt của cơ chế kiểm soát đều tương đương với suy diễn MAP Bayes. Ví dụ, một số đại lượng kiểm soát có thể không phải là logarit của phân phối xác suất. Một số đại lượng kiểm soát khác lại phụ thuộc vào dữ liệu, rõ ràng phân phối xác suất tiên nghiệm không thể thoả mãn điều này.

Suy diễn MAP Bayes cung cấp một phương pháp đơn giản để thiết kế những đại lượng kiểm soát phức tạp, nhưng có thể diễn giải được. Ví dụ, một mức phạt phức tạp hơn có thể được suy ra từ tiên nghiệm là hỗn hợp các phân phối chuẩn thay vì chỉ là một phân phối chuẩn duy nhất [Nowlan và Hilton, 1992].

5.7 Các thuật toán học có giám sát

Như đã được thảo luận trong phần 5.1.3, các thuật toán học có giám sát, nói chung, là các thuật toán học được mối liên kết giữa một đầu vào và một đầu ra, khi biết một tập huấn luyện bao gồm tập các mẫu dữ liệu đầu vào $\boldsymbol{x}$ và đầu ra tương ứng $\boldsymbol{y}$. Trong nhiều trường hợp, việc tự động thu thập đầu ra $\boldsymbol{y}$ là rất khó khăn và buộc phải được cung cấp bởi một người "giám sát", tuy vậy thuật ngữ "học có giám sát" vẫn được sử dụng kể cả trong trường hợp nhãn huấn luyện được thu thập hoàn toàn tự động.

5.7.1 Học có giám sát dựa trên xác suất

Phần lớn các thuật toán học có giám sát được giới thiệu trong quyển sách này dựa trên việc ước lượng một phân phối xác suất $p(y|\boldsymbol{x})$ ta có thể làm việc này đơn giản bằng cách sử dụng ước lượng hợp lý cực đại để tìm ra vector tham số tốt nhất $\boldsymbol{\theta}$ cho họ phân phối có tham số $p(y|\boldsymbol{x};\boldsymbol{\theta})$.

Ta đã biết rằng hồi quy tuyến tính cũng thuộc họ:

$$p(y \mid \boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y;\boldsymbol{\theta}^\intercal \boldsymbol{x}, \boldsymbol{I}). \qquad (5.80)$$

Ta có thể tổng quát hoá hồi quy tuyến tính để phục vụ bài toán phân loại bằng cách định nghĩa một họ các phân phối xác suất khác. Nếu chỉ có hai lớp 0 và 1, ta chỉ cần chỉ định xác suất của một trong hai lớp. Xác suất của lớp 1 xác định xác suất của lớp 0, bởi vì tổng của hai giá trị này luôn luôn là 1.

Phân phối chuẩn đối với các biến thực mà ta đã dùng cho hồi quy tuyến tính được tham số hoá theo một đại lượng trung bình. Ta có thể gán bất kỳ giá trị nào cho đại lượng trung bình này. Xây dựng một phân phối đối với các biến nhị phân thì không đơn giản như vậy, vấn đề phát sinh là trung bình của chúng luôn phải nằm giữa 0 và 1. Một cách giải quyết vấn đề này đó là sử dụng hàm sigmoid để ép đầu ra của hàm tuyến tính thành nằm trong khoảng $(0, 1)$ và coi giá trị này như một xác suất:

$$p(y = 1 \mid \boldsymbol{x};\boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \qquad (5.81)$$

Cách tiếp cận này được gọi là *hồi quy logistic* (logistic regression).

Trong trường hợp hồi quy tuyến tính, ta có thể tìm được trọng số tối ưu bằng cách giải các phương trình chuẩn. Với hồi quy logistic, mọi việc khó khăn hơn

một chút. Không có một lời giải dạng đóng nào cho trọng số tối ưu. Thay vào đó, ta phải tìm nó bằng cách tối đa hoá logarit của hàm hợp lý. Ta có thể tìm các trọng số này bằng cách sử dụng trượt gradient để tối thiểu hoá đối số của logarit hàm hợp lý.

Về cơ bản, mọi bài toán học có giám sát đều có thể giải quyết bởi cùng một chiến lược như trên, bằng cách viết ra họ tham số trong phân phối xác suất có điều kiện đối với biến đầu vào và đầu ra phù hợp.

5.7.2 Support Vector Machine

Một trong những phương pháp có ảnh hưởng nhất trong học có giám sát là *máy vector hỗ trợ* (SVM) [Boser et al., 1992; Cortes and Vapnik, 1995]. Giống như hồi quy logistic, mô hình này cũng được định hướng bởi một hàm tuyến tính $\boldsymbol{w}^\intercal \boldsymbol{x} + b$. Nhưng khác với hồi quy logistic ở chỗ đầu ra của SVM không phải là xác suất, mà chỉ là định danh của lớp. SVM dự đoán rằng đó là lớp dương khi $\boldsymbol{w}^\intercal \boldsymbol{x} + b$ là dương, và là lớp âm khi $\boldsymbol{w}^\intercal \boldsymbol{x} + b$ là âm.

Một trong những phát kiến quan trọng nhất liên quan tới SVM đó là *thủ thuật chọn hàm lõi* (kernel trick). Thủ thuật chọn hàm lõi bắt nguồn từ nhận xét rằng rất nhiều các thuật toán ML có thể chỉ cần viết dưới dạng tích vô hướng giữa các mẫu dữ liệu. Ví dụ, ta có thể chứng minh rằng hàm tuyến tính được dùng trong SVM có thể viết lại như sau

$$\boldsymbol{w}^\intercal \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\intercal \boldsymbol{x}^{(i)}, \qquad (5.82)$$

với $\boldsymbol{x}^{(i)}$ là một mẫu huấn luyện và α là vector hệ số. Viết lại thuật toán học theo cách này cho phép ta thay thế $\boldsymbol{x}$ bằng đầu ra của một hàm đặc trưng $\phi(\boldsymbol{x})$, và toàn bộ phép tích vô hướng $\boldsymbol{x}^\intercal \boldsymbol{x}^{(i)}$ bằng hàm $k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}^{(i)})$ gọi là *hàm lõi* (kernel function). Toán tử $\cdot$ biểu thị một phép nhân vô hướng, tương đương với $\phi(\boldsymbol{x})^\intercal \phi(\boldsymbol{x}^{(i)})$. Trong một vài không gian đặc trưng, không phải lúc nào phép nhân vector vô hướng cũng được thực hiện theo định nghĩa. Trong không gian vô hạn chiều, ta buộc phải sử dụng một số kiểu nhân vô hướng khác, ví dụ như nhân vô hướng dựa trên phép tích phân thay vì phép cộng. Chi tiết cụ thể của các phép nhân vô hướng kiểu này nằm ngoài phạm vi của sách.

Sau khi thay thế phép tích vô hướng bằng hàm lõi, ta có thể đưa ra dự đoán bằng cách sử dụng hàm

$$f(\boldsymbol{x}) = b + \sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) \qquad (5.83)$$

Đối với biến $\boldsymbol{x}$ thì đây là một hàm phi tuyến, nhưng mối quan hệ giữa $\phi(\boldsymbol{x})$ và $f(\boldsymbol{x})$ là tuyến tính. Hàm lõi tương đương với việc tiền xử lý dữ liệu bằng hàm $\phi(\boldsymbol{x})$, sau đó học một mô hình tuyến tính trên không gian dữ liệu mới.

Thủ thuật chọn hãm lõi là một công cụ mạnh mẽ bởi hai lý do. Thứ nhất, nó cho phép ta học một mô hình phi tuyến đối với biến $\boldsymbol{x}$ sử dụng các kỹ thuật tối ưu lồi để đảm bảo thủ tục học sẽ hội tụ nhanh. Ta có thể thực hiện điều này bởi $\phi(\boldsymbol{x})$ là cố định và ta chỉ cần tối ưu $\boldsymbol{\alpha}$, tức là, thuật toán tối ưu coi hàm quyết định là tuyến tính trong một không gian khác. Thứ hai, hàm lõi k cho phép một cài đặt hiệu quả hơn đáng kể về mặt tính toán so với chỉ đơn giản là xây dựng hai vector $\phi(\boldsymbol{x})$ rồi tiến hành lấy tích vô hướng (phương pháp tuần tự).

Một số trường hợp, $\phi(\boldsymbol{x})$ có thể có số chiều vô hạn, dẫn đến việc chi phí tính toán của các phương pháp thực hiện tuần tự cũng là vô hạn. Trong nhiều trường hợp, $k(\boldsymbol{x}, \boldsymbol{x}')$ là một hàm phi tuyến và có chi phí tính toán nhỏ của $\boldsymbol{x}$, ngay cả khi $\phi(\boldsymbol{x})$ có chi phí tính toán cực lớn. Xét ví dụ một không gian đặc trưng vô hạn chiều với một hàm lõi dễ tính toán, ta xây dựng một hàm ánh xạ đặc trưng $\phi(x)$ đối với các số nguyên không âm x. Ánh xạ này trả về một vector gồm x giá trị 1 và theo sau là vô hạn giá trị 0. Hàm lõi có thể viết thành $k(x, x^{(i)}) = \min(x, x^{(i)})$, hoàn toàn tương đương với tích vô hướng tương ứng của hai vector có số chiều vô hạn.

Hàm lõi được sử dụng nhiều nhất là *lõi Gauss* (Gaussian kernel):

$$k(\boldsymbol{u}, \boldsymbol{v}) = \mathcal{N}(\boldsymbol{u} - \boldsymbol{v}; 0, \sigma^2 \boldsymbol{I}), \qquad (5.84)$$

trong đó $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ chính là mật độ của phân phối chuẩn. Hàm lõi này còn được gọi là lõi *hàm cơ sở xuyên tâm* (radial basis function - RBF), bởi vì bởi vì giá trị của nó giảm dọc theo các tia trong không gian $\boldsymbol{v}$ bắt nguồn từ $\boldsymbol{u}$. Lõi Gauss tương ứng với một phép tích vô hướng trong một không gian vô hạn chiều, tuy nhiên việc tìm ra không gian này phức tạp hơn không gian trong ví dụ về lõi min với các số nguyên đã được giới thiệu ở trên.

Có thể xem lõi Gauss như một loại *khớp bản mẫu* (template matching). Một mẫu huấn luyện $\boldsymbol{x}$ với nhãn y trở thành một bản mẫu cho lớp y. Khi một điểm dữ liệu kiểm thử $\boldsymbol{x}'$ gần với $\boldsymbol{x}$ theo khoảng cách Euclidean, lõi Gausss có một phản hồi mạnh, biểu thị rằng $\boldsymbol{x}'$ rất tương đồng với bản mẫu $\boldsymbol{x}$. Sau đó, mô hình gán một trọng số lớn cho nhãn huấn luyện y tương ứng. Sau cùng, dự đoán sẽ là kết hợp

của nhiều nhãn huấn luyện, được đánh trọng số theo độ tương tự của các mẫu huấn luyện tương ứng.

SVM không phải thuật toán duy nhất được tăng cường sức mạnh nhờ thủ thuật chọn hàm lõi. Rất nhiều mô hình tuyến tính cũng có thể được cải thiện bằng kỹ thuật này.Tất cả các thuật toán sử dụng thủ thuật chọn hàm lõi đều được gọi chung là *máy lõi* (kernel machine) hay *phương pháp lõi* (kernel method) [Williams and Rasmussen, 1996; Schölkopf et al., 1999].

Một nhược điểm lớn của các máy lõi nói chung đó là chi phí tính toán hàm quyết định là tuyến tính theo số lượng mẫu dữ liệu huấn luyện, bởi vì mẫu huấn luyện thứ i của tập huấn luyện sẽ đóng góp một lượng $\alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$ vào hàm quyết định. SVM có thể giảm thiểu điều này bằng cách học ra một vector $\boldsymbol{\alpha}$ với phần lớn phần tử có giá trị 0. Để phân loại một mẫu dữ liệu mới, ta chỉ cần tính hàm lõi cho các mẫu huấn luyện tương ứng với α_i khác 0. Các mẫu huấn luyện như vậy được gọi là các *vector hỗ trợ* (support vector).

Các máy lõi cũng có chi phí tính toán lớn khi tập dữ liệu huấn luyện lớn. Chúng tôi sẽ trở lại với vấn đề này ở mục 5.9. Các máy lõi với các hàm lõi chung gặp khó khăn trong việc khái quát hoá dữ liệu, với lý do ở sẽ được giải thích ở phần 5.11. Nguyên dạng của DL hiện đại ban đầu được thiết kế để vượt qua những nhược điểm của máy lõi như đã nêu. Sự phục hưng của DL bắt nguồn từ khi Hilton và các cộng sự (2006) chứng minh rằng mạng neuron có thể vượt hiệu suất của SVM với hàm lõi là RBF trong bài toán MNIST.

5.7.3 Các thuật toán học có giám sát đơn giản khác

Chúng tôi đã từng giới thiệu một cách ngắn gọn về một thuật toán học có giám sát khác không dựa trên xác suất, đó là hồi quy điểm gần nhất. Tổng quát hơn, k điểm gần nhất có thể được sử dụng cho cả hai bài toán: phân lớp và hồi quy. Là một thuật toán học phi tham số, thuật toán k điểm gần nhất không bị hạn chế bởi một số lượng tham số cố định. Khi nhắc đến thuật toán k điểm gần nhất, ta thường nghĩ đến nó bởi tính chất không có tham số thay vì là thuật toán thực thi một hàm đơn giản của dữ liệu huấn luyện. Trong thực tế, thuật toán này thậm chí còn không có quá trình huấn luyện hoặc một tiến trình học nào cả. Thay vào đó, ở giai đoạn kiểm thử, khi muốn tìm đầu ra y của một mẫu $\boldsymbol{x}$ mới, ta sẽ tìm k

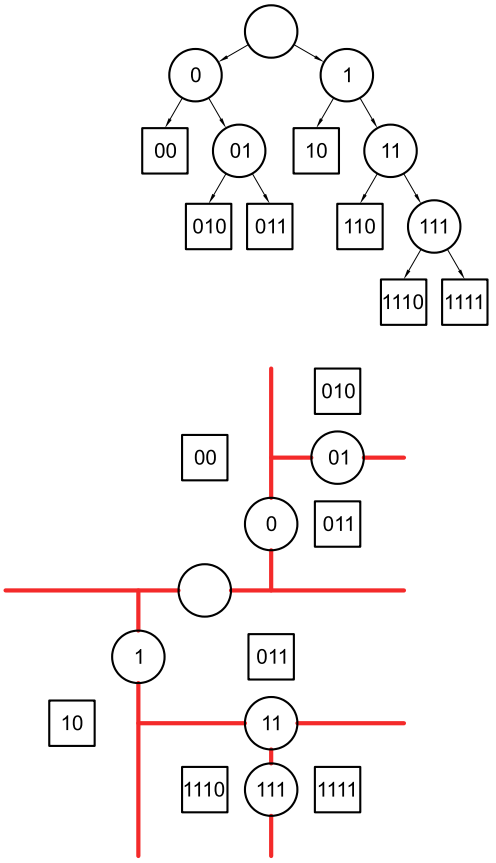
điểm gần nhất của $\boldsymbol{x}$ trong tập dữ liệu huấn luyện $\boldsymbol{X}$. Đầu ra y của thuật toán chính là giá trị trung bình của các đầu ra ứng với k điểm gần nhất vừa tìm được. Thuật toán này về cơ bản là có thể áp dụng cho bất kỳ loại bài toán học có giám sát nào mà ta xác định được giá trị trung bình của các giá trị đầu ra y. Trong bài toán phân lớp, ta có thể lấy giá trị trung bình thông qua các vector mã hóa đơn trội (one-hot code) $\boldsymbol{c}$ với $c_y = 1$ và $c_i = 0$ với tất cả các giá trị khác của i. Sau đó, ta có thể diễn giải mã hóa đơn trội như là phân phối xác suất của các lớp. Là một thuật toán học không tham số, k điểm gần nhất có thể có capacity lớn. Chẳng hạn, giả sử rằng có một bài toán phân loại đa lớp và một thang đo hiệu năng với hàm mất mát $0 - 1$. Trong thiết lập này, thuật toán 1-điểm gần nhất hội tụ tới giá trị gấp đôi sai số Bayes khi số lượng các mẫu trong tập huấn luyện tiến tới vô cùng. Phần sai số vượt quá sai số Bayes gây bởi việc chọn ngẫu nhiên duy nhất một lân cận trong trường hợp có nhiều điểm cùng là điểm gần nhất. Khi tập huấn luyện là vô hạn, mọi điểm dữ liệu kiểm thử $\boldsymbol{x}$ sẽ có một tập vô số các lân cận có khoảng cách bằng 0. Nếu ta cho phép thuật toán sử dụng tất cả những lân cận này tham gia bình chọn giá trị đầu ra của $\boldsymbol{x}$ thay vì chọn ngẫu nhiên một trong số những lân cận đó, thì thủ tục này sẽ hội tụ tới tỉ lệ sai số Bayes. Dung lượng lớn của mô hình k điểm gần nhất cho phép thuật toán đạt được độ chính xác cao với một tập huấn luyện lớn, nhưng cũng là nguyên nhân khiến nó có chi phí tính toán lớn, và khả năng tổng quát hóa có thể rất tệ khi tập huấn luyện là hữu hạn. Một điểm yếu của thuật toán k điểm gần nhất là nó không thể nhận biết được liệu có đặc trưng nào có tính phân biên rõ ràng hơn đặc trưng khác. Ví dụ, hãy tưởng tượng ta có một bài toán hồi quy với $\boldsymbol{x} \in \mathbb{R}^{100}$ lấy ngẫu nhiên từ một phân phối Gauss đẳng hướng, nhưng chỉ có một biến duy nhất x_1 có liên quan đến đầu ra. Giả sử rằng biến này trực tiếp mã hóa đầu ra, nghĩa là $y = x_1$ trong mọi trường hợp. Hồi quy điểm gần nhất không thể phát hiện được mô hình đơn giản này. Điểm gần nhất của hầu hết các điểm $\boldsymbol{x}$ sẽ được xác định bởi các đặc trưng từ x_2 tới x_{100}, chứ không phải bởi duy nhất đặc trưng x_1. Do đó, đầu ra của thuật toán đối với các tập huấn luyện nhỏ về cơ bản sẽ là ngẫu nhiên.

Một loại thuật toán học tập khác có thể chia nhỏ dữ liệu đầu vào thành các vùng và có các tham số riêng biệt cho mỗi vùng được gọi là *cây quyết định* (decision tree) [Breiman et al., 1984] và nhiều biến thể của nó. Như được mô tả trong hình 5.7, mỗi nút của cây quyết định được liên kết với một vùng trong không gian đầu vào, và các nút trong (internal node) chia nhỏ vùng đó thành một tiểu vùng cho mỗi nút con của nút trong đó (thường bằng cách sử dụng lát cắt dọc theo trục).

Do đó, không gian được phân chia thành các vùng không chồng lấn, với sự tương quan một-một giữa các nút lá và các vùng đầu vào. Mỗi nút lá thường ánh xạ mọi điểm trong vùng đầu vào của nó tới cùng một đầu ra. Cây quyết định thường được huấn luyện bằng các thuật toán chuyên biệt, nhưng phần này nằm ngoài phạm vi của cuốn sách. Thuật toán học tập này có thể được coi là không tham số nếu nó được phép học một cây với kích thước tùy ý. Nhưng trong thực tế, cây quyết định thường được kiểm soát với các ràng buộc về kích thước, biến chúng thành các mô hình có tham số. Ta thường sử dụng các cây quyết định có các lát cắt dọc trục và đầu ra không đổi bên trong mỗi nút. Các cây này gặp khó khăn trong quyết một số bài toán dễ dàng, ngay cả đối với hồi quy logistic. Ví dụ, ta có một bài toán hai lớp, và lớp dương tương ứng với trường hợp $x_2 > x_1$, ranh giới quyết định không song song với các trục. Do đó cây quyết định sẽ phải ước lượng biên quyết định thông qua nhiều nút, bằng việc cài đặt một hàm bước nhảy, đi qua đi lại hàm quyết định thực sự bằng các bước dọc theo trục.

Như ta đã thấy, các thuật toán dự đoán thông qua điểm gần nhất và cây quyết định có nhiều khuyết điểm. Tuy nhiên, chúng lại hữu ích khi tài nguyên tính toán bị hạn chế. Ta cũng có thể hiểu một cách trực quan về các thuật toán phức tạp hơn bằng những điểm giống và khác nhau giữa những thuật toán phức tạp và các thuật toán cơ sở như k điểm gần nhất hoặc cây quyết định.

Bạn đọc có thể tham khảo Murphy (2012), Bishop (2006), Hastie et al. (2001) hoặc các cuốn sách về ML khác để biết hiểu rõ hơn về các thuật toán học có giám sát truyền thống.



Hình 5.7: Lược đồ mô tả cách thức hoạt động của cây quyết định. (*Hình trên*) Mỗi nút của cây chọn đầu vào để gửi cho nút con của nó: 0 ở nút trái và 1 ở nút phải. Các nút trong được vẽ bằng hình tròn, và hình vuông đối với các nút lá. Mỗi nút được hiển thị bằng một chuỗi nhị phân tương ứng với vị trí của nó trong cây, thu được bằng việc thêm một bit vào chuỗi nhị phân định dạng của nút cha ($0 =$ cho nút trái, $1 =$ cho nút phải). (*Hình dưới*) Cây phân chia không gian thành các vùng. Mặt phẳng hai chiều này chỉ ra cách một cây quyết định có thể phân chia không gian $\mathbb{R}^2$. Các nút của cây được vẽ trong mặt phẳng, với mỗi nút trong được vẽ dọc theo đường phân chia mà nó sử dụng để phân loại các mẫu, và các nút lá được vẽ ở trung tâm của vùng các mẫu dữ liệu mà chúng nhận. Kết quả của sự phân chia trên là một hàm hằng số từng phần, mỗi phần tương ứng một lá. Mỗi lá yêu cầu ít nhất một mẫu huấn luyện để xác định, vì vậy cây quyết định không thể học một hàm mà có nhiều cực đại địa phương hơn số lượng các mẫu huấn luyện.

5.8 Các thuật toán học không giám sát

Ta đã biết trong phần 5.1.3 rằng các thuật toán không giám sát là những thuật toán chỉ dựa trên những "đặc trưng" chứ không phải một tín hiệu giám sát. Sự khác biệt giữa thuật toán có giám sát và không giám sát không được định nghĩa một cách hình thức và rõ ràng, bởi không có một phép thử khách quan nào để phân biệt liệu một giá trị là một đặc trưng hay một nhãn được cung cấp bởi giám sát viên. Nói một cách nôm na, học không giám sát đề cập đến hầu hết các nỗ lực để trích xuất thông tin từ một phân phối mà không yêu cầu nhân lực để gán nhãn các mẫu. Thuật ngữ này thường được kết hợp với vấn đề ước lượng mật độ, với mục đích học để lấy mẫu từ một phân phối, học để khử nhiễu dữ liệu sinh bởi một số phân phối nào đó, tìm kiếm một một mô phỏng gần giống với dữ liệu, hoặc phân cụm dữ liệu thành các nhóm các mẫu có liên quan với nhau.

Một bài toán học không giám sát cổ điển là xác định biểu diễn "tốt nhất" của dữ liệu. Cụm từ "tốt nhất" có thể mang nhiều nghĩa khác nhau, nhưng nói chung ta đang tìm kiếm một cách biểu diễn bảo tồn nhiều thông tin của mẫu $\boldsymbol{x}$ nhất có thể, trong khi tuân theo một số mức phạt hoặc ràng buộc nhằm đảm bảo rằng biểu diễn này đơn giản hơn hoặc dễ tiếp cận hơn chính $\boldsymbol{x}$.

Có nhiều cách để xác định một dạng biểu diễn đơn giản hơn. Ba trong số những cách thông dụng nhất bao gồm biểu diễn với số chiều nhỏ hơn, biểu diễn thưa, và biểu diễn độc lập. Phép biểu diễn với số chiều nhỏ hơn cố gắng nén càng nhiều thông tin thông tin nhất có thể của $\boldsymbol{x}$ trong một biểu diễn nhỏ hơn. Các biểu diễn thưa [Barlow, 1989; Olshausen and Field, 1996; Hinton and Ghahramani, 1997] nhúng tập dữ liệu vào một biểu diễn mà hầu hết các phần tử của nó có giá trị bằng 0. Việc sử dụng các biểu diễn thưa thường đòi hỏi việc gia tăng kích thước của biểu diễn, với mục đích không bỏ qua quá nhiều thông tin khi phần lớn các giá trị của biểu diễn có giá trị bằng 0. Điều này dẫn đến cấu trúc tổng thể của nó có xu hướng phân phối dữ liệu dọc theo các trục của không gian biểu diễn. Cuối cùng, các biểu diễn độc lập cố gắng phân tách các nguồn gây ra sự biến đổi đằng sau phân phối dữ liệu sao cho các chiều của biểu diễn là độc lập về mặt thống kê.

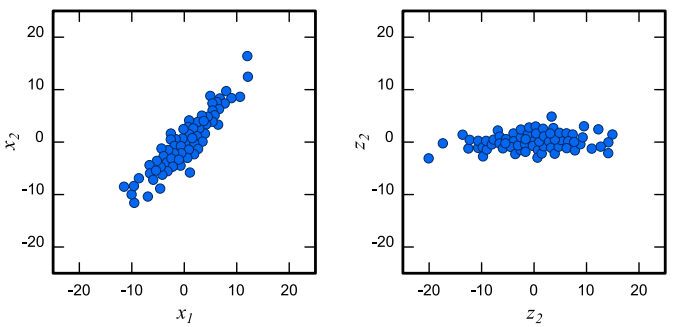
Tất nhiên, ba tiêu chí này chắc chắn không loại trừ lẫn nhau. Các biểu diễn với số chiều nhỏ hơn thường tạo ra các phần tử có sự phụ thuộc kém hoặc yếu hơn so với dữ liệu có số chiều cao ban đầu. Điều này là do việc giảm kích cỡ của biểu diễn được thực hiện bằng việc tìm và loại bỏ sự dư thừa. Xác định và loại bỏ nhiều dư thừa cho phép thuật toán giảm chiều nén dữ liệu được nhiều hơn đồng thời loại bỏ ít thông tin hơn.

Khái niệm về biểu diễn là một trong những chủ đề trung tâm của DL và do đó là một trong những chủ đề trọng tâm trong cuốn sách này. Trong phần này, chúng tôi sẽ phát triển một số ví dụ đơn giản của các thuật toán học biểu diễn. Các thuật toán ví dụ này cùng nhau cho thấy cách vận hành cả ba tiêu chí trên. Hầu hết các chương còn lại giới thiệu các thuật toán học biểu diễn khác để phát triển các tiêu chí này theo những cách khác nhau, hoặc giới thiệu các tiêu chí khác.

5.8.1 Phân tích thành phần chính

Trong phần 2.12, ta đã thấy rằng thuật toán phân tích thành phần chính (PCA) là một cách nén dữ liệu. Ta cũng có thể xem PCA như một thuật toán học không giám sát để học cách biểu diễn dữ liệu. Biểu diễn này dựa trên hai trong số các tiêu chí cho một biểu diễn đơn giản được mô tả ở trên. PCA học một biểu diễn có số chiều thấp hơn so với đầu vào ban đầu. Nó cũng học một biểu diễn mà các phần tử của chúng không có mối tương quan tuyến tính với nhau. Đây là bước đầu tiên hướng tới tiêu chí học các biểu diễn mà các thành phần của nó độc lập về mặt thống kê. Để đạt được sự độc lập hoàn toàn, một thuật toán học biểu diễn cũng phải loại bỏ các mối quan hệ phi tuyến giữa các biến.

PCA học một phép biến đổi trực giao và tuyến tính của dữ liệu để chiếu một đầu vào $\boldsymbol{x}$ thành một biểu diễn $\boldsymbol{z}$ như được trình bày trong hình 5.8. Trong phần 2.12, ta đã thấy rằng ta có thể học một biểu diễn một chiều cho phép tái tạo lại dữ liệu gốc tốt nhất (theo nghĩa sai số bình phương trung bình thấp nhất) và rằng biểu diễn này thực tế tương ứng với thành phần chính thứ nhất của dữ liệu. Do đó, ta có thể sử dụng PCA như là một phương pháp giảm chiều đơn giản và hiệu quả để giữ thông tin trong dữ liệu nhiều nhất có thể (vẫn được đo bằng bình phương sai số tái thiết nhỏ nhất). Trong phần tiếp theo, ta sẽ nghiên cứu xem biểu diễn PCA khử tính tương quan của biểu diễn dữ liệu gốc $\boldsymbol{X}$ như thế nào.



> Hình 5.8: PCA học một phép chiếu tuyến tính cho phép điều chỉnh trục của không gian mới theo hướng của phương sai lớn nhất. (*Hình trái*) Dữ liệu gốc bao gồm các mẫu dữ liệu $\boldsymbol{x}$. Trong không gian này, hướng của phương sai có thể khác hướng của trục không gian. Hình bên phải biểu diễn dữ liệu đã biến đổi $\boldsymbol{z} = \boldsymbol{x}^\top \boldsymbol{W}$, dữ liệu này biến thiên nhiều nhất dọc theo trục z_1. Hướng biến thiên nhiều thứ hai theo trục z_2.

Xét ma trận thiết kế $\boldsymbol{X}$ với kích thước $m \times n$. Giả sử rằng dữ liệu có kỳ vọng bằng 0, tức là $\mathbb{E}[\boldsymbol{X}] = 0$. Nếu dữ liệu không có kỳ vọng bằng không, ta có thể dễ dàng chuẩn hoá bằng cách trừ tất cả các phần tử của ma trận cho kỳ vọng trong bước tiền xử lý.

Ma trận hiệp phương sai mẫu không chệch ứng với $\boldsymbol{X}$ được biểu diễn bởi công thức:

$$\mathrm{Var}[\boldsymbol{x}] = \frac{1}{m-1} \boldsymbol{X}^\top \boldsymbol{X} \qquad (5.85)$$

PCA tìm một biểu diễn (thông qua biến đổi tuyến tính) $\boldsymbol{z} = \boldsymbol{W}^\top x$, trong đó $\mathrm{Var}[\boldsymbol{z}]$ là một ma trận đường chéo.

Trong mục 2.12, ta thấy rằng những thành phần chính của ma trận $\boldsymbol{X}$ được xác định bởi vector riêng của ma trận vuông $\boldsymbol{X}^\top \boldsymbol{X}$. Từ góc nhìn này, ta có

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{W} \boldsymbol{\Lambda} \boldsymbol{W}^\top \qquad (5.86)$$

Trong phần này, ta sẽ khai thác một cách khác để xác định các thành phần chính. Các thành phần chính có thể thu được qua phân tích giá trị suy biến (SVD). Cụ thể hơn, chúng chính là vector suy biến phải của $\boldsymbol{X}$. Để thấy được điều này, ta ký hiệu $\boldsymbol{W}$ là các vector suy biến phải trong phép phân tích $\boldsymbol{X} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{W}^\top$. Từ đó tìm lại được phương trình vector riêng ban đầu với $\boldsymbol{W}$ là hệ cơ sở các vector riêng:

$$\boldsymbol{X}^\top \boldsymbol{X} = (\boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{W}^\top)^\top \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{W}^\top = \boldsymbol{W} \boldsymbol{\Sigma}^2 \boldsymbol{W}^\top. \qquad (5.87)$$

SVD giúp ta thấy rằng kết quả của PCA là một ma trận đường chéo $\mathrm{Var}[z]$. Sử dụng SVD của $\boldsymbol{X}$, ta có thể diễn tả phương sai của $\boldsymbol{X}$ như sau:

$$\begin{aligned}
\mathrm{Var}[\boldsymbol{x}] &= \frac{1}{m-1}\boldsymbol{X}^\intercal\boldsymbol{X} && (5.88)\\
&= \frac{1}{m-1}(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\intercal)^\intercal\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\intercal && (5.89)\\
&= \frac{1}{m-1}\boldsymbol{W}\boldsymbol{\Sigma}^\intercal\boldsymbol{U}^\intercal\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\intercal && (5.90)\\
&= \frac{1}{m-1}\boldsymbol{W}\boldsymbol{\Sigma}^2\boldsymbol{W}^\intercal && (5.91)
\end{aligned}$$

trong đó dấu bằng ở (5.91) xảy ra do $\boldsymbol{U}^\intercal\boldsymbol{U} = \boldsymbol{I}$ bởi ma trận $\boldsymbol{U}$ của phép phân tích giá trị suy biến, theo định nghĩa, là trực giao. Điều này cho thấy hiệp phương sai của z là một ma trận đường chéo như yêu cầu:

$$\begin{aligned}
\mathrm{Var}[\boldsymbol{z}] &= \frac{1}{m-1}\boldsymbol{Z}^\intercal\boldsymbol{Z} && (5.92)\\
&= \frac{1}{m-1}\boldsymbol{W}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{W} && (5.93)\\
&= \frac{1}{m-1}\boldsymbol{W}^\intercal\boldsymbol{W}\boldsymbol{\Sigma}^{\boldsymbol{2}}\boldsymbol{W}^\intercal\boldsymbol{W} && (5.94)\\
&= \frac{1}{m-1}\boldsymbol{\Sigma}^{\boldsymbol{2}} && (5.95)
\end{aligned}$$

với $\boldsymbol{W}^\intercal\boldsymbol{W} = \boldsymbol{I}$ theo định nghĩa của phân tích giá trị suy biến (SVD).

Phân tích trên cho thấy rằng khi ta chiếu dữ liệu $\boldsymbol{x}$ xuống $\boldsymbol{z}$ qua phép biến đổi tuyến tính với $\boldsymbol{W}$, kết quả biểu diễn có một ma trận hiệp phương sai là ma trận đường chéo (được cho bởi $\boldsymbol{\Sigma}^{\boldsymbol{2}}$), từ đó ngay lập tức ta thấy được các phần tử của $\boldsymbol{z}$ không tương quan lẫn nhau.

Khả năng biến đổi dữ liệu thành biểu diễn mới trong đó các phần tử không tương quan lẫn nhau là đặc tính cực kỳ quan trọng của PCA. Đây là một ví dụ đơn giản cho cho một phép biểu diễn có gắng *tách các yếu tố biến thiên tiềm ẩn* trong dữ liệu. Đối với PCA, cách nó thực hiện quá trình bóc tách này là tìm một phép quay của không gian dữ liệu đầu vào (được mô tả bởi $\boldsymbol{W}$) mà hướng các trục chính của phương sai theo hệ cơ sở của không gian biểu diễn mới gắn liền với $\boldsymbol{z}$.

Trong khi tương quan là một dạng phụ thuộc quan trọng giữa các phần tử trong dữ liệu, ta cũng quan tâm đến việc học cách biểu diễn có thể bóc tách các dạng phụ thuộc phức tạp hơn giữa các đặc trưng. Để làm được điều này, ta cần nhiều hơn so với những gì có thể làm được với biến đổi tuyến tính đơn giản.

5.8.2 Phân cụm k-means

Một ví dụ đơn giản khác của học biểu diễn là phân cụm k-means. Thuật toán phân cụm k-means chia tập huấn luyện thành k cụm dữ liệu khác nhau, mỗi cụm là các phần tử gần nhau. Vì vậy, ta có thể hiểu rằng thuật toán này sẽ cung cấp vector mã hóa đơn trội $\boldsymbol{h}$ có k chiều biểu diễn dữ liệu đầu vào $\boldsymbol{x}$. Nếu $\boldsymbol{x}$ thuộc cụm i thì $h_i = 1$, và tất cả các giá trị còn lại trong vector biểu diễn $\boldsymbol{h}$ là 0.

Mã hóa đơn trội thu được từ phân cụm k-means là một ví dụ của biểu diễn thưa, vì phần lớn các phần tử có giá trị 0 với mỗi đầu vào. Sau này, ta sẽ phát triển các thuật toán khác có thể học được biểu diễn thưa một cách linh hoạt hơn, mà ở đó có thể có nhiều hơn một phần tử có giá trị khác 0 cho mỗi đầu vào $\boldsymbol{x}$. Mã hóa đơn trội là trường hợp cực đoan của biểu diễn thưa, bởi nó làm mất rất nhiều lợi ích của một phép biểu diễn phân tán. Tuy nhiên, mã hóa đơn trội vẫn thể hiện một số lợi thế về mặt thống kê (nó truyền đạt một cách tự nhiên ý tưởng rằng các phần tử trong cùng một cụm là giống nhau), và nó mang lại lợi thế về tính toán vì toàn bộ biểu diễn được thể hiện bởi một số nguyên duy nhất.

Thuật toán phân cụm k-means bắt đầu bằng việc khởi tạo k trọng tâm khác nhau $\{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(k)}\}$ ứng với những giá trị khác nhau, sau đó lặp đi lặp lại hai bước khác nhau cho đến khi hội tụ. Ở bước đầu tiên, mỗi mẫu huấn luyện được gắn với cụm i, trong đó i là chỉ số của trọng tâm gần nhất $\boldsymbol{\mu}^{(i)}$. Ở bước còn lại, mỗi trọng tâm $\boldsymbol{\mu}^{(i)}$ được cập nhật bằng giá trị trung bình của tất cả mẫu huấn luyện $\boldsymbol{x}^{(j)}$ được phân vào cụm i.

Một khó khăn gắn liền với các bài toán phân cụm là vấn đề mập mờ, theo nghĩa là không có một tiêu chuẩn đơn lẻ nào có thể đánh giá một cách phân cụm dữ liệu thể hiện trong thực tế là tốt hay không tốt. Ta có thể đánh giá các đặc tính của phân cụm, ví dụ như khoảng cách Euclid trung bình từ trọng tâm của một cụm đến các phần tử còn lại trong cụm đó. Nó thể hiện khả năng tái tạo lại dữ liệu huấn luyện từ các phép phân cụm. Tuy nhiên, ta vẫn không biết các phép gán tương thích với các tính chất mong muốn trong thực tế như thế nào. Ngoài ra, có thể có nhiều cách phân cụm khác nhau và tất cả chúng đều tương thích tốt với một tính chất nào đó trong thực tế. Ta có thể tìm một cách phân cụm liên quan đến một đặc trưng nhưng lại thu được một phân cụm khác, có kết quả tương đương nhưng không liên quan đến tác vụ của ta. Ví dụ, giả sử ta chạy hai thuật phân cụm để tìm hai cụm trên dữ liệu gồm hình ảnh của xe tải đỏ, xe hơi

đỏ, xe tải xám và xe hơi xám. Nếu ta yêu cầu mỗi thuật tìm hai cụm, một thuật có thể phân thành hai cụm gồm một cụm xe hơi và một cụm xe tải, thuật toán còn lại thì phân thành hai cụm gồm một cụm phương tiện giao thông màu đỏ và một cụm gồm phương tiện giao thông màu xám. Giả sử ta chạy một thuật phân cụm thứ ba có khả năng tự xác định số cụm. Thuật toán này có thể phân các mẫu dữ liễu thành bốn cụm: xe hơi đỏ, xe tải đỏ, xe hơi xám, xe tải xám. Thuật toán phân loại mới này ít nhất đã xác định thông tin về cả hai đặc tính, nhưng có thể đã mất một số thông tin về sự tương đồng. Xe hơi đỏ ở cụm khác cụm xe hơi xám cũng có ý nghĩa tương đương với việc chúng khác cụm xe tải xám. Kết quả phân cụm không cho thấy việc xe hơi đỏ giống với xe hơi xám hơn là xe tải xám. Tất cả những gì ta biết là chúng khác nhau ở cả hai đặc tính.

Những vấn đề này làm rõ một số lí do mà ta nghiêng về biểu diễn phân tán hơn biểu diễn đơn trội. Biểu diễn phân tán có thể có hai đặc tính cho mỗi loại phương tiện: một thể hiện màu sắc và một thể hiện rằng phương tiện đó là xe hơi hay xe tải. Tuy việc tìm ra phép biểu diễn phân tán tối ưu vẫn chưa thực sự rõ ràng (làm cách nào để một thuật toán học có thể xác định liệu hai đặc tính ta quan tâm là màu sắc và phân loại xe hơi-xe tải hay là nhà sản xuất hoặc tuổi của phương tiện), nhưng có nhiều đặc tính làm giảm gánh nặng của thuật toán trong việc dự đoán tính chất duy nhất ta quan tâm là gì, và cho ta khả năng đánh giá sự tương đồng giữa các đối tượng một cách chi tiết hơn bằng cách so sánh nhiều đặc tính với nhau thay vì chỉ thử xem một đặc tính có phù hợp hay không.

5.9 Stochastic Gradient Descent

Gần như mọi công cụ của DL được tiếp sức bởi một thuật toán cực kỳ quan trọng: *stochastic gradient descent* (SGD). SGD là mở rộng của thuật toán trượt gradient được giới thiệu trong mục 4.3.

Một vấn đề muôn thuở của ML đó là nó cần tập huấn luyện lớn để có thể tổng quát hoá tốt, nhưng tập huấn luyện lớn cũng đồng nghĩa với việc chi phí tính toán lớn hơn.

Hàm chi phí sử dụng bởi thuật toán ML thường là tổng của tất cả hàm mất mát của từng mẫu huấn luyện. Ví dụ, hàm đối logarit của hàm hợp lý với phân phối có điều kiện của tập huấn luyện có thể viết dưới dạng

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y \sim \hat{p}_{\text{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}), \qquad (5.96)$$

với L là hàm mất mát cho từng mẫu $L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log p(y|\boldsymbol{x}; \boldsymbol{\theta})$

Với những hàm chi phí cộng tính này, để thực hiện trượt gradient, ta cần tính:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta}) \qquad (5.97)$$

Chi phí cho quá trình này là $O(m)$. Khi tập huấn luyện lên đến hàng tỉ mẫu, chi phí quá trình cho một bước trượt gradient sẽ cực kỳ lâu và tốn kém.

Một nhận xét sắc bén của SGD đó là gradient trong phương trình (5.97) là một kỳ vọng. Giá trị kỳ vọng này có thể được tính xấp xỉ bằng cách sử dụng một tập dữ liệu nhỏ. Cụ thể hơn, ở mỗi bước của thuật toán, ta có thể lấy mẫu một *lô nhỏ* (mini-batch) $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$ gồm các mẫu được lấy ngẫu nhiên đồng đều từ tập huấn luyện. Kích thước m' của lô nhỏ thường được chọn là một số tương đối nhỏ các mẫu dữ liêu, có giá trị từ một đến vài trăm. Quan trọng hơn, m' thường được giữ nguyên ngay cả khi tập huấn luyện của m tăng lên. Ta có thể thực hiện quá trình khớp một tập huấn luyện với hàng tỉ mẫu bằng các cập nhật được tính toán trên chỉ khoảng một trăm mẫu.

Ước lượng của gradient khi đó có dạng

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \qquad (5.98)$$

sử dụng các mẫu từ lô nhỏ $\mathbb{B}$. Thuật toán SGD sau đó trượt theo gradient vừa ước lượng để "xuống đồi"

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}, \qquad (5.99)$$

trong đó ϵ là tốc độ học.

Trượt gradient nói chung thường bị đánh giá là chậm hoặc không ổn định. Trong quá khứ, việc ứng dụng trượt gradient trong các vấn đề tối ưu hóa hàm không lồi thường bị cho là điên rồ và vô ích. Ngày nay, ta biết rằng các mô hình ML được mô tả ở phần II hoạt động rất tốt khi được huấn luyện với trượt gradient. Thuật toán tối ưu có thể không đảm bảo khả năng sẽ dừng ở điểm cực tiểu cục bộ trong khoảng thời gian cho phép, nhưng việc nó thường tìm ra điểm mà ở đó hàm chi phí có giá trị rất thấp trong thời gian đủ nhanh cũng là rất hữu dụng rồi.

SGD có rất nhiều ứng dụng ngoài các vấn đề trong DL. Nó là cách chủ yếu để huấn luyện các mô hình tuyến tính trong tập dữ liệu khổng lồ. Với mô hình có kích thước không đổi, chi phí của mỗi cập nhật SGD thường không phụ thuộc vào kích thước m của tập huấn luyện. Trong thực tế, ta thường dùng mô hình lớn hơn khi kích thước của tập huấn luyện tăng lên, tuy nhiên điều này là không bắt buộc. Số lần cập nhật trước khi hội tụ thường tăng lên theo kích thước của tập huấn luyện. Tuy nhiên, khi m tiến đến vô cùng, mô hình sẽ dần hội tụ về dạng có sai số kiểm thử tốt (thấp) nhất trước khi SGD chạy hết trên từng mẫu của tập huấn luyện. Việc tăng m sẽ không làm tăng thời gian huấn luyện cần thiết để đạt được mô hình có sai số kiểm thử thấp nhất của mô hình. Từ góc nhìn này, ta có thể cho rằng chi phí để huấn luyện một mô hình với SGD là tiệm cận với $O(1)$ theo biến m.

Trước khi DL xuất hiện, phương pháp chủ yếu để học các mô hình phi tuyến tính là sử dụng thủ thuật hàm lõi kết hợp với một mô hình tuyến tính. Nhiều thuật toán học hàm lõi yêu cầu xây dựng một ma trận $m \times m$: $G_{i,j} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$. Xây dựng ma trận này có chi phí là $O(m^2)$, một điều rõ ràng là không mong muốn với những tập dữ liệu có hàng tỉ mẫu. Từ năm 2006, DL đã bắt đầu được quan tâm trong giới nghiên cứu nhờ khả năng tổng quát những mẫu mới tốt hơn những thuật toán cạnh tranh khi huấn luyện trên tập dữ liệu có kích thước trung bình với khoảng mười nghìn mẫu. Không lâu sau đó, DL tiếp tục nhận được sự quan tâm từ giới công nghiệp nhờ khả năng nâng quy mô huấn luyện mô hình phi tuyến tính trên tập dữ liệu lớn.

SGD và những cải tiến của nó sẽ được miêu tả chi tiết trong chương 8.

5.10 Xây dựng một thuật toán ML

Gần như mọi thuật toán DL đều có thể được mô tả bằng một công thức khá đơn giản: kết hợp đặc điểm của một tập dữ liệu, một hàm chi phí, một phương pháp tối ưu hóa và một mô hình.

Ví dụ, thuật toán hồi quy tuyến tính kết hợp giữa một tập dữ liệu gồm $\boldsymbol{X}$ và $\boldsymbol{y}$, hàm chi phí:

$$J(\boldsymbol{w}, b) = -\mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{data}} \log p_{\mathrm{model}}(y|\boldsymbol{x}), \qquad (5.100)$$

mô hình $p_{\mathrm{model}}(y|\boldsymbol{x}) = \mathcal{N}(y; \boldsymbol{x}^\intercal \boldsymbol{w} + b, 1)$, và trong phần lớn trường hợp, thuật toán tối ưu hóa được định nghĩa bằng cách tìm tham số mà tại đó gradient của hàm

chi phí bằng 0 thông qua các phương trình chuẩn.

Nhận thấy rằng ta có thể thay thế bất kì thành phần nào ở trên một cách gần như độc lập với các thành phần khác, từ đó ta có thể tạo ra rất nhiều loại thuật toán.

Hàm chi phí thường chứa ít nhất một đại lượng đóng vai trò giúp quá trình học thực hiện ước lượng thống kê. Hàm chi phí phổ biến nhất là hàm đối của logarit hàm hợp lý, sao cho quá trình cực tiểu hoá hàm chi phí sẽ dẫn đến kết quả là ước lượng khi sử dụng hợp lý cực đại.

Hàm chi phí có thể chứa bao gồm đại lượng khác, chẳng hạn như các số hạng kiểm soát. Ví dụ, ta có thể thêm suy giảm trọng số vào hàm chi phí của hồi quy tuyến tính để thu được

$$J(\boldsymbol{w}, b) = \lambda ||\boldsymbol{w}||_2^2 - \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{data}} \log p_{\mathrm{model}}(y|\boldsymbol{x}). \qquad (5.101)$$

Hàm này vẫn cho phép ta thu được lời giải tối ưu ở dạng công thức đóng.

Nếu ta thay đổi mô hình thành dạng phi tuyến thì hầu hết các hàm chi phí đều không còn được tối ưu ở dạng đóng. Điều này này đòi hỏi ta phải chọn một phương pháp tối ưu theo vòng lặp, chẳng hạn như trượt gradient.

Công thức chung để xây dựng thuật toán học bằng việc kết hợp các mô hình, hàm chi phí và thuật toán tối ưu có thể được sử dụng cho các phương thức học có giám sát hoặc không giám sát. Ví dụ về hồi quy tuyến tính cho thấy cách mà công thức này áp dụng cho bài toán học có giám sát. Đối với học không giám sát, ta có thể định nghĩa một tập dữ liệu chỉ chứa $\boldsymbol{X}$ và cung cấp một hàm chi phí và mô hình không giám sát phù hợp. Ví dụ, ta có thể thu được vector PCA đầu tiên bằng cách chỉ rõ hàm mất mát là

$$J(\boldsymbol{w}) = \mathbb{E}_{\mathrm{x}\sim\hat{p}_{\mathrm{data}}} ||\boldsymbol{x} - r(\boldsymbol{x}; \boldsymbol{w})||_2^2 \qquad (5.102)$$

trong khi mô hình được định nghĩa để thu được $\boldsymbol{w}$ có chuẩn bằng 1 và hàm khôi phục $r(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} \boldsymbol{w}$.

Trong một số trường hợp, hàm chi phí có thể là một hàm mà về mặt tính toán ta không thể định trị được. Khi đó, ta vẫn có thể tối thiểu hóa nó một cách gần đúng bằng phương pháp tối ưu theo vòng lặp chừng nào ta còn có thể xấp xỉ các gradient của hàm.

Hầu hết các thuật toán ML đều có thể sử dụng công thức ở trên, mặc dù không phải trường hợp nào công thức này cũng rõ ràng. Với một thuật toán ML có vẻ rất

độc đáo hoặc do ai đó tự thiết kế, ta vẫn thường có thể coi như nó được tạo ra bằng một phép tối ưu đặc biệt. Một số mô hình như cây quyết định hoặc k-means cần phải có phép tối ưu đặc biệt vì hàm chi phí của chúng có các vùng phẳng, vốn không phù hợp để cực tiểu hóa bằng các phép tối ưu dựa trên gradient. Việc đưa hầu hết các thuật toán ML về công thức chung này giúp ta nhìn nhận các thuật toán khác nhau như một phần của sự phân loại các phương pháp thực hiện những tác vụ có liên quan, hơn là một danh sách dài các thuật toán mà mỗi trong số đó lại có các cách giải thích cách thức hoạt động riêng biệt.

5.11 Những thách thức thúc đẩy sự phát triển của DL

Các thuật toán ML đơn giản được mô tả ở chương này hoạt động tốt cho một loạt các bài toán quan trọng. Dù vậy, chúng vẫn chưa giải quyết được các vấn đề trọng tâm của AI, chẳng hạn là nhận biết giọng nói hay nhận dạng vật thể.

Sự phát triển của DL có được thúc đẩy một phần từ những thất bại của các thuật toán truyền thống trong việc đạt được sự tổng quát hóa tốt trong các tác vụ AI.

Phần này thảo luận về việc những thách thức trong việc tổng quát hóa các mẫu dữ liệu mới sẽ trở nên khó khăn gấp bội như thế nào khi làm việc với dữ liệu nhiều chiều, và việc các cơ chế được sử dụng để tổng quát hóa trong ML truyền thống là chưa đủ để học các hàm phức tạp trong không gian đa chiều. Những không gian này yêu cầu khối lượng tính toán khá lớn, và DL được thiết kế với mục đích chính là để giải quyết những thách thức như vậy.

5.11.1 Hiểm họa số chiều lớn

Nhiều bài toán về ML sẽ trở nên cực kì khó khăn khi dữ liệu có nhiều chiều. Hiện tượng này được gọi là *hiểm họa số chiều lớn* (the curse of dimensionality). Mối quan tâm cụ thể ở đây là số lượng cấu hình khác nhau có thể có của một tập hợp các biến sẽ tăng theo cấp số mũ khi số lượng các biến tăng lên.

Hiểm họa số chiều lớn xảy ra ở rất nhiều lĩnh vực trong ngành khoa học máy tính, đặc biệt là ML.

Hình 5.9: Khi số chiều của dữ liệu tăng lên (từ trái qua phải), số lượng cấu hình cần xét đến có thể tăng theo cấp số mũ. *(Trái)* Trong ví dụ một chiều này, ta có một biến mà ở đó chỉ có 10 vùng cần quan tâm để phân biệt. Khi có đủ số lượng mẫu rơi vào mỗi vùng (mỗi vùng được thể hiện bằng một ô vuông), các thuật toán học có thể dễ dàng tổng quát hóa một cách chính xác. Một cách tổng quát hóa đơn giản là ước lượng giá trị của hàm đích trong mỗi vùng (và có thể nội suy giữa các vùng lân cận). *(Giữa)* Với dữ liệu hai chiều, việc phân biệt 10 giá trị khác nhau của mỗi biến sẽ trở nên khó khăn hơn. Ta phải theo dõi đến $10 \times 10 = 100$ vùng, và cần ít nhất chừng đó mẫu để phủ hết các vùng này. *(Phải)* Khi tăng số chiều lên 3, số vùng sẽ là $10^3 = 1,000$ và cũng cần ít nhất chừng đó mẫu. Một cách tổng quát, với d chiều và v giá trị, ta sẽ cần $O(v^d)$ vùng và mẫu để phân biệt các giá trị trên mỗi trục tọa độ. Đây là một ví dụ về hiểm họa số chiều lớn. Hình ảnh được cung cấp bởi Nicolas Chapados.

Một thách thức đặt ra bởi hiểm họa số chiều lớn là thách thức về thống kê. Như mô tả trong hình 5.9, thách thức về thống kê xuất hiện khi số lượng cấu hình cần xem xét của $\boldsymbol{x}$ lớn hơn rất nhiều so với số lượng mẫu huấn luyện. Để hiểu vấn đề này, xét không gian đầu vào được tổ chức thành dạng lưới như trong ví dụ trên. Ta có thể mô tả không gian số chiều nhỏ với một lượng nhỏ các ô và hầu hết đều chứa dữ liệu. Khi tổng quát cho một điểm dữ liệu mới, ta có thể dễ dàng biết cần phải làm gì thông qua kiểm thử các mẫu huấn luyện nằm trong cùng ô với đầu vào mới đó. Ví dụ, nếu cần ước lượng mật độ xác suất ở một điểm $\boldsymbol{x}$ nào đó, ta có thể chỉ cần trả về số mẫu huấn luyện nằm trong cùng đơn vị thể tích với $\boldsymbol{x}$, chia cho tổng số mẫu huấn luyện. Nếu muốn phân loại một mẫu, ta có thể trả về lớp có số mẫu huấn luyện nhiều nhất trong cùng một ô.

Còn nếu cần thực hiện phép hồi quy, ta chỉ cần tính trung bình các giá trị đích của các mẫu quan sát được trong ô đó. Còn những ô không chứa mẫu thì sao? Vì trong không gian nhiều chiều, số lượng cấu hình là cực lớn, hơn rất nhiều so với số mẫu ta có, thường thì một ô sẽ không có mẫu dữ liệu nào nằm trong đó. Liệu ta có thể kết luận gì về những cấu hình mới này? Nhiều thuật toán ML truyền thống chỉ đơn giản là giả định rằng đầu ra tại một điểm mới nên xấp xỉ đầu ra tại điểm huấn luyện gần nhất.

5.11.2 Tính bất biến cục bộ và cơ chế kiểm soát độ trơn

Để tổng quát hóa tốt, các thuật toán ML cần được dẫn dắt bởi các kinh nghiệm tiền đề về loại hàm mà chúng sẽ học. Ta đã thấy những tiền đề này được kết hợp như dưới dạng các phân bố xác suất của các tham số trong mô hình. Nói một cách tương đối hơn, ta có thể cho rằng kinh nghiệm tiền đề ảnh hưởng đến chính hàm ta cần học một cách trực tiếp, và chỉ ảnh hưởng các tham số mô hình một cách gián tiếp. Sự ảnh hưởng gián tiếp này là do mối quan hệ giữa hàm và các tham số đó. Ngoài ra, ta đã thảo luận về kinh nghiệm tiên đề như một cách chọn ngầm định các thuật toán có tính thiên vị một lớp hàm nào đó hơn các lớp khác, mặc dù sự thiên vị này có thể không được diễn tả (hay thậm chí là không thể diễn tả) bằng một phân phối xác suất biểu diễn mức độ tin tưởng của ta đối với các hàm khác nhau.

Trong số rất nhiều "tiền đề" ngầm định được sử dụng rộng rãi, người ta quan tâm nhiều đến *tiền đề về tính trơn* (smoothness prior), hay *tiền đề về tính bất biến cục bộ* (local constancy prior). Những tiền đề này cho rằng hàm mà ta cần học không nên thay đổi quá nhiều trong một khoảng nhỏ.

Nhiều thuật toán đơn giản hơn chủ yếu dựa vào tiền đề này để tổng quát hóa tốt hơn, nhưng cũng vì thế mà chúng không thể mở rộng để giải quyết những thách thức thống kê trong AI. Xuyên suốt cuốn sách này, chúng tôi mô tả cách mà DL đưa ra những tiền đề (tường minh hoặc ngầm định) bổ sung để giảm sai số tổng quát hóa trong những tác vụ phức tạp. Ở đây, chúng tôi sẽ giải thích tại sao chỉ một mình tiền đề về tính trơn là không đủ để giải quyết những tác vụ này.

Có nhiều cách khác nhau để diễn tả một cách ngầm định hoặc tường minh rằng hàm học tập cần có tính trơn hoặc bất biến cục bộ. Tất cả những phương pháp

khác nhau này được thiết kế để khuyến khích quá trình học để tìm ra một hàm f^* thỏa mãn điều kiện sau

$$f^*(\boldsymbol{x}) \approx f^*(\boldsymbol{x} + \epsilon) \qquad (5.103)$$

với hầu hết các cấu hình $\boldsymbol{x}$ và thay đổi nhỏ ϵ. Nói cách khác, nếu ta biết được một câu trả lời tốt cho một đầu vào $\boldsymbol{x}$ (chẳng hạn, nếu $\boldsymbol{x}$ là một mẫu đã được gán nhãn), thì nó cũng có thể là câu trả lời tốt cho các điểm lân cận của $\boldsymbol{x}$. Nếu có một số câu trả lời tốt cho một số điểm lân cận nhau, ta có thể kết hợp chúng lại (dưới dạng trung bình cộng hoặc nội suy) để đưa ra một câu trả lời tương đồng với những câu trả lời trên càng nhiều càng tốt.

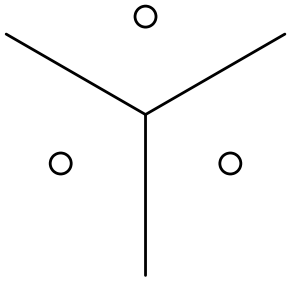
Một ví dụ cực đoan về hướng tiếp cận bất biến cục bộ là họ các thuật toán học k điểm gần nhất. Những bộ dự đoán này là bất biến trên mỗi vùng chứa tất cả các điểm $\boldsymbol{x}$ mà có cùng tập k điểm gần nhất trong tập huấn luyện. Với $k = 1$, số vùng có thể phân tách không thể lớn hơn số mẫu dữ liệu trong tập huấn luyện.

Trong khi thuật toán k điểm gần nhất tính toán đầu ra dựa vào những mẫu dữ liệu nằm lân cận nó thì hầu hết các máy lõi nội suy giữa các giá trị đầu ra của tập huấn luyện gắn với những điểm lân cận. Một lớp hàm lõi quan trọng là họ các *hàm lõi cục bộ* (local kernel), trong đó giá trị của kernel $k(\boldsymbol{u}, \boldsymbol{v})$ là lớn khi $\boldsymbol{u} = \boldsymbol{v}$ và giảm dần khi $\boldsymbol{u}$ và $\boldsymbol{v}$ càng xa nhau. Một hàm lõi cục bộ có thể được coi như một hàm tính độ tương tự thực hiện việc ghép bản mẫu, bằng cách đo lường mức độ tương tự giữa một mẫu kiểm thử $\boldsymbol{x}$ với mỗi mẫu huấn luyện $\boldsymbol{x}^{(i)}$. Nhiều động lực của DL hiện đại được thúc đẩy từ việc tìm hiểu các giới hạn của phương pháp ghép bản mẫu cục bộ và cách thức các mô hình DL có thể xử lý thành công những trường hợp mà các phương pháp ghép bản mẫu cục bộ thất bại [Bengio et al., 2006b].

Các cây quyết định cũng có những giới hạn của phương pháp học dưa trên tính trơn, bởi lẽ chúng chia nhỏ không gian đầu vào thành các vùng với số lượng bằng số lá của cây và sử dụng một tham số (hoặc đôi khi là nhiều hơn một tham số đối với các mở rộng của cây quyết định) phân tách trong mỗi vùng. Nếu hàm mục tiêu yêu cầu một cây với ít nhất n lá để được biểu diễn một cách chính xác, thì cần tối thiểu n mẫu để khớp cái cây đó, và một bội số của n các mẫu huấn luyện để đạt được mức độ tin cậy thống kê nào đó ở kết quả dự đoán.

Nhìn chung, để phân biệt $O(k)$ vùng trong không gian mẫu, tất cả những phương pháp này đòi hỏi cần $O(k)$ mẫu. Thông thường sẽ có $O(k)$ tham số, với $O(1)$ tham số gắn với mỗi vùng trong số $O(k)$ vùng. Trong thuật toán điểm gần nhất, ở

đó mỗi mẫu huấn luyện có thể được sử dụng để định nghĩa tối đa một vùng, được minh họa trong hình 5.10.



> Hình 5.10: Minh họa cách thuật toán điểm gần nhất chia không gian đầu vào thành các vùng. Một mẫu huấn luyện (được biểu diễn bằng một đường tròn) bên trong mỗi vùng xác định biên của vùng (được biểu diễn bằng các đường kẻ). Giá trị y ứng tới mỗi mẫu xác định giá trị đầu ra cho tất cả các điểm trong vùng tương ứng. Những vùng được định nghĩa bởi phép so khớp với điểm gần nhất tạo nên một cấu trúc hình học được gọi là lược đồ Voronoi (Voronoi diagram). Số lượng các vùng tiếp giáp này không thể tăng nhanh hơn số mẫu huấn luyện. Mặc dù hình này minh họa hoạt động cụ thể của thuật toán điểm gần nhất, nhưng những thuật toán ML khác chỉ dựa trên tiền đề về độ trơn cục bộ để khái quát hóa cũng có hành vi tương tự: mỗi mẫu huấn luyện chỉ cung cấp cho thuật toán học về cách tổng quát hóa trong một số vùng lân cận xung quanh mẫu huấn luyện đó.

Liệu có một cách biểu diễn một hàm phức tạp có nhiều vùng cần được phân biệt hơn số mẫu huấn luyện không? Rõ ràng, chỉ giả định về tính trơn của hàm không cho phép thuật toán học làm điều đó. Ví dụ, hình dung rằng hàm cần học có dạng một bàn cờ. Một bàn cờ có nhiều biến thể nhưng chúng có chung một cấu trúc đơn giản. Tưởng tượng xem điều gì xảy ra khi mà số lượng mẫu huấn luyện nhỏ hơn đáng kể số lượng các ô vuông trắng và đen trên bàn cờ. Chỉ dựa trên khái quát hóa cục bộ và tính trơn hay tiền đề về tính bất biến cục bộ, thuật toán học có thể đảm bảo đoán đúng màu của một điểm mới nếu nó nằm trong cùng ô bàn cờ với một mẫu huấn luyện. Nhưng sẽ không có gì đảm bảo rằng thuật toán học có thể mở rộng chính xác đối với các điểm nằm trên ô vuông không có mẫu huấn luyện. Chỉ với tiền đề này, thông tin duy nhất mà một mẫu cho ta biết là màu của ô bàn cờ chứa nó, và cách duy nhất để biết được màu chính xác của cả bản cờ là phủ mỗi ô của nó với ít nhất một mẫu huấn luyện.

Giả thiết về tính trơn và các thuật toán học không tham số gắn với nó sẽ hoạt động rất tốt khi có đủ mẫu huấn luyện để thuật toán học quan sát các điểm nằm trên hầu hết các đỉnh và các điểm nằm dưới hầu hết các thung lũng của hàm ta cần học. Điều này thường đúng khi mà hàm được học là đủ trơn và biến thiên trong số chiều đủ thấp. Trong trường hợp số chiều lớn, ngay cả một hàm trơn cũng có thể thay đổi mượt mà nhưng theo một cách khác nhau theo mỗi chiều. Ngoài ra, nếu đặc tính của hàm này khác nhau tại nhiều vùng, việc mô tả nó thông qua một tập các mẫu huấn luyện sẽ trở nên rất phức tạp. Nếu một hàm là phức tạp (chẳng hạn như muốn phân biệt một số lượng các vùng rất lớn so với số mẫu huấn luyện), liệu ta có thể khái quát hóa tốt hay không?

Câu trả lời cho cả hai câu hỏi này - liệu ta có thể biểu diễn một hàm phức tạp hiệu quả, và liệu hàm ước lượng có thể tổng quát hóa tốt cho các giá trị đầu vào mới - là có. Lí do chính là đối một số lượng lớn các vùng, chẳng hạn $O(2^k)$, có thể được xác định với $O(k)$ mẫu huấn luyện miễn là ta đưa vào một vài mối liên hệ giữa các vùng thông qua các giả thiết bổ sung về phân phối sinh dữ liệu thực tế. Với cách này, ta có thể khái quát hóa một cách không cục bộ [Bengio and Monperrus, 2005; Bengio et al., 2006c]. Rất nhiều các thuật toán DL khác nhau cung cấp các giả thiết khá hợp lý một cách ngầm định hoặc tường minh cho một loạt các nhiệm vụ AI để nắm bắt những lợi thế này.

Những cách tiếp cận khác đối với ML thường đặt ra các giả định mạnh hơn và cụ thể cho từng tác vụ. Ví dụ, ta có thể dễ dàng giải quyết tác vụ bàn cờ phía trên bằng cách đưa vào giả định rằng hàm cần học có tính chu kì. Thông thường, ta sẽ không đưa các giả định mạnh và cụ thể cho từng tác vụ như vậy vào mạng neuron nhân tạo để chúng có thể tổng quát hóa nhiều cấu trúc khác nhau hơn. Các tác vụ AI có cấu trúc quá phức tạp để có thể mô tả bằng những tính chất đơn giản, có các đặc tính được chỉ định một cách thủ công như chu kỳ, vì vậy ta cần các thuật toán học thể hiện các giả định có mục đích chung chung hơn. Ý tưởng cốt lõi trong DL là giả định rằng dữ liệu được tạo ra bởi *sự tổng hợp các các thành tố* (composition of factors) hoặc các đặc trưng (features), tiềm năng ở nhiều cấp độ trong một hệ thống phân cấp. Nhiều giả định chung tương tự khác có thể cải thiện các thuật toán DL hơn nữa. Những giả định này cho phép lợi ích ta nhận được, trong mối quan hệ giữa số lượng mẫu huấn luyện và số vùng có thể phân biệt được, tăng theo cấp số mũ. Chúng tôi sẽ mô tả những lợi ích tăng theo cấp số mũ này trong phần 6.4.1, 15.4 và 15.5. Các lợi ích này đạt được nhờ

việc sử dụng các biểu diễn phân tán đa tầng để đối phó với những thách thức theo cấp số mũ gây bởi hiểm họa số chiều lớn.

5.11.3 Manifold

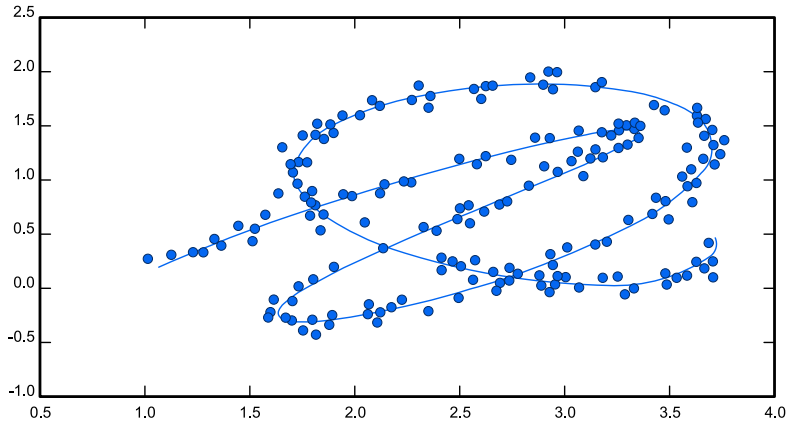
Một khái niệm quan trọng làm nền tảng cho nhiều ý tưởng trong ML là đa tạp (manifold).

Một *đa tạp* là một vùng liên thông. Về mặt toán học, nó là một tập hợp các điểm liên kết với một lân cận xung quanh mỗi điểm. Từ một điểm bất kì, đa tạp, về mặt cục bộ, giống như một không gian Euclide. Trong cuộc sống hàng ngày, ta thường nghĩ bề mặt của Trái Đất là một mặt phẳng 2 chiều, nhưng thực tế nó là một đa tạp hình cầu trong không gian 3 chiều.

Khái niệm về một lân cận xung quanh mỗi điểm ngụ ý sự tồn tại của các phép biến hình được áp dụng để di chuyển trên đa tạp từ một vị trí đến một vị trí lân cận. Trong ví dụ bề mặt Trái Đất là một đa tạp, một người có thể đi về phía Đông, Tây, Nam, hoặc Bắc.

> ***ND***: Xem thêm về đa tạp tại đây.

Mặc dù ta có một khái niệm toán học chính thức để diễn tả ý nghĩa của từ "đa tạp", nhưng trong ML, nó được sử dụng một cách đại khái để chỉ định một tập hợp các điểm liên thông với nhau mà ta có thể xấp xỉ gần đúng chỉ bằng việc xét một số lượng nhỏ bậc tự do, hoặc số chiều, được nhúng trong một không gian nhiều chiều. Mỗi chiều tương ứng với một phương biến thiên cục bộ. Hình 5.11 cho ta thấy ví dụ về tập huấn luyện nằm gần đa tạp một chiều được nhúng trong không gian hai chiều. Trong phạm vi của ML, ta cho phép số chiều của đa tạp thay đổi từ một điểm tới một điểm khác. Điều này thường xảy ra khi một đa tạp tự giao cắt với chính nó. Chẳng hạn, xét một đa tạp có dạng hình số 8. Đa tạp này là một chiều ở hầu hết mọi nơi, duy chỉ có điểm nằm ở vùng giao cắt tại trung tâm đa tạp là có hai chiều.



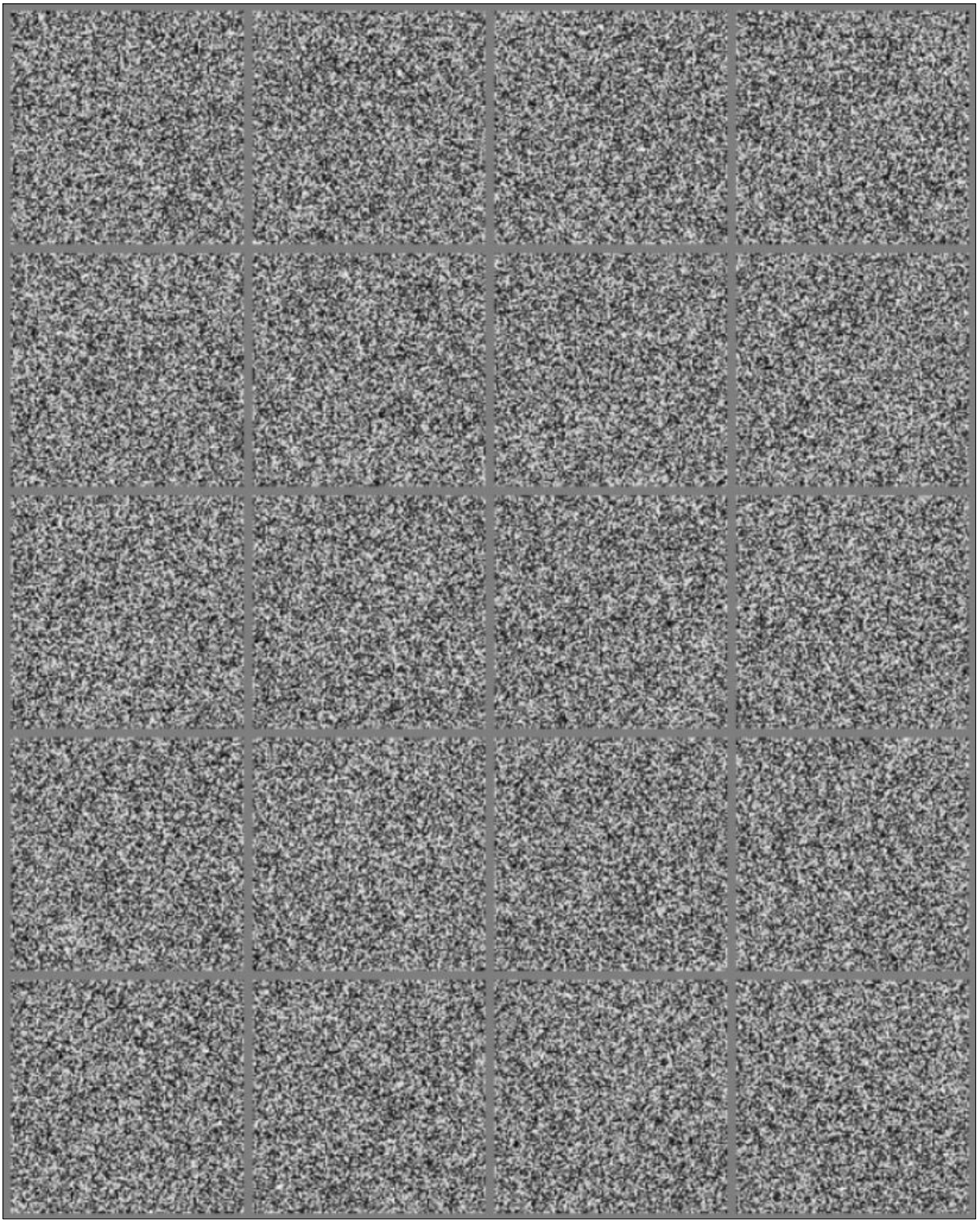
> Hình 5.11: Dữ liệu được lấy mẫu từ một phân phối trong một không gian hai chiều mà thực tế được tập trung gần một đa tạp một chiều, có hình dạng một dây xoắn. Đường liền mạch biểu thị phần đa tạp mà thuật toán học cần luận ra.

Nhiều bài toán ML có vẻ như không có hy vọng được giải quyết nếu ta đòi hỏi thuật toán ML phải học các hàm với các biến thiên theo mọi hướng của $\mathbb{R}^n$. Các thuật toán *học đa tạp* (manifold learning) vượt qua trở ngại này bằng cách giả định rằng $\mathbb{R}^n$ chứa hầu hết những dữ liệu đầu vào không hợp lệ, và các điểm đầu vào ta quan tâm chỉ xuất hiện dọc theo một tập các đa tạp chứa một tập con các điểm, với các biến thiên cần quan tâm ở đầu ra của hàm cần học chỉ xảy ra dọc theo đa tạp, hoặc các biến thiên cần quan tâm chỉ xảy ra khi ta chuyển từ một đa tạp này sang một đa tạp khác. Học đa tạp được đưa vào trong trường hợp dữ liệu liên tục và học không giám sát, mặc dù ý tưởng tập trung xác suất này có thể được tổng quát hoá cho cả dữ liệu rời rạc và trong học có giám sát: giả định chính ở đây vẫn là khối xác suất được tập trung cao độ.

Giả định rằng dữ liệu nằm trên đa tạp ít chiều có thể không phải lúc nào cũng đúng hoặc hữu dụng. Chúng tôi cho rằng trong phạm vi của các tác vụ AI, ví dụ như trong xử lý ảnh, âm thanh, hay văn bản, thì giả định về đa tạp ít nhất cũng gần đúng. Bằng chứng cho giả định này đến từ hai cách nhìn nhận.

Cách nhìn nhận thứ nhất ủng hộ *giả thuyết đa tạp* (manifold hypothesis), đó là phân phối xác suất của hình ảnh, chuỗi văn bản, và âm thanh trong đời thực có tính tập trung cao. Nhiễu đồng đều (uniform noise) về cơ bản không bao giờ giống với dữ liệu đầu vào có cấu trúc từ những miền này. Thay vào đó, hình 5.12 cho thấy các điểm được lấy mẫu đồng đều nhìn rất giống với các mô thức tĩnh được biểu thị trên màn hình tivi khi không có tín hiệu sóng. Tương tự, nếu bạn

soạn thảo một tài liệu bằng cách chọn ra các chữ cái một cách đồng đều và ngẫu nhiên, thì xác suất mà bạn có được một văn bản có ý nghĩa là bao nhiêu? Câu trả lời là gần như bằng 0, bởi vì hầu hết các chuỗi chữ cái dài đều không tương đồng với các chuỗi ngôn ngữ tự nhiên: phân phối của chuỗi ngôn ngữ tự nhiên chiếm một thể tích rất nhỏ trong tổng không gian của các chuỗi chữ cái.



> Hình 5.12 Lấy mẫu ảnh theo phân phối đồng đều (bằng cách lấy ra mỗi điểm ảnh ngẫu nhiên theo một phân phối đều) sẽ cho ra các ảnh nhiễu. Mặc dù xác suất tạo ra một bức ảnh có nghĩa là lớn hơn 0, nhưng chúng tôi chưa từng quan sát được hiện tượng này xảy ra trong thực tế. Điều này gợi ý cho ta rằng những bức ảnh xuất hiện trong các ứng dụng AI chỉ chiếm một tỷ lệ không đáng kể trong thể tích của không gian các hình ảnh.

Tất nhiên, những phân phối có xác suất tập trung là chưa đủ để diễn tả việc dữ liệu nằm trên một số lượng nhỏ đa tạp. Ta cũng cần làm rõ rằng những mẫu dữ liệu ta gặp phải được liên kết với nhau bằng những mẫu khác, trong đó mỗi mẫu được bao quanh bởi các mẫu dữ liệu rất tương đồng mà ta có thể tiếp cận bằng cách áp dụng các phép biến hình để di chuyển trên đa tạp. Cách nhìn nhận thứ hai hỗ trợ cho giả thuyết đa tạp đó là ta có thể tưởng tượng ra những vùng lân cận và phép biến hình, ít nhất là trên cục bộ. Xét trường hợp với các bức ảnh, ta có thể nghĩ ra nhiều phép biến hình khả thi cho phép lần theo dấu vết của một đa tạp trong không gian các hình ảnh: ta có thể dần dần điều chỉnh độ sáng hoặc tối của ảnh, từ từ di chuyển hay xoay các vật thể trong ảnh, dần thay đổi màu trên bề mặt các vật thể, và nhiều phép biến hình khác nữa. Hầu hết các ứng dụng đều bao gồm nhiều đa tạp khác nhau. Ví dụ, đa tạp của các bức ảnh chụp khuôn mặt con người có thể không liên kết với đa tạp của các bức ảnh chụp khuôn mặt con mèo.

Những thí nghiệm này cho ta nhiều lý do trực quan giúp hỗ trợ cho giả thuyết đa tạp. Nhiều thí nghiệm chặt chẽ hơn [Cayton, 2005; Narayanan and Mitter, 2010; Schölkopf et al., 1998; Roweis and Saul, 2000; Tenenbaum et al., 2000; Brand, 2003; Belkin and Niyogi, 2003; Donoho and Grimes, 2003; Weinberger and Saul, 2004] hỗ trợ giả thuyết này trên nhiều bộ dữ liệu được quan tâm trong AI.

Khi dữ liệu nằm trên một đa tạp số chiều thấp, các thuật toán ML có thể biểu diễn dữ liệu một cách tự nhiên nhất dựa vào các tọa độ của đa tạp, thay vì các tọa độ trong tập hợp $\mathbb{R}^n$. Trong cuộc sống hàng ngày, ta có thể giả định các con đường bộ là các đa tạp 1 chiều được nhúng trong không gian 3 chiều. Ta đưa ra chỉ dẫn để đi tới các địa điểm cụ thể dựa trên các số nhà dọc theo con đường 1 chiều, chứ không phải dựa trên các tọa độ trong không gian 3 chiều. Việc trích xuất những tọa độ đa tạp này là một thách thức, nhưng lại có tiềm năng cải thiện được rất nhiều các thuật toán ML. Quy tắc căn bản này được áp dụng trong nhiều ngữ cảnh. Hình 5.13 cho thấy cấu trúc đa tạp của một bộ dữ liệu bao gồm các khuôn mặt. Đến cuối cuốn sách này, chúng tôi sẽ phát triển các phương

pháp cần thiết để học những cấu trúc đa tạp như vậy. Trong hình 20.6, ta sẽ thấy cách một thuật toán ML hoàn thành mục tiêu này như nào.

> Hình 5.13: Các mẫu huấn luyện thu được từ Bộ dữ liệu khuôn mặt Đa góc nhìn QMUL [Gong et al., 2000], trong đó các đối tượng được yêu cầu di chuyển khuôn mặt theo cách nhất định để biểu diễn đa tạp 2 chiều tương ứng với hai góc quay. Ta muốn các thuật toán học có khả năng khám phá và xử lý các tọa độ đa tạp. Hình 20.60 minh họa một quá trình như vậy.

Phần I kết thúc ở đây, chúng tôi đã điểm qua những khái niệm cơ bản trong toán học và ML sẽ đi theo ta xuyên suốt các phần còn lại của cuốn sách. Bây giờ các bạn đã được chuẩn bị một cách đầy đủ để tiếp tục nghiên cứu về DL.