

NHẬP MÔN HỌC MÁY VÀ KHAI PHÁ DỮ LIỆU

TÀI LIỆU ĐỌC

TẬP THỂ TÁC GIẢ:

PGS.TS. THÂN QUANG KHOÁT

PGS.TS. NGUYỄN THỊ KIM ANH

TS. ĐỖ TIẾN DŨNG

TS. NGÔ VĂN LINH

TS. NGUYỄN ĐỨC ANH

ĐƠN VỊ: KHOA KHOA HỌC MÁY TÍNH

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐẠI HỌC BÁCH KHOA HÀ NỘI

MỤC LỤC

MỞ ĐẦU	3
CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ HỌC MÁY VÀ KHAI PHÁ DỮ LIỆU	5
1.1 Các khái niệm và bài toán học cơ bản	5
1.2 Quy trình xây dựng một hệ thống học	11
1.3 Vài vấn đề cần biết khi dùng ML	14
1.4 Thu thập và tiền xử lý dữ liệu	18
CHƯƠNG 2. HỒI QUI VÀ PHÂN CỤM	27
2.1 Hồi qui	27
2.2 Phân cụm	35
CHƯƠNG 3. PHÂN LOẠI	58
3.1 Học dựa trên lảng giềng	58
3.2 Cây quyết định và Rừng ngẫu nhiên	65
3.3 Máy véctơ hỗ trợ (SVM)	75
3.4 Học dựa trên xác suất	88
3.5 Đánh giá hiệu năng và lựa chọn tham số	104
3.6 Mạng nơron nhân tạo	116
CHƯƠNG 4. KHAI PHÁ DỮ LIỆU	133
4.1 Khai phá dữ liệu	133
4.2 Khai phá luật kết hợp	141
4.3 Mở rộng: Hiệu chỉnh	157

MỞ ĐẦU

Một trong những khát vọng lớn của ngành Trí tuệ nhân tạo (Artificial Intelligence - AI) là tạo ra các máy móc có khả năng hỗ trợ và sống hòa nhập với con người. Những máy móc ấy cần có khả năng hiểu, ứng dụng một cách đúng đắn tri thức và văn hoá của con người trong suốt thời gian sống của chúng. Đặc biệt, một khả năng không thể thiếu cho những máy móc ấy là khả năng học từ những quan sát về môi trường xung quanh.

Con người có thể học từ rất nhiều nguồn khác nhau, như sách, báo, tin tức, bạn bè, người thân, thầy, cô,... Chúng ta có thể học để làm những thứ đơn giản (như cầm đua, đi lại, đạp xe, nấu cơm, giặt đồ,...), cho đến những thứ rất phức tạp (như lái xe ô tô, kinh doanh,...). Chúng ta có thể tự đúc kết kinh nghiệm sau khi đã quan sát nhiều lần về các sự vật hoặc hiện tượng. Đó là khả năng tự học rất tuyệt vời mà tự nhiên đã ban tặng.

Máy móc cũng có thể học như thế. Chúng có thể học được gì đó sau khi đã quan sát nhiều ví dụ khác nhau về một thứ gì đó. Đây là khả năng tự học từ các ví dụ (Learning from examples) hoặc quan sát (observations). Trong thực tế, các ví dụ này có thể thu thập được khá dễ dàng (qua camera, GPS, ăngten,...) và thường được gọi là Dữ liệu (Data). Máy có thể học tri thức mới từ dữ liệu đã thu thập được trong quá khứ, và từ đó có thể vận dụng tri thức đó để giải quyết các vấn đề trong tương lai, chẳng hạn như dự đoán, khám phá tri thức, lập kế hoạch. Khả năng này có thể giúp con người giải quyết nhiều thách thức trong thực tế.

Chúng ta đang sống trong kỷ nguyên lớn, nơi mà dữ liệu mới có thể được sinh ra hay được thu thập rất dễ dàng. Thông qua một số thiết bị hoặc phần mềm đơn giản, chúng ta có thể thu thập một lượng lớn dữ liệu từ một số nguồn (chẳng hạn Internet). Hơn thế nữa, những tiến bộ vượt bậc gần đây trong Học máy có thể giúp chúng ta sinh ra các tập dữ liệu theo mong muốn, bởi các mô hình sinh (Generative models). Do đó, một nhu cầu thực tế là làm sao có thể phát hiện ra được tri thức đang ẩn chứa trong lượng lớn dữ liệu đó. Tri thức ấy có thể hữu ích cho nhiều tác vụ khác trong doanh nghiệp, những người ra quyết định, hoạch định chính sách, ...

Trong khóa học này, chúng ta sẽ tìm hiểu những kiến thức căn bản nhất trong lĩnh vực Học máy và Khai phá dữ liệu. Các nội dung cụ thể sẽ được trình bày ở các chương

tiếp theo. Bộ cục của tài liệu này bao gồm:

Chương 1 sẽ trình bày những khái niệm, bài toán cơ bản nhất, và một số vấn đề của các hệ thống có khả năng học.

Mục 1.4 trình bày về vấn đề thu thập dữ liệu và một số phương pháp tiền xử lý dữ liệu. Đây là một bước cơ bản nhưng cần thiết trong quá trình xây dựng một hệ thống có khả năng học hoặc khám phá tri thức.

Chương 2 bàn luận bài toán hồi qui và mô hình tuyến tính. Một số phương pháp huấn luyện khác nhau sẽ được trình bày, gồm bình phương tối thiểu, Ridge, và LASSO.

Mục 2.2 sẽ trình bày bài toán phân cụm, một ví dụ của lớp bài toán học không giám sát. Một số phương pháp phân cụm sẽ được trình bày, và một số vấn đề cũng sẽ được bàn luận chi tiết.

Chương 3 sẽ trình bày một số mô hình học máy để giải quyết bài toán phân loại, gồm K-NN, cây quyết định, rừng ngẫu nhiên, SVM, mô hình xác suất, Naive Bayes, mạng nơron nhân tạo. Chương này cũng bàn luận một số phương pháp để giúp ta đánh giá hiệu năng của một mô hình học máy, và việc lựa chọn siêu tham số cho chúng.

Chương 4 chứa các khái niệm và vấn đề cơ bản của Khai phá dữ liệu. Một ví dụ cụ thể là khai phá luật kết hợp sẽ được trình bày. Ngoài ra còn có thêm một số bàn luận về kỹ thuật hiệu chỉnh để giúp đối diện với vấn đề quá khớp trong học máy.

CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ HỌC MÁY VÀ KHAI PHÁ DỮ LIỆU

1.1 Các khái niệm và bài toán học cơ bản

Đầu tiên, chúng ta hãy cùng làm quen một số khái niệm và bài toán cơ bản nhất.

1.1.1 Vài quan điểm về ML

Học (learning) là một khả năng căn bản của các hệ thống có trí thông minh thực sự. Do đó việc tạo khả năng học cho máy móc đã được đề ra và nghiên cứu khá lâu trước đây trong lĩnh vực AI. Tuy nhiên, chúng ta cần hiểu rõ học nghĩa là gì. Dưới đây là một quan điểm về việc học.

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time." [Simon, 1983]

Quan điểm này cho ta một cái nhìn khá tổng quan về việc học, ở con người và các hệ thống máy móc. Khả năng học sẽ giúp các hệ thống ấy có khả năng thích nghi, tức là chúng có thể tự cải thiện hiệu năng khi giải quyết một nhiệm vụ (task) hoặc một lớp nhiệm vụ nào đó.

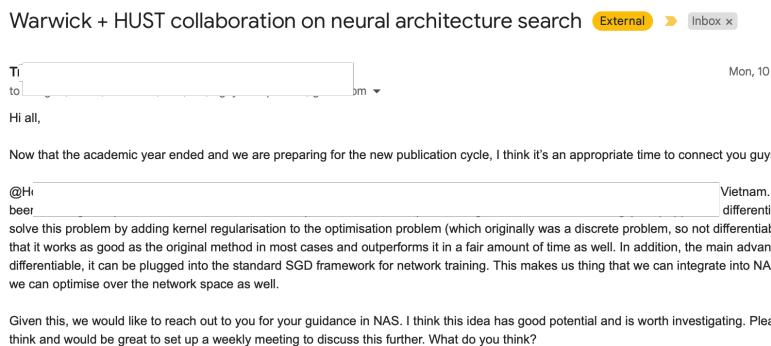
Quan điểm trên cũng có thể áp dụng cho máy để giúp ta hiểu về Học máy (Machine Learning - ML). Mặc dù vậy, cho đến nay có khá nhiều quan điểm (cách nhìn) khác nhau về ML. Dưới đây là một vài ví dụ:

"Build systems that automatically improve their performance" [Simon, 1983]

"Program computers to optimize a performance criterion using example data or past experience" [Alpaydin, 2020]

Dù được phát biểu ở nhiều góc nhìn khác nhau, nhưng chúng ta đều thấy một điểm chung là "*hiệu năng*" (performance). Nếu đứng ở góc độ ứng dụng thì chúng ta có thể xem: *ML là một lĩnh vực cung cấp một con đường để giải quyết một nhiệm vụ thực tế, thông qua khả năng tự cải thiện, dựa vào dữ liệu hoặc kinh nghiệm đã có.*

Cụ thể hơn, theo Mitchell [1997], một hệ thống **có khả năng học** nếu nó có khả năng tự cải thiện bản thân nó theo tiêu chí P khi giải quyết một nhiệm vụ T , dựa vào dữ liệu



Hình 1: Ví dụ về một email.

hoặc kinh nghiệm E . Một bài toán học máy có thể được mô tả bằng bộ ba (T, P, E). Như vậy, đây là một quan điểm rất cụ thể về khả năng học của một hệ thống. Theo quan điểm này, chúng ta có thể đưa nhiều nhiệm vụ trong thực tế về các bài toán học máy. Dưới đây là vài ví dụ.

Ví dụ 1 (Lọc thư rác - Spam filtering) Chúng ta muốn xây dựng một hệ thống có khả năng tự động lọc thư rác (spam) trong hộp thư điện tử (email). Hình 1 có ví dụ về một email.

Có thể đưa nhu cầu này về bài toán học máy như sau:

- T : Cần phán đoán một email cho trước có phải là thư rác hay không
- P : Độ chính xác phán đoán (tỷ lệ emails được dự đoán đúng)
- E : Tập hợp các email trong quá khứ, mỗi email đã được gán nhãn (rác/thường) tương ứng.

Ví dụ 2 (Mô tả ảnh - Image captioning) Chúng ta muốn xây dựng một hệ thống có khả năng tự động đưa ra mô tả nội dung bên trong bức ảnh.

Có thể đưa nhu cầu này về bài toán học máy như sau:

- T : Cần đưa ra mô tả ngắn gọn về nội dung bên trong một bức ảnh cho trước
- P : Độ chính xác của mô tả
- E : Tập hợp các bức ảnh, mỗi bức ảnh đã được mô tả bởi một câu ngắn gọn về nội dung bên trong bức ảnh đó. Hình 2 chứa một tập ví dụ.¹

¹Chúng được lấy từ <https://openai.com/dall-e-3>.



lychee-inspired spherical chair



a girl giving cat a gentle hug



a small hedgehog holding a piece of watermelon

Hình 2: Tập các bức ảnh và các mô tả về chúng.

1.1.2 Bản chất của việc học, suy diễn, và phán đoán

Con người có khả năng học rất tuyệt vời. Chúng ta có thể học từ những thứ đơn giản đến những thứ rất phức tạp (ví dụ như lái xe). Chúng ta có thể tự đúc rút ra kinh nghiệm của riêng mình khi đã quan sát nhiều lần, và sau đó chúng ta có thể sử dụng kinh nghiệm ấy để giải quyết một công việc nào đó (ví dụ, lái xe). Như vậy, mỗi khi học, bộ não của chúng ta sẽ nhận nhiều tín hiệu (thông qua các quan sát bằng mắt, tai, ...) từ bên ngoài, rồi thực hiện xử lý hoặc tính toán nào đó, để thu được tri thức hoặc kinh nghiệm. Những tri thức này có thể được dùng để giải quyết một nhiệm vụ (trong tương lai).

Như vậy, bản chất của mỗi quá trình học là tìm một hàm y^* nào đó mà kết nối mỗi đầu vào x với đầu ra y . Nghĩa là quá trình học sẽ đi tìm một hàm (chưa biết) có dạng:

$$y^* : x \mapsto y \quad (1)$$

trong đó:

- x là một quan sát (observation) hoặc tri thức cũ. Mỗi quan sát thường ở dưới dạng một tập tín hiệu và thường được gọi là một *mẫu dữ liệu* (data instance) hoặc một *ví dụ* (example).
- y có thể là tri thức mới hoặc kinh nghiệm mới. Đôi khi nó là một phán đoán (prediction).

Trong thực tế, chúng ta thường không biết hàm y^* , về hình dạng hay tính chất của nó. Nhưng đôi khi chúng ta có thể thu thập được một số quan sát $\{x_1, x_2, \dots, x_N\}$ về đầu vào x và một số đầu ra $\{y_1, y_2, \dots, y_M\}$ tương ứng, tức là $y_i = y^*(x_i)$ đối với vài mẫu thứ i . Máy sẽ cần tìm ra hàm y^* từ một tập dữ liệu $D = \{x_1, x_2, \dots, x_N, y_1, y_2, \dots, y_M\}$ đã thu

thập trong quá khứ. Tập \mathbf{D} thường được gọi là **tập huấn luyện (training set)** hoặc *tập học*.

Định nghĩa 1 (Bài toán học - Learning problem) *Cần tìm ra hàm $y^* : \mathbf{x} \mapsto y$ từ một tập dữ liệu $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N, y_1, y_2, \dots, y_M\}$ cho trước.*²

trong đó:

- Mỗi \mathbf{x}_i là một quan sát cụ thể về đầu vào \mathbf{x} .
- y_j là một quan sát về y , sao cho $y_j = y^*(\mathbf{x}_j)$. Chúng thường được gọi là *nhãn (label)* hoặc *phản hồi (response)* hoặc *đầu ra (output)*.
- \mathbf{x} thường thuộc một không gian \mathcal{X} nào đó và thường được gọi là *không gian dữ liệu (data space)*. Nghĩa là $\mathbf{x} \in \mathcal{X}$. Còn y thường thuộc một tập \mathcal{Y} hữu hạn hoặc vô hạn. \mathcal{Y} có thể được gọi là *không gian đầu ra* hoặc *tập nhãn*.
- Thông thường $N \geq M$, nghĩa là số lượng nhãn thu thập được thường ít hơn số mẫu đầu vào.

Có thể nói rằng bài toán tìm ra hàm y^* từ một tập dữ liệu \mathbf{D} là một thách thức lớn nếu không có thông tin thêm về hàm đó. Một lý do quan trọng là không gian tất cả các hàm quá lớn. Do đó đây là một bài toán bất khả thi.

Một cách làm khả thi và phổ biến trong ML là sử dụng một lớp hàm \mathcal{H} mà có thể dễ dàng làm việc, rồi tìm ra một hàm $f \in \mathcal{H}$ để xấp xỉ hàm y^* . Một **thuật toán học (learning algorithm)** sẽ đi tìm hàm f này khi cho trước tập \mathbf{D} và lớp hàm \mathcal{H} . Thông thường chúng ta hy vọng rằng hàm f sẽ xấp xỉ tốt hàm y^* tại mọi điểm, nghĩa là $f(\mathbf{x}) \approx y^*(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.

Sau khi học, chúng ta thu được hàm f . Hàm này có thể chứa tri thức hoặc kinh nghiệm mới cho máy. Đôi khi nó cũng được gọi là *mô hình (model)*. Chúng ta có thể dùng hàm này để đưa ra dự đoán nhãn $f(\mathbf{z})$ cho mỗi mẫu dữ liệu \mathbf{z} trong tương lai. Giai đoạn này thường được gọi là **phán đoán (predict)** hoặc **suy diễn (infer)**.

²Trong một số ứng dụng thực tế, hàm y^* có thể ở dạng phức tạp hơn nhiều. Ví dụ hàm thay đổi theo thời gian, hàm của biến ngẫu nhiên \mathbf{x} , hàm có đầu ra dạng cấu trúc, ... Tuy nhiên, ở môn học này chúng ta sẽ xét dạng cơ bản nhất để dễ dàng nắm bắt các khái niệm và một số vấn đề quan trọng trong ML.



Hình 3: Ví dụ về một tập dữ liệu đã được gắn nhãn. Mỗi ảnh là một mẫu dữ liệu, và nhãn tương ứng ("mèo" hoặc "chó").

1.1.3 Hai bài toán học cơ bản

Như đã bàn luận ở trên, bài toán học về cơ bản là khó. Để dễ dàng làm việc hơn thì người ta thường chia thành các bài toán nhỏ hơn. Tiếp theo, chúng ta sẽ tìm hiểu hai bài toán con.

Định nghĩa 2 (Học có giám sát - Supervised learning) *Cần tìm hàm y^* : $\mathbf{x} \mapsto y$ từ một tập dữ liệu $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M, y_1, y_2, \dots, y_M\}$ cho trước, trong đó $y_j = y^*(\mathbf{x}_j)$ với mỗi chỉ số $j \in \{1, 2, \dots, M\}$.*

Trong bài toán này, mỗi mẫu dữ liệu \mathbf{x}_j trong tập học có một nhãn y_j tương ứng. Và như thế, tập \mathbf{D} chứa M cặp (\mathbf{x}_j, y_j) . Ở đây y_j có thể coi là thông tin để hướng dẫn (supervision) việc tìm ra hàm y^* . Trong thực tế đôi khi chúng ta có thể thu thập được những nhãn này khá dễ dàng, chẳng hạn nhãn spam cho một email, tên con vật có trong một bức ảnh. Hình 3 cho ví dụ về một tập dữ liệu đã có nhãn.

Tùy thuộc vào miền \mathcal{Y} của đầu ra y , chúng ta có thể làm việc với các bài toán nhỏ hơn, như

- **Phân loại (classification):** nếu miền \mathcal{Y} rời rạc và hữu hạn. Ví dụ $\mathcal{Y} = \{\text{thường, rác}\}$, $\mathcal{Y} = \{\text{mèo, chó}\}$, $\mathcal{Y} = \{\text{thích, yêu, ghét, giận}\}$. Hình 3 mô tả một tập huấn luyện cho bài toán phân loại với $\mathcal{Y} = \{\text{mèo, chó}\}$.
- **Hồi qui (regression):** nếu $\mathcal{Y} \subseteq \mathbb{R}$, tức là miền số thực

Như vậy đối với bài toán phân loại, chúng ta thường có một tập nhãn lớp \mathcal{Y} chọn trước và cố định. Nếu mỗi mẫu \mathbf{x} có một $y \in \mathcal{Y}$ (tức là chỉ nhận một giá trị trong \mathcal{Y})

thì chúng ta đang làm việc với bài toán *Phân loại nhiều lớp* (multiclass classification). Nếu tập \mathcal{Y} chỉ có 2 phần tử duy nhất thì đôi khi người ta gọi đó là bài toán *Phân loại nhị phân* (binary classification). Trong trường hợp nhãn y cho mỗi mẫu dữ liệu x có thể nhận vài giá trị trong \mathcal{Y} thì chúng ta đang làm việc với bài toán *Phân loại đa nhãn* (multilabel classification).

Ứng dụng: có rất nhiều nhu cầu trong thực tế có thể đưa về một bài toán học có giám sát, chẳng hạn:

- Phân loại: lọc email, dự đoán mức độ rủi ro của một hồ sơ vay tín dụng, dự đoán bất thường trong giao dịch tài khoản ngân hàng, ...
- Phân loại đa nhãn: phân tích quan điểm trong một đoạn văn, gán tag cho ảnh, ...
- Hồi qui: dự đoán chỉ số chứng khoán cho mỗi phiên giao dịch, dự báo lượng mưa trong ngày, dự báo độ ẩm không khí, dự báo giá nhà, ...

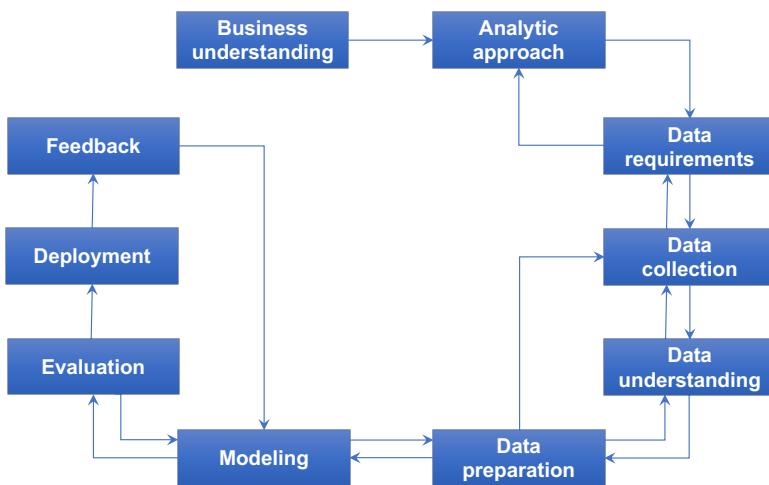
Định nghĩa 3 (Học không giám sát - Unsupervised learning) *Tìm hàm $y^* : \mathbf{x} \mapsto y$ từ một tập dữ liệu $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\}$ cho trước.*

Có rất nhiều nhu cầu trong thực tế có thể đưa về bài toán này. Chẳng hạn, chúng ta muốn gom các bình luận của người dùng khác nhau về các nhóm nhỏ để xem họ đang nói về những gì; chúng ta muốn phát hiện các tương tác ẩn giữa các nhà chính trị gia thông qua tin tức; chúng ta muốn phát hiện xu hướng (thị hiếu) gần đây của người dùng thông qua các trao đổi (comments, posts) trên Facebook, ...

Dễ dàng nhận thấy, trong bài toán học không giám sát, tập huấn luyện không có nhãn cho các mẫu dữ liệu. Mặc dù các mẫu dữ liệu có thể thu thập dễ, nhưng việc thu thập nhãn cho chúng đôi khi rất tốn kém hoặc phức tạp. Ví dụ, "xu hướng" là thứ rất khó thu thập đối với nhiều người, do khả năng phát hiện xu hướng bị hạn chế. Trong mục 2.2, chúng ta sẽ làm quen với một ví dụ cơ bản về bài toán này.

1.1.4 Phân biệt giữa Học máy và Khai phá dữ liệu

Ở phần trước chúng ta đã thấy rằng ML có thể giúp tạo ra một hệ thống mà có thể học từ dữ liệu để tìm ra tri thức mới. Khả năng tìm ra tri thức này có liên hệ chặt chẽ với một lĩnh vực có tên gọi là **Khai phá dữ liệu** (Data mining - DM) [Han et al., 2023].



Hình 4: Quy trình xây dựng một hệ thống dựa trên học máy.

Mỗi quan tâm chính của DM là việc tìm ra những tri thức (knowledge) hoặc mẫu (patterns) ẩn trong các tập dữ liệu lớn, bằng việc sử dụng các thuật toán cụ thể nào đó một cách (bán) tự động. Những tri thức này thường không tường minh, ở nhiều hình dạng khác nhau, và có thể hữu ích trong tương lai. DM là một bước chính trong toàn bộ quy trình về phát hiện tri thức (KDD) [Fayyad et al., 1996].

Như vậy có một mối liên hệ mật thiết giữa ML và DM. Mỗi thuật toán học thường cần tìm ra tri thức từ một tập dữ liệu cho trước. Do đó các thuật toán ML hay được sử dụng cho DM hay KDD. Tuy nhiên "học" chưa thể coi là khám phá tri thức [Simon, 1983], bởi việc học đôi khi chỉ thu được những tri thức đã sẵn có từ người hoặc hệ thống khác (ví dụ học lái xe, học viết, ...). Trong khi đó, KDD thường muốn tìm ra những tri thức mới mẻ và có tiềm năng hữu ích.

1.2 Quy trình xây dựng một hệ thống học

Để xây dựng một hệ thống có khả năng học, chúng ta cần thực hiện những bước chính như trong Hình 4.³ Cụ thể như sau:

1. *Business understanding*: Đầu tiên chúng ta cần hiểu rõ nhu cầu thực tế đang được đặt ra để giải quyết. Đôi khi nhu cầu đó có thể nằm ngoài lĩnh vực hiểu biết của chúng ta. Cho nên cần tìm hiểu rõ nó là gì.
2. *Analytic approach*: Chúng ta cần đưa nhu cầu trên về một bài toán ML. Ở bước này, chúng ta cần phát biểu bài toán ML rõ ràng, phù hợp để giải quyết nhu cầu

³Một số bước có tham khảo từ <http://www.theta.co.nz>.

thực tế kia. Chú ý rằng, một nhu cầu thực tế có thể đưa về nhiều bài toán ML khác nhau. Ví dụ, nhu cầu "chuẩn đoán bệnh viêm gan" có thể đưa về bài toán phân loại, nhưng cũng có thể đưa về một bài toán ML khác; nhu cầu "dự đoán giá nhà" có thể đưa về bài toán hồi qui, nhưng cũng có thể đưa về bài toán phân loại. Lựa chọn ở bước này sẽ quyết định trực tiếp bước sau.

3. *Data requirements:* Tiếp theo chúng ta cần xác định rõ các yêu cầu về tập dữ liệu cần thu thập để huấn luyện hệ thống, tuỳ vào bài toán ML đã chọn ở trên. Ví dụ, nếu ở trên ta chọn bài toán hồi qui thì mỗi mẫu dữ liệu huấn luyện cần có nhãn. Nhãn là yêu cầu bắt buộc trong trường hợp này. Ngoài ra có thể đưa thêm một số yêu cầu khác.
4. *Data collection:* Sau khi đã biết các yêu cầu về dữ liệu, chúng ta cần xây dựng một hệ thống (bán) tự động hoặc cử người để thu thập tập dữ liệu huấn luyện. Chú ý rằng, nếu thấy có vấn đề gì đó trong việc thu thập dữ liệu, chẳng hạn quá đắt đỏ, thì cần xem xét chỉnh sửa hai bước trên.
5. *Data understanding:* Sau khi có một tập dữ liệu, chúng ta có thể thực hiện một vài tính toán đơn giản hoặc dùng các công cụ trực quan hóa để xem xét một vài tính chất của dữ liệu. Ví dụ, xem sự đa dạng của một thuộc tính, mức độ trống/thiếu giá trị trong các mẫu dữ liệu, ... Sau bước này, chúng ta có thể thu được một vài hiểu biết về đặc trưng của dữ liệu trong bài toán này. Chúng có thể hữu ích cho các bước sau.
6. *Data preparation:* Tiếp đó, chúng ta cần thực hiện tiền xử lý dữ liệu. Tập dữ liệu thu thập được thường ở dạng thô. Do đó cần đưa chúng về dạng phù hợp hơn với các mô hình ML. Có thể thực hiện một vài bước nhỏ, chẳng hạn lọc nhiễu, loại bỏ lỗi, biến đổi về dạng vectơ, ... Chương 1.4 sẽ bàn luận kỹ hơn.
7. *Modeling:* Sau khi có tập dữ liệu đã được tiền xử lý \mathbf{D} , chúng ta chọn một mô hình ML phù hợp và huấn luyện nó từ \mathbf{D} .
8. *Evaluation:* Sau khi có mô hình vừa huấn luyện, chúng ta cần tiến hành đánh giá hiệu năng (chất lượng) của nó. Cần dùng chiến lược và độ đo phù hợp để đánh giá. Nếu thấy mô hình chưa tốt, chúng ta có thể quay lại những bước trước để kiểm tra hoặc chỉnh sửa. Ví dụ, kiểm tra việc huấn luyện đã thành công chưa, tham số đã

tốt chưa, mô hình có phù hợp với bài toán không, ... Đôi khi cần kiểm tra việc tiền xử lý, hoặc quá trình thu thập dữ liệu.

9. *Deployment*: Nếu mô hình được huấn luyện tốt thì có thể dùng nó để đưa vào triển khai hệ thống (sản phẩm) đầy đủ.

10. *Feedback*: Hệ thống sau khi hoàn thành có thể cần người dùng đánh giá. Ở bước này, chúng ta thu thập các phản hồi của người dùng về hệ thống. Nếu các phản hồi cho thấy hệ thống chưa đủ tốt thì có thể quay trở lại các bước phía trước để cải thiện. Ngược lại thì xem như hệ thống đạt yêu cầu và việc xây dựng đã thành công.

Đối với nhu cầu khai phá dữ liệu, một số bước ở trên có thể sử dụng. Chi tiết sẽ được trình bày trong Chương 4.

1.2.1 Hàm cần học và tập huấn luyện

Ở bước 2 trong quy trình trên, chúng ta cần đưa nhu cầu thực tế về một bài toán ML. Nghĩa là chúng ta cần xác định rõ hàm cần học $y^* : \mathcal{X} \rightarrow \mathcal{Y}$ có đặc trưng cơ bản ra sao về đầu vào và đầu ra. Nói cách khác, ta cần xác định được không gian dữ liệu \mathcal{X} và tập nhãn \mathcal{Y} . Ví dụ:

- Bài toán lọc thư rác: \mathcal{X} là không gian tất cả thư điện tử (emails); $\mathcal{Y} = \{0, 1\}$, trong đó nhãn 1 nghĩa là rác, 0 là thư thường
- Bài toán phân biệt loài vật: \mathcal{X} là không gian tất cả những bức ảnh RGB có kích cỡ 1024×1024 , $\mathcal{Y} = \{\text{dog}, \text{cat}, \text{elephant}\}$

Đối với việc thu thập dữ liệu: ta cần thiết kế kỹ lưỡng bởi dữ liệu thu thập được sẽ dùng để huấn luyện hệ thống. Chất lượng (hay hiệu năng) của một hệ thống phụ thuộc rất lớn vào tập dữ liệu này. Để hệ thống có hiệu quả cao, tập huấn luyện cần chứa những mẫu mà có thể mô tả được nhiều đặc trưng cốt lõi của không gian \mathcal{X} . Nghĩa là, tập dữ liệu này cần có tính đại diện cao cho không gian \mathcal{X} . Ngược lại, tập dữ liệu đó có thể chứa quá ít thông tin về \mathcal{X} và sẽ làm cho hệ thống làm việc kém với những mẫu dữ liệu trong tương lai.

1.2.2 Giai đoạn Mô hình hoá

Ở giai đoạn mô hình hoá (Modeling), có hai bước quan trọng cần làm: chọn kiểu mô hình và giải thuật học.

a) Chọn kiểu mô hình:

Vì hàm y^* thường không biết và ta chỉ thu thập được tập dữ liệu \mathcal{D} , nên một cách phổ biến trong ML là chọn một kiểu (lớp) mô hình \mathcal{H} để xấp xỉ hàm y^* . Lớp mô hình này thường ở dạng đã biết và dễ làm việc. Ví dụ, mô hình tuyến tính, mạng nơron, cây quyết định, ...

Như vậy \mathcal{H} là một lớp các hàm, mà mỗi hàm còn được gọi là *giả thuyết* (hypothesis). Khi chọn \mathcal{H} , ta đã ngầm giả thuyết rằng trong \mathcal{H} sẽ chứa một hàm f mà có thể xấp xỉ tốt y^* tại mọi điểm, nghĩa là $f(\mathbf{x}) \approx y^*(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.

b) Chọn giải thuật học:

Tiếp theo ta cần chọn một **giải thuật học** A nào đó để giúp ta tìm ra một hàm $f \in \mathcal{H}$, dựa vào tập học \mathcal{D} . Nói cách khác, một giải thuật học A sẽ trả về một hàm f khi cho trước tập học \mathcal{D} và lớp mô hình \mathcal{H} . Khi đó ta có thể viết $f = A(\mathcal{H}, \mathcal{D})$. Hàm f này đôi khi còn được gọi là *mô hình đã được huấn luyện* (trained model).

Trong ML, mỗi kiểu mô hình khác nhau có thể có phương pháp học khác nhau. Ví dụ phương pháp Lan truyền ngược (backpropagation) sẽ huấn luyện một mạng nơron, phương pháp ID3 sẽ huấn luyện một cây quyết định, ... Chúng ta sẽ làm quen với một vài giải thuật học ở các chương tiếp theo.

1.3 Vài vấn đề cần biết khi dùng ML

Trong phần này, chúng ta sẽ bàn luận đến mục tiêu quan trọng của ML và một số vấn đề liên quan đến việc học.

1.3.1 Thuật toán học (Learning algorithm)

Đối với nhiều mô hình học máy, một thuật toán học A có thể ở dạng lặp. Mỗi bước lặp có thể tìm ra một hàm nào đó trong \mathcal{H} , bước lặp sau cố gắng cải thiện bước lặp trước. Tuy nhiên nếu không cẩn thận thì thuật toán có thể không hội tụ tới một hàm tốt. Khi đó mô hình chưa được huấn luyện tốt.

Đối với một bài toán học đã cho, có nhiều thuật toán học khác nhau để lựa chọn. Ví dụ, có nhiều giải thuật khác nhau để huấn luyện mạng nơron [Goodfellow et al., 2016]. Mỗi thuật toán khác nhau có thể cho kết quả khác nhau. Đó là đặc điểm quan trọng trong ML. Lúc đó, một câu hỏi là thuật toán nào sẽ tốt nhất đối với bài toán ta đang có?

Định lý "No Free Lunch" [Wolpert and Macready, 1997] nói rằng *nếu một thuật toán hiệu quả đối với một lớp bài toán này thì nó sẽ trả giá về hiệu quả đối với tập các bài toán còn lại.*

Định lý này gợi rằng không có thuật toán nào luôn hiệu quả nhất trên mọi miền ứng dụng. Đây là một đặc trưng quan trọng trong ML và phân tích dữ liệu. Do đó khi đối diện với mỗi bài toán, chúng ta cần lựa chọn một giải thuật học (và mô hình) phù hợp với bài toán đó.

1.3.2 Tập huấn luyện (Training set)

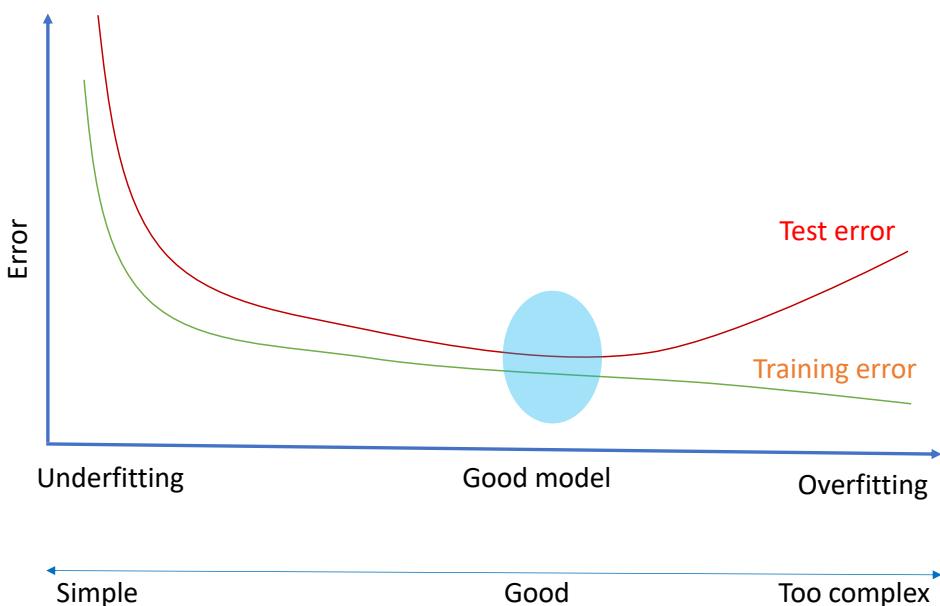
Tập huấn luyện là nơi chứa tri thức để cho máy học và quyết định chất lượng tri thức của máy. Một tập học tốt có thể cung cấp nhiều tri thức hữu ích cho máy. Do đó tính đại diện của các mẫu dữ liệu trong tập học đóng vai trò quan trọng.

Trong ML có một đặc trưng thường thấy là các mô hình có thể có chất lượng càng tốt hơn nếu được huấn luyện từ nhiều mẫu dữ liệu hơn. Nói cách khác, việc tăng kích cỡ tập học lên thường giúp mô hình có thể tốt hơn. Tuy nhiên, điều này sẽ dẫn đến một câu hỏi là "kích cỡ tập huấn luyện như thế nào là đủ?" Việc trả lời câu hỏi này sẽ cần một số kiến thức về lý thuyết học [Mohri et al., 2018].

Trong nhiều miền ứng dụng, dữ liệu thường bị nhiễu, lỗi, thiếu, ... Những yếu tố này thường ảnh hưởng mạnh mẽ đến hiệu quả của các mô hình ML. Nếu bị nhiễu nhiều, tập học có thể tạo ra hướng dẫn sai lệch cho giải thuật học, và do đó tri thức học được có thể không tốt. Chúng ta cần tìm cách vượt qua những tình huống khó này.

1.3.3 Khả năng tổng quát hóa (Generalization)

Một trong những mục tiêu quan trọng nhất của các mô hình ML là **Khả năng tổng quát hóa (Generalization)**, tức là sau khi học, mô hình có thể làm việc (phán đoán) tốt với những mẫu dữ liệu sẽ gặp trong tương lai. Như vậy khi có khả năng tổng quát hóa cao, mô hình có thể làm việc hiệu quả đối với những mẫu dữ liệu không có trong



Hình 5: Minh họa về quá khớp (Overfitting) và kém khớp (Underfitting). "Test error" mô tả lỗi phán đoán của mô hình đối với những mẫu dữ liệu không có mặt trong quá trình huấn luyện mô hình ấy.

tập huấn luyện. Những mô hình như thế sẽ giúp những hệ thống ML mang lại lợi ích cho nhiều ứng dụng thực tế.

Ngược lại, nếu khả năng tổng quát hoá kém, mô hình ML có thể đang bị vướng phải một số vấn đề, chẳng hạn Quá khớp hoặc kém khớp. Trong những trường hợp đó, chúng có thể không có ích.

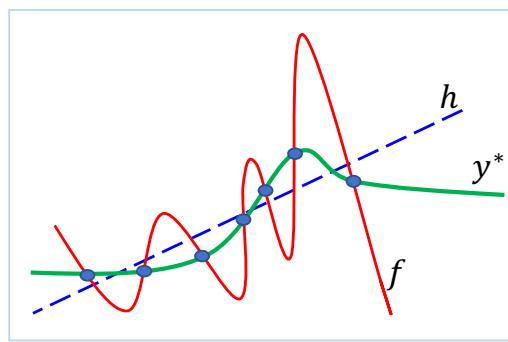
1.3.4 Quá khớp (Overfitting) và kém khớp (Underfitting)

Đôi khi trong thực tế chúng ta có thể gặp phải tình huống: một mô hình đạt độ chính xác cao trên tập huấn luyện D , nhưng phán đoán kém chính xác với các mẫu dữ liệu bên ngoài D . Tình huống này thường được gọi là **Quá khớp (Overfitting)**.

Theo Mitchell [1997], một hàm f được gọi là quá khớp nếu tồn tại hàm g sao cho: g có thể tệ hơn f khi phán đoán các mẫu huấn luyện D , nhưng g tốt hơn f đối với các mẫu dữ liệu ngoài D .

Hình 5 mô tả một vài tình huống có thể gặp khi dùng một mô hình ML. Có nhiều nguyên nhân dẫn đến vấn đề quá khớp. Chẳng hạn,

- *Mô hình quá phức tạp:* nghĩa là mỗi hàm trong lớp mô hình \mathcal{H} quá phức tạp. Khi đó có nhiều hàm $f \in \mathcal{H}$ dễ dàng khớp hoàn hảo với một tập huấn luyện. Hình 6 chứa một ví dụ về một hàm f phức tạp (màu đỏ) mà có thể xấp xỉ tập học một cách hoàn hảo. Nhưng f xấp xỉ y^* rất kém về tổng thể.



Hình 6: Ví dụ về một số hàm khi xấp xỉ một tập dữ liệu (các dấu chấm). y^* là hàm đúng (ground truth), nhưng ta thường không biết.

- *Nhiều hoặc lỗi trong dữ liệu*: nhiều hoặc lỗi có thể làm sai lệch thông tin ẩn chứa trong các mẫu dữ liệu. Do đó thuật toán học dễ bị đánh lừa và tìm ra hàm không tốt trong \mathcal{H} .
- *Tập học quá bé*: khi đó một mô hình dễ dàng có thể xấp xỉ tập học một cách hoàn hảo. Tuy nhiên do kích cỡ bé nên tập dữ liệu này chưa chắc đã đại diện tốt cho toàn bộ không gian dữ liệu \mathcal{X} và đặc trưng của hàm y^* . Nghĩa là tập học đó có thể cung cấp rất ít tri thức về \mathcal{X} và hàm y^* .

Một tình huống khác là **Kém khớp** (Underfitting). Khi đó một hàm kém khớp h thường có lỗi phán đoán lớn (về mặt trung bình) cho bất kỳ mẫu dữ liệu nào, ngay cả những mẫu đã học. Hình 5 minh họa tình huống kém khớp. Như vậy h đã không được huấn luyện đến nơi đến chốn hoặc là hàm quá đơn giản so với y^* .

Trường hợp kém khớp có thể diễn ra nếu chúng ta dùng một kiểu mô hình \mathcal{H} quá đơn giản để giải quyết một bài toán phức tạp. Khi đó bất kỳ hàm nào trong \mathcal{H} đều xấp xỉ y^* kém. Xét ví dụ hàm y^* trong Hình 6: nếu chúng ta chọn \mathcal{H} là lớp mô hình dạng tuyến tính (các đường thẳng) thì vẫn đề kém khớp sẽ diễn ra. Hàm $h \in \mathcal{H}$ (đường nét đứt, màu xanh) sẽ luôn có độ chính xác kém tại phần lớn các mẫu dữ liệu.

1.3.5 Hiệu chỉnh (regularization)

Chúng ta đã biết rằng khả năng tổng quát hoá cao mà một mục tiêu quan trọng của các hệ thống có khả năng học. Nghĩa là khả năng phán đoán trong tương lai phải có độ chính xác cao. Nhưng nếu quá khớp hay kém khớp diễn ra thì hệ thống không đạt được mục tiêu này. Do đó khi xây dựng những hệ thống học (hay khi huấn luyện một mô hình) thì cần có giải pháp để tránh những vấn đề đó.

Đối với kém khớp, chúng ta có thể lựa chọn kiểu mô hình mới mà có khả năng mạnh hơn, chứa các hàm phức tạp hơn. Ví dụ, thay vì chọn mô hình tuyến tính, chúng ta chọn mô hình cây quyết định hoặc mạng nơron. Ngoài ra, chúng ta cũng cần kiểm tra việc huấn luyện và đảm bảo rằng quá trình huấn luyện đã thành công và tốt.

Đối với quá khớp, một con đường thường được sử dụng trong ML là **Hiệu chỉnh** (regularization). Đây là cách giúp ta giảm bớt vấn đề quá khớp cho các mô hình ML. Cho đến nay, có rất nhiều phương pháp hiệu chỉnh khác nhau đã được đề ra. Nói một cách dễ hiểu thì các phương pháp hiệu chỉnh thường thu hẹp không gian tìm kiếm. Nghĩa là quá trình học sẽ tìm kiếm một hàm f trong một vùng nhỏ $\mathcal{H}' \subset \mathcal{H}$. Mỗi cách hiệu chỉnh sẽ tạo ra vùng \mathcal{H}' khác nhau. Chúng ta sẽ tìm hiểu chi tiết hơn về một số phương pháp hiệu chỉnh ở Chương 2.

1.4 Thu thập và tiền xử lý dữ liệu

Thu thập (data collection) và tiền xử lý dữ liệu (data preprocessing) là hai bước quan trọng trong quá trình xử lý thông tin hiện đại. Việc thu thập dữ liệu đòi hỏi sự tập trung và tổ chức để lấy được thông tin cần thiết từ nhiều nguồn khác nhau, bao gồm cả cơ sở dữ liệu, bảng tính, thiết bị cảm biến, và nguồn thông tin trực tuyến. Quá trình này giúp xây dựng cơ sở dữ liệu đầy đủ và đa dạng, làm nền tảng cho các phân tích và quyết định sau này.

Sau khi dữ liệu được thu thập, quá trình tiền xử lý trở thành bước quan trọng để làm cho dữ liệu trở nên hữu ích và chính xác hơn. Tiền xử lý dữ liệu bao gồm loạt các bước: Làm sạch dữ liệu (data cleaning) (lọc dữ liệu nhiễu, điền giá trị còn thiếu, chuẩn hóa đơn vị đo, và kiểm tra tính nhất quán của dữ liệu), tích hợp dữ liệu (data integration) và biến đổi dữ liệu (data transformation). Mục tiêu là tạo ra một bộ dữ liệu có chất lượng cao, sẵn sàng cho các phân tích và mô hình hóa dữ liệu trong bước khai phá dữ liệu về sau. Quá trình này không chỉ giúp cải thiện độ chính xác của thông tin mà còn làm giảm thời gian và công sức cần thiết cho các công đoạn xử lý dữ liệu tiếp theo. Sự kết hợp chặt chẽ giữa thu thập và tiền xử lý dữ liệu đóng vai trò quan trọng trong việc đảm bảo rằng dữ liệu được sử dụng là đáng tin cậy và có giá trị trong quyết định và phân tích.

1.4.1 Thu thập dữ liệu

Thu thập dữ liệu là quá trình tổng hợp dữ liệu từ nhiều nguồn để tìm kiếm câu trả lời cho các vấn đề nghiên cứu, giải đáp câu hỏi, đánh giá kết quả và dự đoán xu hướng và khả năng xảy ra. Đây là một pha quan trọng trong tất cả các loại nghiên cứu, phân tích và quyết định, bao gồm cả những nghiên cứu trong các ngành khoa học xã hội, doanh nghiệp và chăm sóc sức khỏe. Trong quá trình thu thập dữ liệu, chúng ta phải xác định loại dữ liệu, nguồn dữ liệu và phương pháp đang được sử dụng. Có nhiều phương pháp thu thập dữ liệu khác nhau. Lựa chọn cách thu thập dữ liệu phụ thuộc vào từng bài toán.

Quá trình thu thập dữ liệu, lấy mẫu (sampling) thường xuyên được thực hiện. Lấy mẫu là quá trình chọn lựa một phần của toàn bộ dữ liệu để đại diện cho toàn bộ tập dữ liệu này. Lấy mẫu giúp chi phí và tài nguyên tính toán khi tập dữ liệu là rất lớn. Phương pháp lấy mẫu phải có tính đại diện và hiệu quả. Tính đại diện hướng tới lấy mẫu đại diện cho đặc điểm của quần thể mà chúng ta đang nghiên cứu. Để làm được điều này, mẫu lấy được phải đa dạng và tránh bị thiên lệch. Trong khi đó, tính hiệu quả hướng tới tiết kiệm thời gian, chi phí và nỗ lực so với việc lấy dữ liệu từ toàn bộ tập dữ liệu lớn.

Trong phạm vi bài giảng, chúng ta quan tâm đến các phương pháp thu thập các nguồn dữ liệu có sẵn từ tương tác của người dùng trong hệ thống mà chúng ta xây dựng hoặc nguồn dữ liệu cho phép tiếp cận trực tiếp từ internet.

- **Ghi lịch sử (logging):** Là quá trình ghi lại các sự kiện, hành động hoặc thông tin liên quan trong hệ thống, ứng dụng hoặc môi trường máy tính. Việc thu thập dữ liệu dựa trên logging là một phương pháp phổ biến để nắm bắt và lưu trữ thông tin về hoạt động của hệ thống, người dùng. Đây là một công cụ quan trọng cho việc giám sát, phân tích hiệu suất, và hiểu rõ hành vi của người dùng trong các ứng dụng trực tuyến. Chúng ta cần xác định mục tiêu cụ thể của việc thu thập dữ liệu thông qua logging, đặt ra các điểm mốc quan trọng hoặc sự kiện cần được ghi lại. Ví dụ: có thể là giám sát hiệu suất, theo dõi lỗi, hoặc hiểu hành vi người dùng.
- **Thu thập dữ liệu từ internet (crawling/scraping):** Là quá trình tự động hóa việc trích xuất thông tin từ trang web. Phương pháp này đặc biệt hữu ích khi muốn thu thập dữ liệu từ các trang web công cộng để phân tích, nghiên cứu thị trường, hoặc tạo cơ sở dữ liệu thông tin. Tuy nhiên, cần tuân thủ các quy định và điều kiện của trang web để tránh việc vi phạm quy định về bản quyền và quyền riêng tư. Chúng

ta cần xác định rõ mục tiêu của việc thu thập dữ liệu và xác định loại dữ liệu cần lấy từ trang web, nghiên cứu cấu trúc HTML của trang web để hiểu cách dữ liệu được tổ chức và lưu trữ. Có nhiều công cụ hỗ trợ cho bước thu thập dữ liệu. Chúng ta có thể lựa chọn công cụ phù hợp, ví dụ BeautifulSoup, Scrapy, hoặc Selenium tùy thuộc vào độ phức tạp của trang web và yêu cầu thu thập.

1.4.2 Tiềm xử lý dữ liệu

Dữ liệu thực tế ngày nay rất dễ bị ảnh hưởng bởi nhiều, thiếu sót và không nhất quán do kích thước của chúng thường rất lớn (gigabyte trở lên) và xuất phát từ nhiều nguồn khác nhau, đa dạng. Dữ liệu chất lượng kém sẽ dẫn đến kết quả khai phá dữ liệu kém chất lượng. "Làm thế nào có thể tiềm xử lý dữ liệu để cải thiện chất lượng của dữ liệu và cải thiện chất lượng kết quả khai phá dữ liệu? Làm thế nào có thể tiềm xử lý dữ liệu để cải thiện hiệu suất và dễ dàng trong quá trình khai phá dữ liệu?".

Có nhiều kỹ thuật trong tiềm xử lý dữ liệu. Việc làm sạch dữ liệu (data cleaning) có thể được áp dụng để loại bỏ nhiều và sửa các không nhất quán trong dữ liệu. Tích hợp dữ liệu (data integration) kết hợp dữ liệu từ nhiều nguồn thành một kho dữ liệu nhất quán như một kho dữ liệu. Giảm dữ liệu (data reduction) có thể giảm kích thước dữ liệu bằng cách tổng hợp, loại bỏ các đặc trưng dư thừa hoặc gom cụm. Biến đổi dữ liệu (data transformation) (ví dụ, chuẩn hóa) có thể được áp dụng, trong đó dữ liệu được chia tỷ lệ để nằm trong một khoảng nhỏ hơn, chẳng hạn từ 0.0 đến 1.0. Điều này có thể cải thiện độ chính xác và hiệu suất của các thuật toán khai phá dữ liệu liên quan đến độ đo khoảng cách. Các kỹ thuật này không phải là độc lập lẻ; chúng có thể hoạt động cùng nhau. Ví dụ, việc làm sạch dữ liệu có thể bao gồm các biến đổi để sửa các dữ liệu sai, chẳng hạn như biến đổi tất cả các mục cho một trường ngày thành một định dạng chung.

Làm sạch dữ liệu

Các quy trình làm sạch dữ liệu gồm các bước để "làm sạch" dữ liệu bằng cách điền vào các giá trị thiếu, làm mịn dữ liệu nhiễu, xác định hoặc loại bỏ các giá trị ngoại lệ và giải quyết sự không nhất quán. Nếu dữ liệu có vấn đề, kết quả của việc khai phá dữ liệu khó có thể tin tưởng. Hơn nữa, dữ liệu không sạch có thể tạo ra sự nhầm lẫn trong quy

trình khai phá, dẫn đến kết quả không đáng tin cậy. Mặc dù hầu hết các quy trình khai phá có một số phương pháp để xử lý dữ liệu không đầy đủ hoặc nhiễu, nhưng chúng không phải lúc nào cũng mạnh mẽ. Do đó, một bước tiền xử lý hữu ích là thực hiện một số quy trình làm sạch dữ liệu.

Điền giá trị thiếu

Vấn đề thiếu giá trị (missing value) trong dữ liệu là một thách thức phổ biến và có nhiều nguyên nhân khác nhau. Dưới đây là một số lý do chính:

- Thu thập dữ liệu không hoàn hảo: Trong quá trình thu thập dữ liệu, có thể xảy ra các lỗi kỹ thuật hoặc hệ thống, dẫn đến việc không nhận được giá trị cho một số quan sát.
- Khách quan không cung cấp thông tin: Trong một số trường hợp, người tham gia nghiên cứu hoặc người nhập dữ liệu có thể từ chối cung cấp một số thông tin hoặc để trống một số trường thông tin.
- Quy trình lỗi: Trong quá trình chuyển dữ liệu giữa các hệ thống hoặc quy trình lưu trữ, có thể xảy ra lỗi dẫn đến việc mất mát dữ liệu.
- Tự nhiên của dữ liệu: Đôi khi, việc thiếu giá trị là do tính tự nhiên của dữ liệu. Ví dụ, một biểu đồ nhiệt độ có thể không có giá trị cho một ngày nào đó do thiên nhiên không thu thập dữ liệu vào thời điểm đó.
- Thiếu chủ ý: Đôi khi, người sử dụng có thể chủ ý bỏ trống một số trường thông tin, đặc biệt là khi họ không muốn tiết lộ thông tin cá nhân hoặc cảm thấy không thoải mái.

Việc giải quyết vấn đề thiếu giá trị là một phần quan trọng của quá trình tiền xử lý dữ liệu. Một số phương pháp phổ biến được liệt kê như sau:

- Bỏ qua bản ghi: Thường được thực hiện khi nhãn lớp bị thiếu (giả sử nhiệm vụ khai phá dữ liệu liên quan đến phân loại). Phương pháp này sử dụng khi bản ghi chứa nhiều thuộc tính có giá trị bị thiếu. Bằng cách bỏ qua bản ghi, chúng ta không sử dụng giá trị của các thuộc tính còn lại trong bản ghi.
- Điền giá trị thiếu bằng cách thủ công: Phương pháp này mất thời gian và có thể không khả thi đối với một bộ dữ liệu lớn với nhiều giá trị thiếu.

- Sử dụng hằng số toàn cục để điền giá trị thiếu: Thay thế tất cả các giá trị thuộc tính bị thiếu bằng một hằng số chung như một nhãn "Unknown" hoặc -1. Nếu giá trị bị thiếu được thay thế bằng "Unknown," chương trình khai phá dữ liệu có thể lầm tưởng rằng chúng tạo thành một khái niệm thú vị, vì chúng có một giá trị chung là "Unknown." Do đó, mặc dù phương pháp này đơn giản, nhưng không hoàn toàn đảm bảo kết quả thu được sẽ tốt.
- Sử dụng một đại lượng trung bình cho thuộc tính (ví dụ: giá trị trung bình (mean) hoặc trung vị (median)): Đối với phân phối dữ liệu bình thường (đối xứng), có thể sử dụng giá trị trung bình, trong khi phân phối dữ liệu lệch nên sử dụng trung vị.
- Sử dụng giá trị trung bình hoặc trung vị của thuộc tính cho tất cả các mẫu thuộc cùng một lớp: Ví dụ, nếu phân loại khách hàng theo rủi ro tín dụng, chúng ta có thể thay thế giá trị thiếu bằng giá trị thu nhập trung bình cho khách hàng thuộc cùng một loại rủi ro tín dụng với bản ghi đã cho. Nếu phân phối dữ liệu cho một lớp cụ thể là lệch, giá trị trung vị là một lựa chọn tốt hơn.
- Sử dụng giá trị có xác suất cao nhất để điền vào giá trị thiếu: Điều này có thể được xác định thông qua hồi quy, các công cụ dự đoán dựa trên suy diễn sử dụng mô hình xác suất Bayesian, hoặc quá trình học cây quyết định. Các phương pháp này sẽ được giới thiệu trong các chương tiếp theo.

Vấn đề không nhất quán trong dữ liệu

Sự không nhất quán trong cách lưu trữ dữ liệu, đơn vị đo lường và các chỉ số có thể dẫn đến sự hiểu lầm và sai sót trong quá trình phân tích. Một số nguyên nhân của sự không nhất quán dữ liệu:

- Đơn vị đo lường và thang đo: sử dụng các biểu diễn thang đo không nhất quán. Ví dụ, sử dụng cả "1, 2, 3" và "A, B, C" để biểu diễn đánh giá có thể tạo ra sự nhầm lẫn. Quan trọng là phải chuẩn hóa thang điểm số để đảm bảo tính nhất quán.
- Định dạng ngày tháng: Ngày tháng thường được biểu diễn dưới nhiều định dạng khác nhau, nhưng để phân tích, cần có một định dạng chuẩn nhất. Các hệ thống khác nhau có thể chấp nhận ngày tháng trong các định dạng khác nhau, gây khó khăn khi tích hợp dữ liệu.

- Không đồng nhất ngôn ngữ: Sự không nhất quán trong cách sử dụng ngôn ngữ và thuật ngữ có thể gây hiểu lầm và khó khăn trong việc đồng thuận ý nghĩa của dữ liệu.

Để giải quyết vấn đề không nhất quán dữ liệu, chúng ta phải phát hiện, thiết lập và thực hiện các tiêu chuẩn về định dạng, đơn vị đo lường, và ngôn ngữ.

Loại dữ liệu nhiễu

Lỗi và nhiễu trong dữ liệu có thể xuất hiện vì nhiều lý do khác nhau, và chúng thường làm ảnh hưởng đến chất lượng của quá trình phân tích dữ liệu và kết quả của mô hình. Một số nguyên nhân được liệt kê cụ thể như sau: Đầu tiên, thu thập dữ liệu không chính xác. Nếu quá trình thu thập dữ liệu không được thực hiện đúng cách, có thể xuất hiện lỗi và nhiễu. Các lỗi này có thể bao gồm sai sót trong quy trình đo lường, nhập liệu sai, hoặc thiết bị đo lường không chính xác. Nguồn gốc đa dạng của dữ liệu: Khi dữ liệu được tổng hợp từ nhiều nguồn, đặc biệt là từ các nguồn không đồng nhất, nhiễu có thể xuất hiện do sự không nhất quán giữa các nguồn. Tính không ổn định của môi trường: Trong một số trường hợp, dữ liệu có thể bị ảnh hưởng bởi những yếu tố không kiểm soát được trong môi trường, như nhiễu điện từ, độ rung, hoặc những sự cố tự nhiên. Không chắc chắn và không chính xác trong đo lường: Một số biến đo không chắc chắn và có độ chính xác hạn chế. Điều này có thể dẫn đến sự xuất hiện của nhiễu trong dữ liệu. Nhiều ngẫu nhiên: Nhiều có thể xuất hiện do sự ngẫu nhiên trong dữ liệu, không có nguyên nhân rõ ràng. Điều này thường xảy ra trong quá trình đo lường và ghi lại dữ liệu. Outliers và dữ liệu sai lệch: Các giá trị ngoại lệ, hay các điểm dữ liệu lạc quan, có thể tạo ra nhiễu nếu chúng không phản ánh đúng tính chất của phân phối dữ liệu chung. Kiểu lỗi trong quá trình xử lý dữ liệu: Trong quá trình xử lý dữ liệu, có thể xảy ra các lỗi như làm mất mát thông tin, sai sót trong tính toán, hoặc không chính xác trong quá trình chuyển đổi dữ liệu.

Vài kỹ thuật làm mịn, loại nhiễu gồm:

- Phân khoảng (Binning): Sắp xếp dữ liệu và phân thành các "khoảng" (equi-depth bins). Sau đó, thực hiện việc làm mịn bằng giá trị trung bình (smooth by bin means), làm mịn bằng trung vị (smooth by bin median), làm mịn bằng biên (smooth by bin boundaries). Có 2 cách chia khoảng chính: Chia thành các khoảng với độ rộng bằng nhau (Equal-width partitioning) và Chia thành các khoảng với độ sâu

bằng nhau (Equal-depth partitioning). Chia thành các khoảng với độ rộng bằng nhau (Equal-width partitioning) thực hiện chia khoảng giá trị thành N khoảng có kích thước bằng nhau: lưới đồng đều. Nếu A và B lần lượt là giá trị thấp nhất và cao nhất của thuộc tính, độ rộng của các khoảng sẽ là: $W = \frac{(B-A)}{N}$. Chia thành các khoảng với độ sâu bằng nhau (Equal-depth partitioning): Phương pháp này chia khoảng giá trị thành N khoảng, mỗi khoảng chứa gần như cùng số lượng mẫu.

- **Hồi quy:** Làm mịn dữ liệu cũng có thể được thực hiện thông qua phương pháp hồi quy. Hồi quy tuyến tính liên quan đến việc tìm ra đường "tốt nhất" để khớp hai thuộc tính (hoặc biến) sao cho một thuộc tính có thể được sử dụng để dự đoán thuộc tính kia. Khi có nhiều trong dữ liệu, có thể sử dụng mô hình hồi quy tuyến tính để tìm ra một đường tuyến tính (đường hồi quy) mà dữ liệu "gần nhất" và giảm thiểu sai số. Sau khi học mô hình hồi quy, chúng ta so sánh giữa giá trị thực tế và giá trị dự đoán để đánh giá mức độ nhiễu. Các điểm dữ liệu có thể được coi là nhiễu nếu chúng có sai số lớn. Từ đó, chúng ta có thể loại bỏ nhiễu hoặc sử dụng giá trị của mô hình dự đoán ra để thay thế giá trị sai. Các kỹ thuật hồi quy sẽ được học trong các chương sau.
- **Phân cụm loại nhiễu:** Phân cụm là một phương pháp mạnh mẽ để loại bỏ nhiễu trong dữ liệu. Phân cụm giúp nhóm các điểm dữ liệu tương tự vào các cụm và xác định các điểm ngoại lai (outliers). Các điểm ngoại lai thường là những điểm dữ liệu không thuộc vào bất kỳ cụm nào hoặc thuộc vào cụm với sự khác biệt lớn. Các kỹ thuật phân cụm sẽ được trình bày trong các chương sau.

Tích hợp dữ liệu

Khai phá dữ liệu thường đòi hỏi tích hợp dữ liệu - sự hợp nhất dữ liệu từ nhiều nguồn dữ liệu khác nhau. Việc tích hợp cẩn thận có thể giúp giảm và tránh dư thừa và không nhất quán trong tập dữ liệu kết quả. Điều này có thể giúp cải thiện độ chính xác và tốc độ của quy trình khai phá dữ liệu tiếp theo.

Sự không đồng nhất về ý nghĩa (semantic heterogeneity) và cấu trúc dữ liệu tạo ra những thách thức lớn trong việc tích hợp dữ liệu. Trong quá trình tích hợp dữ liệu, chúng ta phải giải quyết các vấn đề: Xác định thực thể (entity identification), phát hiện

và giải quyết vấn đề xung đột giá trị (data value conflicts), giải quyết vấn đề dư thừa (redundant data) trong khi tích hợp.

Xác định thực thể: Cùng một thực thể có thể được đặt tên khác nhau. Làm thế nào chúng ta có thể tìm được các thực thể mà cùng định danh cho một đối tượng duy nhất trong thế giới thực từ nhiều nguồn dữ liệu khác nhau? Ví dụ, làm thế nào người phân tích dữ liệu hoặc máy tính có thể đảm bảo rằng mã khách hàng trong một cơ sở dữ liệu và số khách hàng trong cơ sở dữ liệu khác đều tham chiếu đến cùng một thuộc tính?

Xác định và giải quyết xung đột: Tích hợp dữ liệu cũng liên quan đến việc phát hiện và giải quyết xung đột giá trị dữ liệu. Ví dụ, đối với cùng một thực thể trong thế giới thực, giá trị thuộc tính từ các nguồn khác nhau có thể khác nhau. Điều này có thể xuất phát từ sự khác biệt trong biểu diễn, tỷ lệ hoặc mã hóa. Ví dụ, một thuộc tính cân nặng có thể được lưu trữ trong đơn vị mét trong một hệ thống và đơn vị Anh quốc trong hệ thống khác.

Vấn đề dư thừa dữ liệu: Sự dư thừa là một vấn đề quan trọng khác trong tích hợp dữ liệu. Một thuộc tính có thể trở nên dư thừa nếu nó có thể "tính toán" từ một thuộc tính khác hoặc một tập hợp các thuộc tính. Sự không nhất quán trong tên thuộc tính hoặc kích thước cũng có thể gây ra sự dư thừa trong tập dữ liệu kết quả. Một số dư thừa có thể được phát hiện thông qua phân tích tương quan (correlation). Với hai thuộc tính, phân tích này có thể đo lường mức độ mạnh mẽ một thuộc tính gợi ý thuộc tính khác, dựa trên dữ liệu có sẵn. Đối với dữ liệu định danh, chúng ta sử dụng kiểm tra chi-square. Đối với các thuộc tính số, chúng ta có thể sử dụng hệ số tương quan (correlation coefficient) và hiệp phương sai (covariance), cả hai đều đo lường cách giá trị của một thuộc tính biến đổi so với giá trị của một thuộc tính khác.

Biến đổi dữ liệu

Trong quá trình biến đổi dữ liệu, dữ liệu được chuyển đổi hoặc tổng hợp thành các dạng phù hợp để khai phá. Các chiến lược cho quá trình biến đổi dữ liệu được trình bày dưới đây:

- **Làm mịn (smoothing):** giúp loại bỏ nhiễu từ dữ liệu. Các kỹ thuật bao gồm phương pháp như binning, hồi quy và phân cụm như trình bày trong phần làm sạch dữ liệu
- **Xây dựng thuộc tính (attribute construction):** Giúp tạo ra các thuộc tính mới để hỗ

trợ quá trình khai thác thông tin, có thể xây dựng các thuộc tính mới từ tập hợp thuộc tính đã cho.

- **Tổng hợp (aggregation):** Giúp tóm lược hoặc tổng hợp cho dữ liệu. Ví dụ: Dữ liệu bán hàng hàng ngày có thể được tổng hợp để tính toán số lượng hàng tháng và hàng năm.
- **Chuẩn hóa (normalization):** đưa các giá trị thuộc tính về một phạm vi nhỏ hơn, chia tỷ lệ dữ liệu để nằm trong một phạm vi như –1.0 đến 1.0 hoặc 0.0 đến 1.0.
- **Rời rạc hóa (discretization):** thay thế giá trị thô của thuộc tính số bằng nhãn khoảng (ví dụ, 0–10, 11–20) hoặc nhãn khái niệm (ví dụ, thanh niên, người lớn, người cao tuổi).
- **Tạo ra hệ thống khái niệm (concept hierarchy generation):** Xây dựng các hệ thống khái niệm cho dữ liệu định danh. Ví dụ, thuộc tính như đường phố có thể được tổng quát hóa thành khái niệm cao cấp như thành phố hoặc quốc gia.

CHƯƠNG 2. HỒI QUI VÀ PHÂN CỤM

2.1 Hồi qui

Trong phần này, chúng ta làm quen với mô hình tuyến tính (linear model) cho bài toán hồi qui, khái niệm hàm măt măt, lõi thực nghiệm, và lõi tổng quát hóa. Sau đó chúng ta sẽ tìm hiểu ba phương pháp huấn luyện mô hình.

2.1.1 Mô hình hồi qui tuyến tính

Chúng ta đã biết rằng bản chất của việc học từ dữ liệu là tìm ra được hàm y^* kết nối giữa đầu vào $\mathbf{x} \in \mathcal{X}$ với một đầu ra $y \in \mathcal{Y}$ nào đó. Ta thường không biết trước hình dạng hay đặc trưng nào cụ thể về y^* , ngoại trừ các mẫu dữ liệu (có nhãn) đã thu thập được.

Đối với bài toán học có giám sát, tập dữ liệu huấn luyện thường có nhãn cho mỗi mẫu dữ liệu. Nếu \mathcal{Y} bao gồm các số thực thì nó được gọi là bài toán hồi qui.

Định nghĩa 4 (Hồi qui cơ bản - Regression problem) Cần tìm hàm $y^* : \mathcal{X} \rightarrow \mathcal{Y}$ từ một tập dữ liệu $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M, y_1, y_2, \dots, y_M\}$ cho trước, trong đó $y_j = y^*(\mathbf{x}_j)$ với mỗi chỉ số $j \in \{1, 2, \dots, M\}$ và $\mathcal{Y} \subseteq \mathbb{R}$.

Hàm y^* lúc này thường được gọi là *hàm hồi qui*.

Để tìm hiểu về mô hình tuyến tính, chúng ta xét trường hợp \mathcal{X} là tập các vectơ trong không gian n chiều, tức là $\mathcal{X} \subseteq \mathbb{R}^n$. Như vậy mỗi điểm $\mathbf{x} \in \mathcal{X}$ có thể được biểu diễn bằng vectơ cột dưới dạng $\mathbf{x} = (x_1, \dots, x_n)^T$. Chú ý rằng:

- Chúng ta đã dùng *n thuộc tính* (attribute) hoặc tín hiệu (signal) hoặc *đặc trưng* (feature) để biểu diễn một mẫu dữ liệu, và $n \geq 1$. Mỗi x_i là một giá trị cụ thể của thuộc tính thứ i .
- Nếu tập dữ liệu ban đầu ở dạng thô, chúng ta có thể thực hiện tiền xử lý (đã được trình bày trong Mục 1.4) để đưa chúng về dạng vectơ.

Mặc dù bài toán hồi qui 4 đã rất cụ thể, nhưng việc giải quyết nó không dễ chút nào. Một lý do quan trọng là việc giải quyết bài toán ấy sẽ cần làm việc với không gian tất cả các hàm có dạng $h : \mathbb{R}^n \rightarrow \mathbb{R}$. Không gian này quá lớn và có vô số chiều. Do đó, đây là một bài toán bất khả thi.

Một cách làm phổ biến trong ML là chọn một dạng hàm (mô hình) cụ thể mà chúng ta có thể dễ dàng làm việc. Ví dụ, chúng ta chỉ xét những hàm ở dạng siêu phẳng (hyperplanes). Khi đó, chúng ta sẽ làm việc với một mô hình tuyến tính.

Mô hình tuyến tính (Linear Model): xét các hàm có dạng

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_nx_n \quad (2)$$

trong đó

- $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ là một vectơ đầu vào.
- $\mathbf{w} = (w_0, w_1, \dots, w_n)^T \in \mathbb{R}^{n+1}$ là vectơ trọng số. Mỗi w_j sẽ đánh trọng số cho thuộc tính x_j tương ứng. w_0 thường được gọi là độ lệch (bias).
- \mathbf{w} còn được gọi là bộ tham số. Mỗi vectơ \mathbf{w} sẽ mô tả một hàm khác nhau.
- Chú ý rằng f là hàm tuyến tính theo biến \mathbf{w} .

Như vậy với việc lựa chọn kiểu mô hình tuyến tính (2), chúng ta đã thu hẹp không gian hàm cần xem xét. Không gian hàm đó là

$$\mathcal{H} = \{f : f(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_nx_n, \mathbf{x} \in \mathbb{R}^n, \mathbf{w} \in \mathbb{R}^{n+1}\} \quad (3)$$

Việc chọn này cũng có nghĩa là chúng ta đã ngầm giả thuyết rằng hàm y^* có thể được xấp xỉ tốt bởi một hàm dạng tuyến tính. Nói cách khác, ta đã giả thuyết rằng tồn tại một hàm $f^* \in \mathcal{H}$ sao cho f^* xấp xỉ tốt y^* tại mọi điểm.

Sau khi đã chọn kiểu mô hình tuyến tính, việc còn lại là dùng một giải thuật học để tìm ra một hàm tốt nhất trong \mathcal{H} , dựa vào một tập học \mathbf{D} cho trước. Một câu hỏi đặt ra là *Làm thế nào để biết được một hàm f tốt hơn hàm g ?* Một giải thuật học sẽ cần trả lời câu hỏi này để tìm ra một hàm tốt nhất có thể. Đối với mô hình tuyến tính (2), về bản chất giải thuật học sẽ cần tìm một vectơ trọng số \mathbf{w} tốt nhất.

2.1.2 Hàm mất mát (loss function)

Để so sánh hai hàm khác nhau, chúng ta cần chọn một độ đo cụ thể. Một cách thường dùng trong ML là xem xét lỗi phán đoán. Giả sử y là giá trị đúng và \hat{y} là giá trị phán

đoán. Khi đó ta có thể tính lỗi phán đoán như sau:

$$c(y, \hat{y}) = (y - \hat{y})^2 \quad (\text{Square loss})$$

Đại lượng này đôi khi còn được gọi là *mất mát* (loss), hoặc *rủi ro* (risk). Dạng hàm này thường được gọi là *lỗi bình phương* (square loss). Ngoài ra, người ta có thể dùng một số dạng hàm lỗi khác. Chẳng hạn:

$$\begin{aligned} c(y, \hat{y}) &= |y - \hat{y}| && (\text{Absolute loss, } L_1 \text{ loss}) \\ c(y, \hat{y}) &= \max\{0, 1 - y\hat{y}\} && (\text{Hinge loss}) \\ c(y, \hat{y}) &= y \log \hat{y} && (\text{Cross-entropy loss}) \end{aligned}$$

Xét một mẫu dữ liệu \mathbf{x} với đầu ra đúng là y . *Lỗi (error)* phán đoán của hàm $f \in \mathcal{H}$ cho mẫu này có thể đo như sau (sử dụng dạng lỗi bình phương):

$$\ell(f, \mathbf{x}) = [y - f(\mathbf{x}; \mathbf{w})]^2 \quad (4)$$

Định nghĩa 5 *Lỗi trung bình* (*expected loss*) của f trên toàn bộ không gian dữ liệu là $\mathbb{E}_{(\mathbf{x}, y)}[\ell(f, \mathbf{x})]$.

Như vậy chúng ta có thể dựa vào lỗi trung bình để so sánh khả năng của hai hàm nào đó trong \mathcal{H} . Mục tiêu cốt lõi của việc học là cần tìm một hàm

$$f^* = \arg \min_{f \in \mathcal{H}} \mathbb{E}_{(\mathbf{x}, y)}[\ell(f, \mathbf{x})] \quad (5)$$

Nghĩa là f^* có lỗi trung bình bé nhất so với tất cả các hàm khác trong \mathcal{H} .

Rõ ràng f^* là hàm tốt nhất trong \mathcal{H} . Tuy nhiên việc tìm ra hàm này không dễ chút nào, bởi việc tính kỳ vọng liên quan đến toàn bộ không gian dữ liệu. Trong thực tế chúng ta chỉ có thể thu thập được một tập dữ liệu \mathbf{D} hữu hạn. Do đó chúng ta thường quan tâm đến lỗi trên tập này:

$$RSS(f) = \sum_{i=1}^M \ell(f, \mathbf{x}_i) = \sum_{i=1}^M (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 \quad (6)$$

Đại lượng $RSS(f)$ thường được gọi là *tổng phần dư bình phương* (residual sum of

squares). Ở đây chúng ta dùng dạng hàm lỗi bình phương để định nghĩa RSS. Các dạng hàm lỗi khác có thể dùng để thay thế.

Định nghĩa 6 *Lỗi thực nghiệm (empirical loss) của hàm f trên tập \mathbf{D} được định nghĩa là* $L(f, \mathbf{D}) = \frac{1}{M} \sum_{i=1}^M \ell(f, \mathbf{x}_i)$.

Đây chính là trung bình lỗi trên \mathbf{D} , và có thể tính toán được cụ thể nếu cho trước một tập dữ liệu. Chú ý rằng $L(f, \mathbf{D})$ là một xấp xỉ của lỗi trung bình $\mathbb{E}_{(\mathbf{x}, y)}[\ell(f, \mathbf{x})]$. Sự sai khác giữa hai đại lượng này thường được gọi là **Lỗi tổng quát hoá** (generalization error): $|\mathbb{E}_{(\mathbf{x}, y)}[\ell(f, \mathbf{x})] - L(f, \mathbf{D})|$. Lỗi này càng bé nếu tập \mathbf{D} càng lớn. Một hệ thống có khả năng tổng quát hoá cao thì lỗi tổng quát hoá sẽ nhỏ.

2.1.3 Phương pháp Bình phương tối thiểu

Mặc dù (5) là bài toán cần giải, nhưng chúng ta không thể làm việc trực tiếp. Do đó, một cách đơn giản hơn là chúng ta hãy đi tìm một hàm f mà làm cho lỗi trung bình trên tập huấn luyện \mathbf{D} là nhỏ. Nghĩa là cần tìm

$$f^* = \arg \min_{f \in \mathcal{H}} RSS(f) \quad (7)$$

Đây chính là ý tưởng của phương pháp **Bình phương tối thiểu** (Ordinary least squares - OLS).

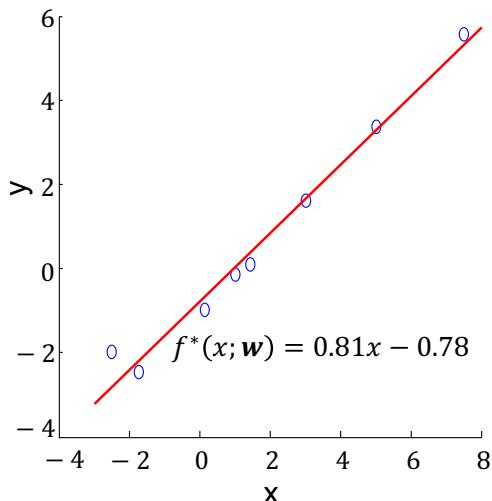
Vì ta đang xét lớp mô hình tuyến tính, nên mỗi hàm f có một vectơ trọng số \mathbf{w} đại diện cho nó. Mỗi vectơ \mathbf{w} sẽ xác định duy nhất một hàm $f(\mathbf{x}; \mathbf{w}) \in \mathcal{H}$. Do đó bài toán (7) có thể đưa về bài toán tương đương sau:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^M (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 = \arg \min_{\mathbf{w}} \sum_{i=1}^M (y_i - w_0 - w_1 x_{i1} - \cdots - w_n x_{in})^2 \quad (8)$$

trong đó $\mathbf{x}_i = (x_{i1}, \dots, x_{in})^T$ là một mẫu dữ liệu trong tập học \mathbf{D} .

Như vậy ta cần tìm \mathbf{w}^* mà là điểm cực tiểu của hàm $RSS(f) = \sum_{i=1}^M (y_i - w_0 - w_1 x_{i1} - \cdots - w_n x_{in})^2$. Chú ý rằng đây là một hàm dạng bình phương của \mathbf{w} và là hàm lỗi. Do đó những điểm mà có đạo hàm 0 sẽ là điểm cực tiểu. Như vậy để giải bài toán (8), ta có thể tính đạo hàm và giải hệ phương trình $\frac{\partial RSS}{\partial \mathbf{w}} = 0$. Sau khi giải hệ phương trình này, ta

x	y
0.13	-1.00
1.02	-0.17
3.00	1.61
-2.50	-2.00
1.44	0.10
5.00	3.36
-1.74	-2.46
7.50	5.56



Bảng 1: Tập học.

Hình 7: Hàm học được bồi OLS từ tập dữ liệu ở bên trái.
Các dấu chấm tròn mô tả các mẫu dữ liệu trong tập học.

sẽ thu được nghiệm

$$\mathbf{w}^* = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \quad (9)$$

với giả thuyết rằng tồn tại ma trận nghịch đảo $(\mathbf{A}^T \mathbf{A})^{-1}$, trong đó \mathbf{A} là ma trận dữ liệu mà mỗi hàng thứ i là $\mathbf{A}_i = (1, x_{i1}, \dots, x_{in})$ và $\mathbf{y} = (y_1, \dots, y_M)^T$.

Hình 7 chứa một ví dụ về hàm học được bồi phương pháp OLS từ một tập học ở bảng phía bên trái. Dễ dàng thấy rằng, OLS tìm ra hàm f^* mà có thể xấp xỉ tập học khá tốt. Đây là hàm có lỗi huấn luyện bé nhất so với các hàm dạng tuyến tính.

Như vậy, ý tưởng của OLS rất tự nhiên và đơn giản. Tuy nhiên, nó còn một số nhược điểm. Chẳng hạn:

- Việc tìm ra công thức nghiệm (9) sẽ không khả thi nếu ma trận $\mathbf{A}^T \mathbf{A}$ không khả nghịch. Ví dụ trường hợp $\det(\mathbf{A}^T \mathbf{A}) = 0$, tức là $\mathbf{A}^T \mathbf{A}$ là ma trận suy biến. Điều này rất có thể xảy ra trong thực tế, chẳng hạn khi một số thuộc tính phụ thuộc với nhau. Hơn nữa, việc tính ma trận nghịch đảo $(\mathbf{A}^T \mathbf{A})^{-1}$ thường tồn kén, đặc biệt là trong trường hợp n lớn (có nhiều thuộc tính).
- Quá khớp có khả năng diễn ra cao. Lý do là vì OLS chỉ tập trung vào việc tối thiểu lỗi trên tập huấn luyện. Tuy nhiên, cách làm này chưa đảm bảo rằng hàm hồi qui tìm được có khả năng phán đoán tốt trong tương lai.

2.1.4 Hồi qui Ridge

Tiếp theo chúng ta sẽ tìm hiểu một ý tưởng rất đơn giản để tránh vấn đề quá khớp trong OLS. Thay vì chỉ tập trung vào tập học, phương pháp *hồi qui Ridge* sẽ tìm một hàm

$$f^* = \arg \min_{f \in \mathcal{H}} RSS(f) + \lambda \|\mathbf{w}\|_2^2 \quad (10)$$

trong đó λ là hệ số phạt (và không âm), $\|\mathbf{w}\|_2 = \sqrt{w_0^2 + w_1^2 + \dots + w_n^2}$ là chuẩn ℓ_2 của vectơ \mathbf{w} .

Đại lượng $\lambda \|\mathbf{w}\|_2^2$ thường được gọi là đại lượng phạt hoặc *hiệu chỉnh (regularization)*. Khi tối thiểu hóa hàm mục tiêu $RSS(f) + \lambda \|\mathbf{w}\|_2^2$, chúng ta đi tìm những hàm f mà có lỗi thực nghiệm nhỏ và vectơ trọng số có $\|\mathbf{w}\|_2$ nhỏ. Điều này hoàn toàn khác với OLS, vì OLS không có ràng buộc cụ thể về độ lớn của trọng số.

Vì ta đang xét lớp mô hình tuyến tính, nên bài toán (10) có thể đưa về bài toán tương đương sau:

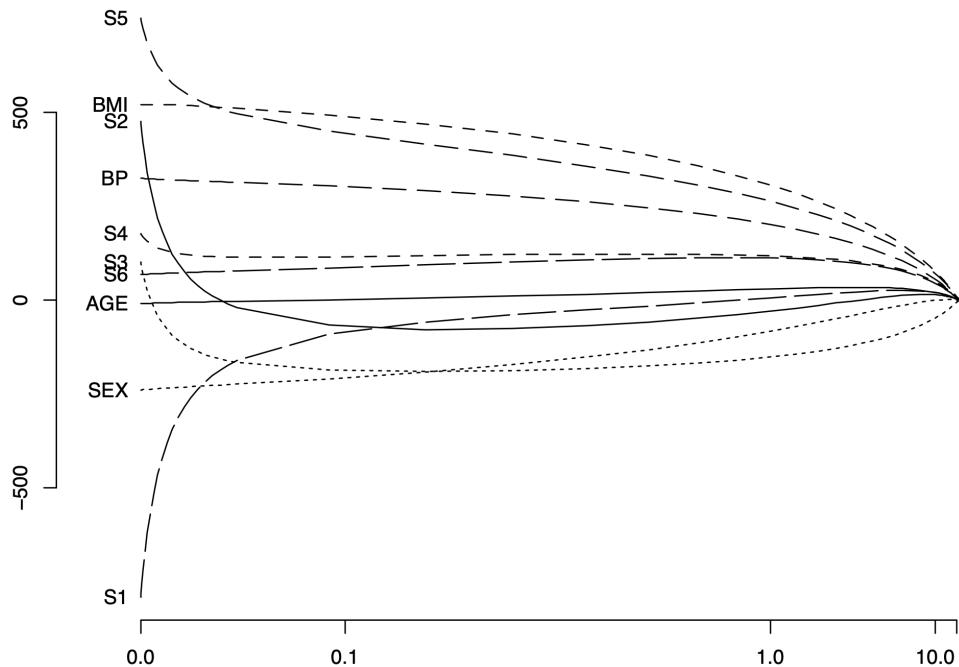
$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^M (y_i - w_0 - w_1 x_{i1} - \dots - w_n x_{in})^2 + \lambda (w_0^2 + w_1^2 + \dots + w_n^2) \quad (11)$$

Việc tìm điểm cực tiểu này có thể làm bằng cách tính đạo hàm của hàm mục tiêu, rồi tìm điểm \mathbf{w}^* làm cho đạo hàm đó bằng 0. Khi đó ta sẽ thu được nghiệm tối ưu

$$\mathbf{w}^* = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}_{n+1})^{-1} \mathbf{A}^T \mathbf{y} \quad (12)$$

trong đó \mathbf{A} là ma trận dữ liệu mà mỗi hàng thứ i là $\mathbf{A}_i = (1, x_{i1}, \dots, x_{in})$, $\mathbf{y} = (y_1, \dots, y_M)^T$, và \mathbf{I}_{n+1} là ma trận đơn vị cỡ $n+1$.

Dễ dàng thấy rằng nếu chọn $\lambda > 0$ thì ma trận $\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}_{n+1}$ luôn khả nghịch. Do đó Ridge không có nhược điểm về giả thuyết khả nghịch như trong OLS. Hệ số phạt này cho phép ta đánh đổi giữa lỗi trên tập học và khả năng tổng quát hoá. Nếu λ lớn thì nghiệm \mathbf{w}^* có xu hướng bé, nhưng lúc đó lỗi trên tập học có xu hướng cao hơn. Hình 8 minh họa một ví dụ. Nếu λ quá lớn thì \mathbf{w}^* sẽ gần 0 và hiện tượng kém khớp có thể diễn ra. Ngược lại nếu λ rất nhỏ thì việc phạt trọng số sẽ ít ý nghĩa và ta chủ yếu tập trung giảm thiểu lỗi huấn luyện. Khi đó quá khớp có thể diễn ra. Như vậy, trong thực tế chúng ta cần chọn hệ số phạt phù hợp cho mỗi bài toán. Hiệu quả của Ridge phụ thuộc lớn



Hình 8: Xu hướng giảm độ lớn của trọng số khi tăng giá trị λ trong Ridge [Hesterberg, 2008]. Đây là kết quả của Ridge trên một tập dữ liệu, trong đó mỗi mẫu được biểu diễn bằng các thuộc tính $\{S1, S2, S3, S4, S5, S6, AGE, SEX, BMI, BP\}$.

vào hệ số này.

Chú ý: Trong thực tế người ta thường thấy rằng Ridge khá nhạy cảm với miền giá trị của đầu ra y . Nghĩa là độ lớn của miền này có thể ảnh hưởng nhiều đến hiệu quả của Ridge. Lý do có thể từ việc phạt độ lệch w_0 . Cho nên nhiều thư viện trong thực tế sẽ bỏ w_0 ra khỏi đại lượng hiệu chỉnh trong (11). Tức là ta tìm cực tiểu của hàm mục tiêu $RSS(f) + \lambda(w_1^2 + \dots + w_n^2)$. Khi đó nghiệm tối ưu có thể thay đổi một chút.

2.1.5 Hồi qui LASSO

Hồi qui Ridge đã dùng chuẩn ℓ_2 làm hiệu chỉnh. Chúng ta có thể thay thế bằng các loại chuẩn khác. Khi dùng chuẩn ℓ_1 thì ta sẽ thu được phương pháp LASSO [Tibshirani, 1996]:

$$f^* = \arg \min_{f \in \mathcal{H}} RSS(f) + \lambda \|\mathbf{w}\|_1 \quad (13)$$

Trong đó $\|\mathbf{w}\|_1 = |w_0| + \dots + |w_n|$ là chuẩn ℓ_1 của vectơ \mathbf{w} . Nói cách khác, chúng ta đi tìm vectơ trọng số mà làm cho hàm mục tiêu $\sum_{i=1}^M (y_i - \mathbf{A}_i \mathbf{w})^2 + \lambda \|\mathbf{w}\|_1$ đạt cực tiểu.

Việc giải bài toán tối ưu (13) không dễ như trong Ridge. Bởi vì hàm mục tiêu $RSS(f) + \lambda \|\mathbf{w}\|_1$ không khả vi tại điểm 0. Do đó, chúng ta cần những phương pháp phức tạp hơn để tìm điểm cực tiểu của hàm mục tiêu này. Độc giả có thể xem một vài

phương pháp được trình bày trong [Hastie et al., 2017].

Tính chất của LASSO

Như vậy đại lượng hiệu chỉnh cũng có vai trò tương tự như trong Ridge là hạn chế độ lớn của vectơ trọng số. λ càng lớn thì mức độ phạt càng mạnh, và do đó độ lớn vectơ trọng số có xu hướng bé dần. Nghĩa là hệ số phạt λ cung cấp một cơ chế đơn giản để thực hiện hiệu chỉnh. Những điều này gợi ý rằng việc giải bài toán (13) tương tự với việc giải bài toán sau

$$\min_{f \in \mathcal{H}} RSS(f) \text{ sao cho } \|\mathbf{w}\|_1 \leq t \quad (\text{LASSO})$$

với t là một hằng số nào đó. Còn đối với Ridge thì sẽ là

$$\min_{f \in \mathcal{H}} RSS(f) \text{ sao cho } \|\mathbf{w}\|_2^2 \leq t \quad (\text{Ridge})$$

Như vậy chúng ta có thể thấy cả hai phương pháp đều cố gắng làm cho lỗi huấn luyện nhỏ, nhưng dùng các ràng buộc khác nhau về vectơ trọng số. Các ràng buộc này tạo ra các miền tìm kiếm khác nhau. Cụ thể hơn, LASSO sẽ tìm một hàm f mà có vectơ trọng số nằm trong miền

$$R_{Lasso} = \{\mathbf{w} : \|\mathbf{w}\|_1 \leq t\}$$

Còn Ridge sẽ tìm trong miền

$$R_{Ridge} = \{\mathbf{w} : \|\mathbf{w}\|_2^2 \leq t\}$$

Với các lựa chọn khác nhau của λ thì kích cỡ của miền ràng buộc R_{Ridge} hoặc R_{Lasso} sẽ khác nhau. Nghĩa là *hệ số phạt cho phép ta thay đổi kích cỡ vùng mà ta tìm kiếm \mathbf{w}^** . Đây là tính chất chung của Ridge và LASSO.

LASSO có một tính chất đặc biệt, đó là "*nghiệm thưa*". Tức là nghiệm \mathbf{w}^* của bài toán (13) thường có nhiều thành phần là 0. Điều này gợi ý rằng hàm hồi qui "có thể" chỉ phụ thuộc vào một số thuộc tính, vì ta có thể viết $f(\mathbf{x}; \mathbf{w}^*) = \sum_{j \in J} w_j^* x_j$, trong đó $J = \{j : w_j^* \neq 0\}$. Nghĩa là bằng cách dùng chuẩn ℓ_1 , LASSO vừa thu hẹp không gian tìm kiếm và vừa lựa chọn một số thuộc tính để đưa vào hàm hồi qui. Các thuộc tính được lựa chọn có thể mang ý nghĩa quan trọng nào đó cho bài toán.

w	0	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	Test RSS
OLS	2.465	0.680	0.263	-0.141	0.210	0.305	-0.288	-0.021	0.267	0.521
Ridge	2.452	0.420	0.238	-0.046	0.162	0.227	0.000	0.040	0.133	0.492
LASSO	2.468	0.533	0.169		0.002	0.094				0.479

Bảng 2: So sánh kết quả của ba phương pháp hồi qui trên một tập dữ liệu về bệnh ung thư tiền liệt tuyến. Sau khi huấn luyện từ một tập học gồm 67 mẫu, chúng ta sẽ thu được vectơ trọng số của hàm hồi qui. Mỗi dòng trong bảng mô tả vectơ trọng số tìm được của phương pháp (bên trái) tương ứng. Kết quả cho thấy LASSO tìm được vectơ thừa, vì nhiều trọng số là 0. Cột bên phải là kết quả phán đoán của các hàm học được cho 30 mẫu dữ liệu kiểm chứng (không nằm trong tập học).

Hiệu quả của OLS, Ridge, và LASSO

Để thấy được hiệu quả thực tế của các phương pháp này, chúng ta sử dụng một tập dữ liệu \mathbf{D}_{train} gồm 67 mẫu để huấn luyện. Đây là một tập dữ liệu về bệnh ung thư tiền liệt tuyến, trong đó mỗi mẫu dữ liệu được biểu diễn bởi 8 thuộc tính (gồm lcavol, lweight, age, lbph, svi, lcp, gleason, pgg45). Bảng 2 chứa vectơ trọng số học được khi sử dụng OLS, Ridge, và LASSO. Kết quả cho thấy các vectơ trọng số thu được khác nhau khá nhiều giữa các phương pháp. Một số trọng số thường có giá trị cao trong các phương pháp. Điều này gợi ý rằng một số thuộc tính tương ứng (ví dụ lcavol, lweight) có thể có vai trò cao trong hàm hồi qui.

Khác với hai phương pháp còn lại, LASSO có thể tìm được nghiệm thừa. Hàm hồi qui lúc này có thể được viết là $f(\mathbf{x}; \mathbf{w}^*) = 2.468 + 0.533x_1 + 0.169x_2 + 0.002x_4 + 0.094x_5$. Mặc dù đây là hàm đơn giản, nhưng khả năng phán đoán của nó vẫn có thể tốt. Để thấy được điều đó, chúng ta dùng một tập dữ liệu gồm 30 mẫu (không nằm trong \mathbf{D}_{train} để kiểm chứng. Kết quả là cả ba phương pháp đều cho lỗi bé (xem cột cuối trong Bảng 2). Dù vậy LASSO có lỗi bé nhất trên tập kiểm chứng này. Trong khi đó, Ridge dùng nhiều thuộc tính hơn trong hàm hồi qui đã học, nhưng lỗi phán đoán vẫn tệ hơn LASSO một chút.

2.2 Phân cụm

Trong phần này, chúng ta làm quen với phân cụm, một trường hợp cụ thể của bài toán học không giám sát. Một số phương pháp phân cụm cơ bản bao gồm phân cụm phân vùng (giải thuật K-means và các biến thể) và phân cụm phân cấp sẽ được trình bày chi tiết.

2.2.1 Giới thiệu bài toán phân cụm

Trong học có giám sát, mỗi mẫu dữ liệu học đi kèm với các nhãn. Do đó tập học có dạng $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, trong đó mỗi cặp thể hiện đầu vào và đầu ra tương ứng của một hàm y^* nào đó. Mỗi mẫu \mathbf{x}_i tương ứng với một nhãn (label) y_i . Nhiệm vụ của học có giám sát là dựa vào tập \mathcal{D} để tìm hàm $f(\mathbf{x})$ mà xấp xỉ tốt hàm y^* . Hàm f sau đó có thể dùng để dự đoán các nhãn/giá trị của các mẫu dữ liệu trong tương lai.

Học không giám sát (Unsupervised learning) xảy ra trong các trường hợp hoặc các ứng dụng mà ta không thu thập được nhãn lớp hoặc đầu ra cho các mẫu dữ liệu. Tập dữ liệu học có dạng $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. Hàm $f(\mathbf{x})$ học được có thể mô tả cấu trúc ẩn, mối liên hệ giữa các thuộc tính đầu vào, mối liên hệ giữa các mẫu dữ liệu, ...

Khái niệm phân cụm (clustering)

Phân cụm (clustering) là một cách để tìm kiếm các cấu trúc như vậy. Nó tổ chức các điểm dữ liệu thành các nhóm tương tự, được gọi là các cụm sao cho các điểm dữ liệu trong cùng một cụm tương tự nhau và các điểm dữ liệu trong các cụm khác nhau thì khác nhau:

- Phương pháp học: Học không giám sát
- Đầu vào: tập huấn luyện không có nhãn.
- Đầu ra: các cụm dữ liệu

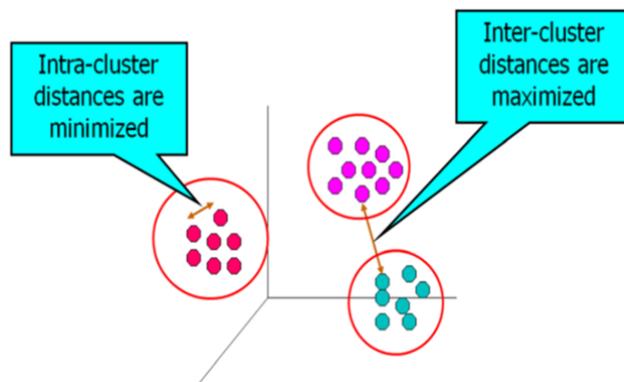
Một cụm (hay cluster) bao gồm các dữ liệu tương tự nhau theo một phép đo hay độ tương tự nào đó. Hai cụm bao gồm dữ liệu khác nhau.

Hình 9 thể hiện điểm dữ liệu gần nhau tạo thành các cụm hay các hình dạng nhất định. Hình vẽ cho thấy các cụm có thể khác nhau về hình dạng, kích thước và mật độ. Sự hiện diện của nhiều trong dữ liệu (ví dụ các điểm nhỏ và cô lập) làm cho việc phát hiện các cụm trở nên khó khăn hơn. Mặc dù con người có thể dễ dàng tìm kiếm các cụm trong không gian hai và có thể cả ba chiều, nhưng chúng ta vẫn cần các giải thuật tự động cho dữ liệu nhiều chiều.

Bài toán phân cụm nhằm gom dữ liệu vào các nhóm/cụm, dựa trên thước đo độ tương tự hay khoảng cách. Thông thường chúng ta mong muốn khoảng cách điểm dữ



Hình 9: Ví dụ về phân cụm dữ liệu



Hình 10: Khoảng cách nội cụm và liên cụm

liệu trong cùng một cụm là nhỏ (hay độ tương tự cao) trong khi khoảng cách giữa các điểm dữ liệu trong các cụm khác nhau là lớn (tham khảo Hình 10).

- Khoảng cách giữa các cụm (inter-cluster): Khoảng cách/sự khác biệt của các điểm dữ liệu giữa hai cụm bất kỳ phải lớn.
- Khoảng cách trong một cụm (intra-cluster): Khoảng cách/sự khác biệt giữa các điểm dữ liệu trong một cụm phải nhỏ.

Đánh giá kết quả phân cụm

Sau khi đã thực hiện phân cụm dữ liệu, một câu hỏi quan trọng là “Làm cách nào có thể đánh giá liệu kết quả phân cụm có tốt hay không?”. Phương án phổ biến là sử dụng các thước đo tính điểm phân cụm. Từ đó có thể so sánh các kết quả phân cụm trên cùng một tập dữ liệu.

Trường hợp chúng ta có sẵn phân cụm trên thực tế đối với tập dữ liệu mẫu, có thể thực hiện pháp đo lường mức độ phù hợp của các cụm tìm ra bằng phương pháp phân cụm với các cụm thực thế đúng này (ground of truth).

Sum-of-Squared Error

Tổng bình phương sai số (SSE) tính bằng tổng bình phương các khoảng cách giữa các điểm dữ liệu và đại diện cụm của chúng (tức là trọng tâm cụm tương ứng). SSE là thước đo đánh giá tính đồng nhất của các kết quả phân cụm đối với các điểm dữ liệu.

SSE sử dụng khoảng cách Euclidean, nhưng có thể áp dụng cho các thước đo khoảng cách khác. Định nghĩa được đưa ra trong Công thức 14, với trọng tâm cụm của cụm C_i được viết tắt là cen_i (center of cluster i).

$$E(C) = \sum_{i=1}^k \sum_{o \in C_i} d(o, cen_i) \quad (14)$$

Giá trị Silhouette

Với phương pháp này, các kết quả phân cụm được đánh giá bằng so sánh các khoảng cách nội cụm (các điểm dữ liệu trong cùng một cụm) và khoảng cách giữa các cụm khác nhau.

Để thu được giá trị Silhouette cho một đối tượng o_i trong cụm C_A , chúng ta so sánh khoảng cách trung bình a giữa o_i và tất cả các đối tượng khác trong C_A với khoảng cách trung bình b giữa o_i và tất cả các đối tượng trong cụm C_B lân cận.

Với mỗi đối tượng $o_i : -1 \leq sil(o_i) \leq 1$. Nếu $sil(o_i)$ lớn thì khoảng cách trung bình của đối tượng trong cụm nhỏ hơn khoảng cách trung bình đến các đối tượng trong cụm lân cận, do đó o_i được phân loại tốt. Nếu $sil(o_i)$ nhỏ thì khoảng cách trung bình của đối tượng trong cụm lớn (tương đối) so với khoảng cách trung bình đến các đối tượng trong cụm lân cận, do đó o_i có thể bị phân cụm sai.

$$a(o_i) = \frac{1}{|C_A| - 1} \sum_{o \in C_A, o \neq o_i} d(o_i, o_j) \quad (15)$$

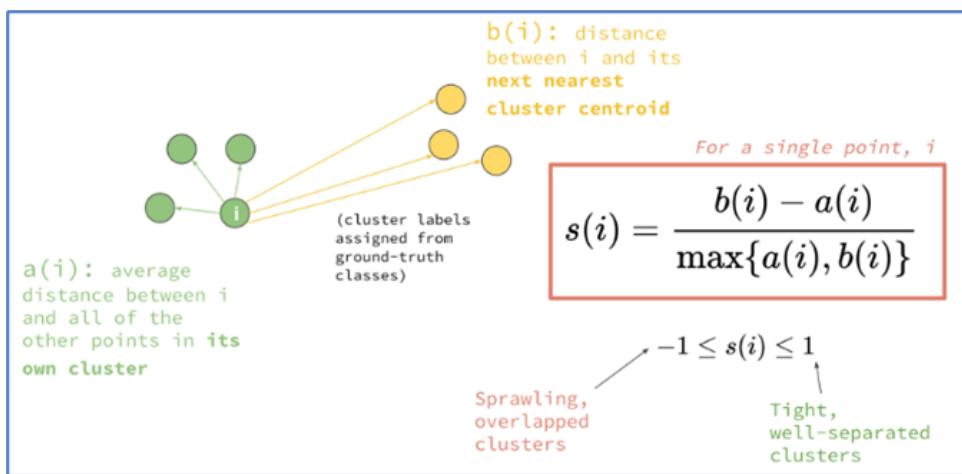
$$b(o_i) = \min_{C_B \neq C_A} \frac{1}{|C_B|} \sum_{o_j \in C_B} d(o_i, o_j) \quad (16)$$

$$sil(o_i) = \frac{b(o_i) - a(o_i)}{\max\{a(o_i), b(o_i)\}} \quad (17)$$

Hình vẽ 11 mô tả trực quan đối với giá trị Silhouette.

So sánh với phân cụm mẫu (thủ công)

Khi có một phân cụm mẫu đúng (ví dụ phân cụm bằng chuyên gia), chúng ta có thể so sánh nó với phân cụm tự động để đánh giá giải thuật phân cụm. Trong trường hợp



Hình 11: Giá trị Silhouette

này, các thước đo trong phần phân loại dữ liệu (để so sánh kết quả phân loại dữ liệu với phân loại mẫu) có thể được sử dụng.

Các phương pháp phân cụm (Han et al. [2023])

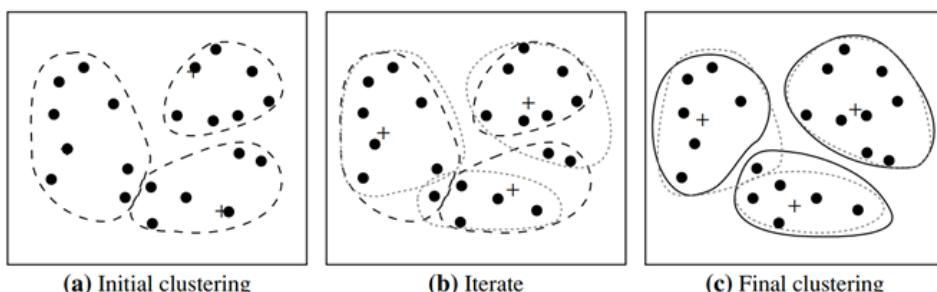
Phương pháp phân vùng (Partition method)

Cho một tập hợp n đối tượng, phương pháp phân vùng sẽ xây dựng k phân vùng dữ liệu, trong đó mỗi phân vùng đại diện cho một cụm và $k \leq n$. Tức là nó chia dữ liệu thành k nhóm sao cho mỗi nhóm chứa ít nhất một đối tượng.

Hầu hết các phương pháp phân vùng đều dựa trên khoảng cách. Cho số lượng k phân vùng, phương pháp phân vùng sẽ tạo ra k phân vùng ban đầu. Sau đó, nó sử dụng kỹ thuật tái định vị, lặp đi lặp lại nhằm cải thiện việc phân vùng bằng cách di chuyển các đối tượng từ nhóm này sang nhóm khác. Tiêu chí chung của một phân vùng tốt là các đối tượng trong cùng một cụm là “gần” hoặc có liên quan với nhau, trong khi các đối tượng trong các cụm khác nhau là “cách xa nhau” hoặc rất khác nhau.

Việc đạt được mức tối ưu toàn cục trong phân cụm dựa trên phân vùng thường bị hạn chế về mặt tính toán do không thể xem xét đầy đủ tất cả các khả năng phân vùng có thể. Thay vào đó, hầu hết các ứng dụng đều áp dụng các phương pháp heuristic phổ biến, chẳng hạn như các phương pháp tham lam như giải thuật k-means và k-medoids, giúp cải thiện dần chất lượng phân cụm và đạt đến mức tối ưu cục bộ. Các phương pháp phân cụm heuristic này hoạt động tốt để tìm các cụm hình cầu (spherical-shaped) và với cơ sở dữ liệu có kích thước vừa và nhỏ.

Hình 12 minh họa phương pháp phân cụm phân vùng. Tóm tắt đặc điểm của phương



Hình 12: Phân cụm phân vùng

pháp phân cụm phân vùng như sau:

- Tìm các cụm hình cầu loại trừ lẫn nhau (mỗi điểm dữ liệu thuộc về một cụm)
- Dựa trên khoảng cách
- Có thể sử dụng giá trị trung bình hoặc medoid để biểu diễn trung tâm cụm
- Hiệu quả đối với các tập dữ liệu có kích thước vừa và nhỏ

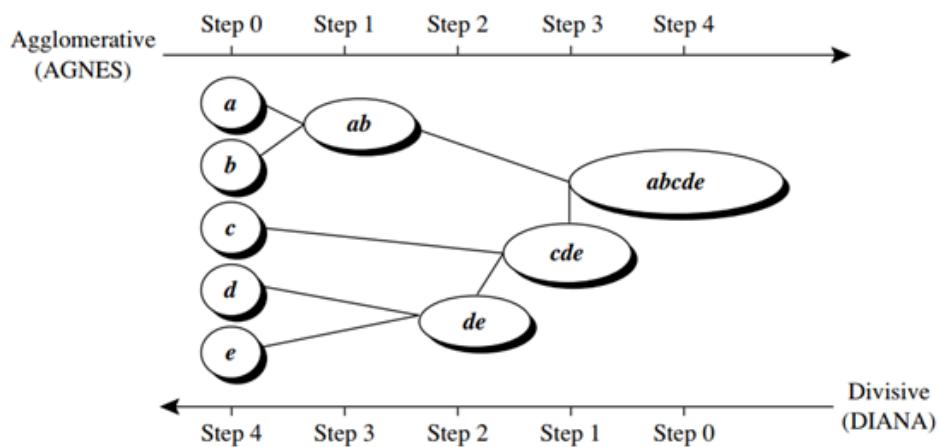
Phương pháp phân cấp (Hierarchical method)

Phương pháp phân cấp tạo ra sự phân rã theo cấp bậc của tập hợp các đối tượng dữ liệu đã cho. Một phương pháp phân cấp có thể được phân loại thành phương pháp agglomerative (tổng hợp) hoặc divisive (phân chia).

Cách tiếp cận tổng hợp, còn được gọi là cách tiếp cận từ dưới lên, bắt đầu bằng việc mỗi đối tượng tạo thành một nhóm riêng biệt. Nó liên tục hợp nhất các đối tượng hoặc nhóm gần nhau, cho đến khi tất cả các nhóm được hợp nhất thành một (cấp cao nhất của hệ thống phân cấp) hoặc đạt đến điều kiện kết thúc.

Cách tiếp cận phân chia, còn được gọi là cách tiếp cận từ trên xuống, bắt đầu với tất cả các đối tượng trong cùng một cụm. Trong mỗi lần lặp liên tiếp, một cụm được chia thành các cụm nhỏ hơn cho đến khi cuối cùng mỗi đối tượng nằm trong một cụm hoặc đạt đến điều kiện kết thúc.

Các phương pháp phân cụm phân cấp có thể dựa trên khoảng cách (distance-based) hoặc dựa trên mật độ và tính liên tục (density-/continuity-based). Các phương pháp phân cấp có một đặc điểm là là một khi một bước (hợp nhất hoặc chia tách) được thực hiện thì không thể quay ngược lại được. Cơ chế này giúp chi phí tính toán nhỏ hơn do hạn chế về số lượng tổ hợp các lựa chọn khác nhau, tuy nhiên cũng hạn chế về chất lượng phân cụm (ví dụ không thể sửa chữa những quyết định sai lầm).



Hình 13: Phân cụm phân cấp

Hình 13 minh họa phương pháp phân cụm phân cấp. Các đặc điểm của phương pháp phân cụm phân cấp có thể tóm tắt như sau:

- Phân rã theo thứ bậc (tức là nhiều cấp độ)
- Có thể dựa trên khoảng cách hoặc mật độ
- Không thể sửa lỗi sáp nhập hoặc phân chia

Phương pháp dựa trên mật độ (Density-based method)

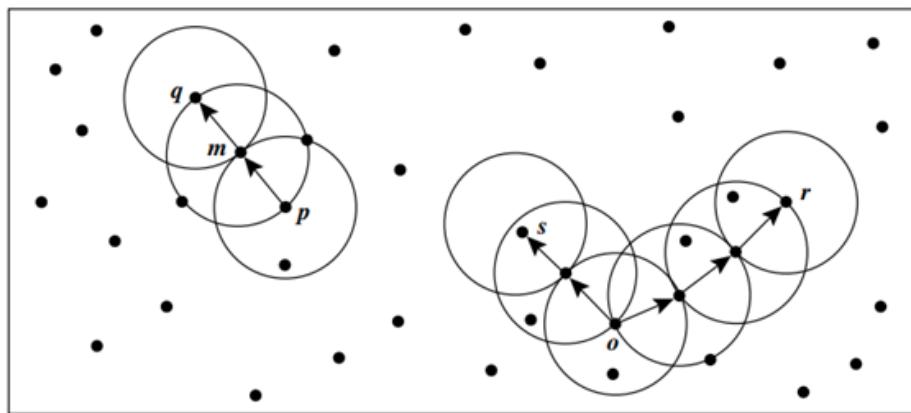
Hầu hết các phương pháp phân vùng đều phân cụm các đối tượng dựa trên khoảng cách giữa các đối tượng. Các phương pháp như vậy chỉ có thể tìm được các cụm có dạng hình cầu và gặp khó khăn trong việc khám phá các cụm có hình dạng tùy ý. Phương pháp phân cụm dựa trên mật độ có thể khắc phục vấn đề này.

Ý tưởng chung của phương pháp là tiếp tục phát triển một cụm nhất định miễn là mật độ điểm dữ liệu trong “vùng lân cận” vượt quá một số ngưỡng. Ví dụ: đối với mỗi điểm dữ liệu trong một cụm nhất định, vùng lân cận (trong phạm vi bán kính nào đó) phải chứa ít nhất một số điểm tối thiểu.

Các phương pháp dựa trên mật độ có thể chia một tập hợp các đối tượng thành nhiều cụm riêng biệt hoặc một hệ thống phân cấp các cụm. Phương pháp cũng có thể được sử dụng để lọc nhiễu hoặc các ngoại lệ và khám phá các cụm có hình dạng tùy ý.

Hình 14 minh họa phương pháp phân cụm dựa trên mật độ. Tóm tắt đặc điểm của phương pháp phân cụm dựa trên mật độ như sau:

- Có thể tìm các cụm có hình dạng tùy ý



Hình 14: Phân cụm dựa trên mật độ

- Cụm là những vùng dày đặc các điểm dữ liệu, được ngăn cách bởi các vùng mật độ thấp
- Mật độ cụm: Mỗi điểm phải có số lượng tối thiểu các điểm trong “lân cận” của nó
- Có thể lọc ra các ngoại lệ

2.2.2 Phương pháp K-means

Giải thuật

K-means là giải thuật phân cụm phân vùng phổ biến nhất do tính đơn giản và hiệu quả của nó. Cho một tập hợp các điểm dữ liệu và số lượng k cụm cần thiết (tham số k do người dùng xác định), giải thuật này lặp lại phân vùng dữ liệu thành k cụm dựa trên hàm khoảng cách.

- Đầu vào: tập dữ liệu huấn luyện D , số lượng cụm k , hàm khoảng cách $dist(x, y)$.
- Đầu ra: k cụm độc lập (các điểm dữ liệu chỉ thuộc về một trong các cụm)

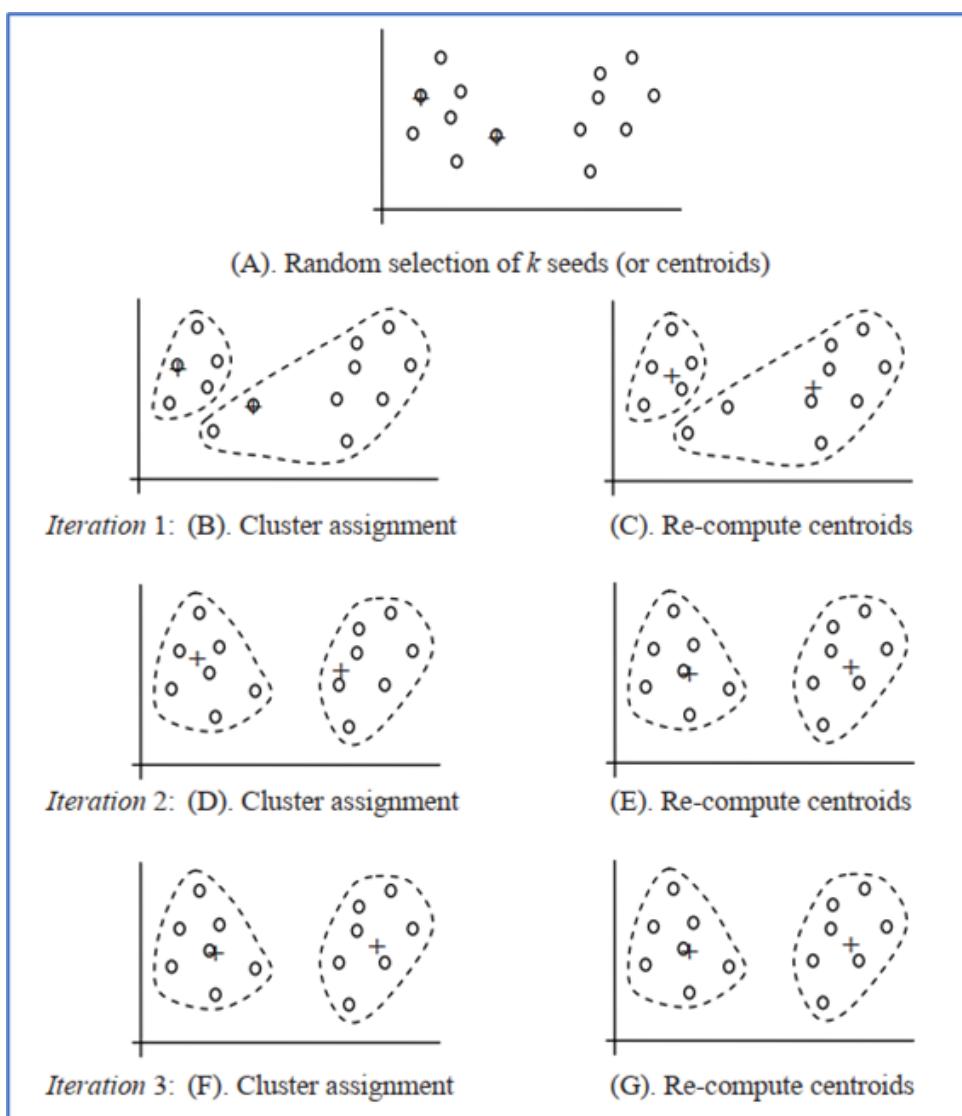
Giả sử D là tập dữ liệu gồm n phần tử dữ liệu, trong đó mỗi phần tử dữ liệu $x \in X$ là một vectơ r chiều $X \subseteq R^r$. Giải thuật K-means phân vùng dữ liệu đã cho thành k cụm. Mỗi cụm có một tâm cụm, còn được gọi là centroid. Centroid chỉ đơn giản là giá trị trung bình của tất cả các điểm dữ liệu trong cụm (do đó giải thuật được đặt tên là K-means). Hình 15 trình bày giải thuật K-means. Các tính toán chi tiết của giải thuật được trình bày ở Hình 16.

1. Initialization: select randomly k initial centroids $C = \{c_1, \dots, c_k\}$.
2. For each instance $x \in D$
 - **Assign** x to the cluster with nearest centroid
3. For each cluster $i \in \{1, \dots, k\}$:
 - **Recompute** the centroid of cluster i from its instances
4. Repeat Steps 2 and 3 until C no longer changes.

Hình 15: Giải thuật K-means

- **Assign** x to the cluster with nearest centroid
 - Compute distance from x to each centroid: $dist(x, centroid_i)$
 - Assign x to cluster of closest centroid:
$$i = \arg \min_{i \in \{1,2,\dots,k\}} dist(x, centroid_i); cluster_i \leftarrow x$$
- **Recompute** the centroid of cluster i from its instances
 - $centroid_i = \frac{1}{|cluster_i|} \sum_{x \in cluster_i} x$
 - Sum of instances: adding the corresponding dimensions of the vectors (vector addition)

Hình 16: Giải thuật K-means (chi tiết)



Hình 17: Ví dụ các bước giải thuật K-means (Liu [2011])

Ví dụ

Hình 17 biểu thị các dữ liệu trong không gian 2 chiều. Bài toán đặt ra làm tìm các cụm dữ liệu với $k = 2$. Đầu tiên, hai điểm dữ liệu (đánh dấu bằng dấu chéo) được chọn ngẫu nhiên làm trọng tâm ban đầu như Hình 17(A). Bước lặp thứ nhất: các điểm dữ liệu được gán cho hai cụm tương ứng với hai centroid (Hình 17(B)) và các centroid được tính toán lại (Hình 17(C)). Bước lặp thứ 2: các điểm dữ liệu được gán cho hai cụm tương ứng với hai centroid (Hình 17(D)) và các centroid được tính toán lại (Hình 17(E)). Bước lặp thứ 3: thực hiện tính toán tương tự hai bước đầu. Do không có sự thay đổi (gán lại) các điểm dữ liệu theo các cụm, giải thuật kết thúc.

<ul style="list-style-type: none"> ▪ Manhattan (L1-norm): 	$d(x, z) = \sum_{i=1}^n x_i - z_i $
<ul style="list-style-type: none"> ▪ Euclid (L2-norm): 	$d(x, z) = \sqrt{\sum_{i=1}^n (x_i - z_i)^2}$
<ul style="list-style-type: none"> ▪ Hamming distance: 	$d(x, z) = \sum_{i=1}^n Difference(x_i, z_i)$

Hình 18: Một số phương pháp tính khoảng cách

Các thành phần

Điều kiện dừng

Giải thuật K-means được xem là hội tụ tại thời điểm thay đổi sau các vòng lặp là không đáng kể. Các thay đổi ở đây liên quan đến trọng tâm, gán lại điểm dữ liệu (đến các trọng tâm), hoặc sai số (tổng bình phương khoảng cách từ các điểm đến trọng tâm gần nhất). Cụ thể, giải thuật hội tụ (dừng) khi thỏa mãn một trong các điều kiện sau đây:

- Rất ít trường hợp được gán lại cho các cụm mới
- Trọng tâm không thay đổi đáng kể
- Tổng sai số bình phương không thay đổi đáng kể

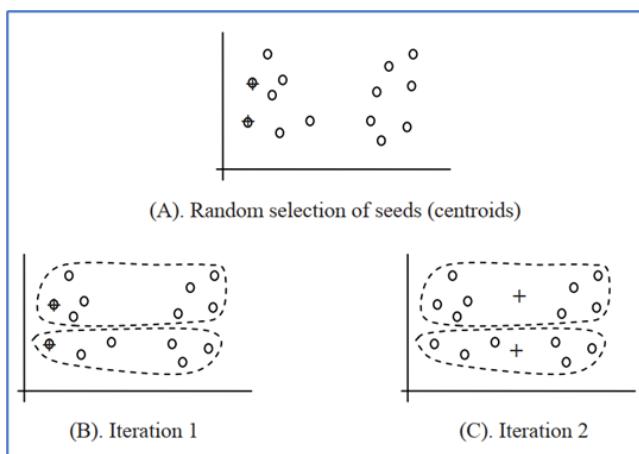
Khoảng cách giữa các điểm dữ liệu

K-means yêu cầu tính khoảng cách giữa các điểm dữ liệu và tâm cụm để gán lại các điểm dữ liệu theo tâm cụm mới. Tính toán khoảng cách phụ thuộc vào loại dữ liệu cụ thể. Một số cách tính khoảng cách giữa các điểm dữ liệu trình bày ở Hình 18.

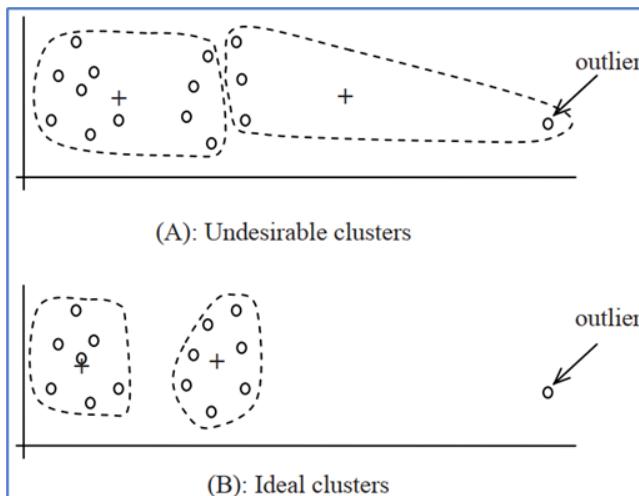
2.2.3 Một số vấn đề với K-means

Vấn đề khởi tạo

Giải thuật rất nhạy cảm với việc khởi tạo các centroids ban đầu. Tập khởi tạo khác nhau có thể tạo ra các cụm khác nhau. Hình 4.10 thể hiện quá trình phân cụm của tập dữ liệu 2 chiều. Mục tiêu là tìm hai cụm. Các hạt ban đầu được chọn ngẫu nhiên được đánh dấu bằng dấu gạch chéo trong Hình 19(A). Hình 19(B) đưa ra kết quả phân cụm



Hình 19: Minh họa vấn đề khởi tạo của giải thuật K-means (Liu [2011])



Hình 20: Minh họa ảnh hưởng các điểm dữ liệu ngoại lệ đến giải thuật K-means (Liu [2011])

của lần lặp đầu tiên. Hình 19(C) cho kết quả của lần lặp thứ hai. Vì không có sự thay đổi (gán lại) của các điểm dữ liệu nên giải thuật kết thúc.

Vấn đề Outliers

Giải thuật rất nhạy cảm với các ngoại lệ. Các ngoại lệ là các điểm dữ liệu ở rất xa các điểm dữ liệu khác. Các ngoại lệ có thể là lỗi trong quá trình ghi dữ liệu hoặc một số điểm dữ liệu đặc biệt có giá trị rất khác nhau. Do giải thuật K-means sử dụng giá trị trung bình làm trọng tâm của mỗi cụm, nên các giá trị ngoại lệ có thể dẫn đến các cụm không mong muốn như ví dụ ở Hình 20.

Vấn đề xác định số cụm tối ưu

K-means yêu cầu người dùng chỉ định số lượng cụm. Số cụm tối ưu là chủ quan và phụ thuộc vào thước đo khoảng cách. Trên thực tế số lượng cụm thích hợp là điều kiện giúp tìm ra phân cụm tối ưu với K-means. Số lượng các cụm phù hợp có thể được coi là tìm ra sự cân bằng tốt giữa khả năng nén và độ chính xác trong phân tích cụm. Hãy xem xét hai trường hợp cực đoan. Trường hợp toàn bộ tập dữ liệu là một cụm, sẽ tối đa hóa khả năng nén dữ liệu (một cụm dữ liệu). Mặt khác, nếu coi từng điểm dữ liệu là một cụm sẽ mang lại độ phân giải phân cụm tốt nhất (các khoảng cách từ điểm dữ liệu đến tâm cụm hay sai số residual sum-of-squares bằng 0).

Phương pháp Elbow

- Thực hiện giải thuật K-means với các giá trị khác nhau của k
- Với mỗi k , tính sai số tổng bình phương khoảng cách (S).
- Vẽ đồ thị của S theo số cụm k .
- Vị trí của khúc cua (khuỷu tay) trong sơ đồ là điểm k tối ưu.

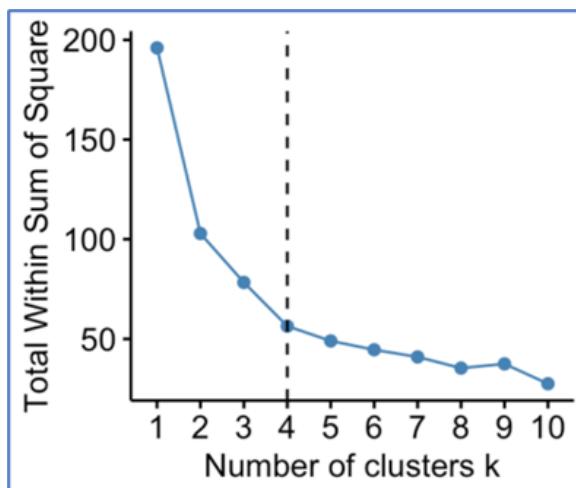
Phương pháp khuỷu tay (Hình 21) dựa trên quan sát rằng việc tăng số lượng cụm có thể giúp giảm tổng phương sai của mỗi cụm. Điều này là do số cụm lớn hơn dẫn đến có nhiều cụm dữ liệu tốt hơn, giống nhau hơn. Trường hợp tăng số lượng cụm mà không tạo nên các cụm tốt hơn có nghĩa là tổng phương sai các cụm không thay đổi đáng kể. Do đó, một phương pháp chọn số cụm phù hợp là sử dụng điểm ngoặt trên đường cong của tổng lỗi đối với số lượng cụm.

Phương pháp Silhouette

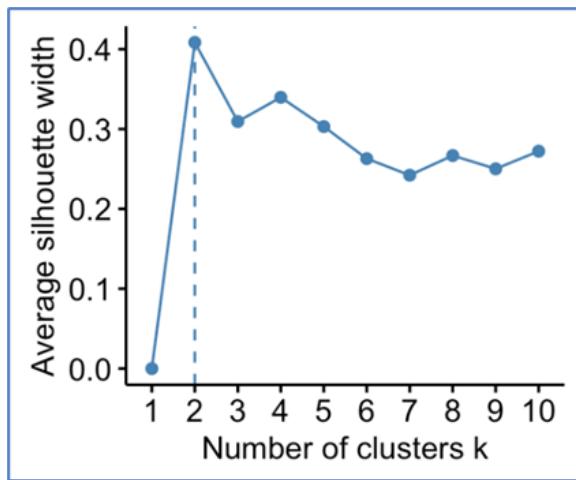
Như nội dung phần 2.2.1, giá trị Silhouette phản ánh chất lượng của phân cụm được thực hiện. Vì vậy, giá trị Silhouette có thể sử dụng để xác định số lượng cụm tối ưu. Phương pháp được thực hiện một cách đơn giản như sau:

- Thực hiện giải thuật K-means với các giá trị khác nhau của k
- Với mỗi k , tính giá trị Silhouette trung bình tất cả các cụm của phân cụm tìm được.
- Chọn số k cho giá trị Silhouette lớn nhất

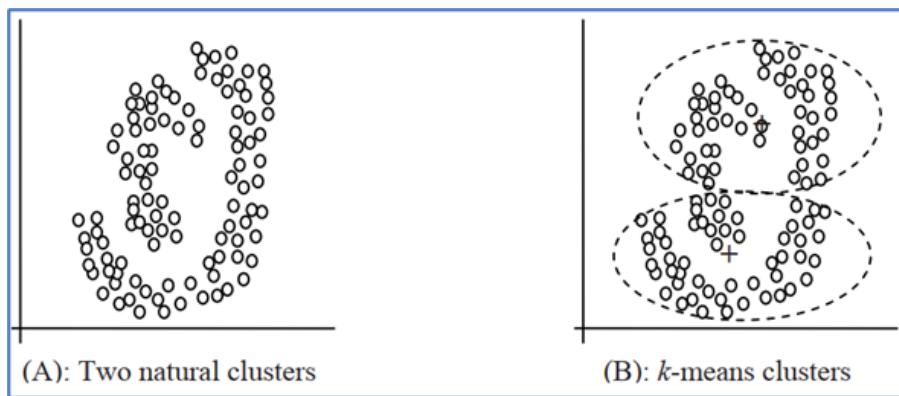
Phương pháp sử dụng giá trị Silhouette tìm số cụm tối ưu được minh họa ở Hình 22



Hình 21: Phương pháp khuỷu tay tìm số cụm tối ưu



Hình 22: Phương pháp Silhouette tìm số cụm tối ưu



Hình 23: Vấn đề đối với dữ liệu cong

Các vấn đề khác

Cụm trống

- Một số cụm có thể bị trống, nghĩa là không có điểm dữ liệu nào được chỉ định trong quá trình phân cụm.
- Một trong các phương pháp là chọn một điểm dữ liệu làm tâm thay thế. Ví dụ, chọn điểm dữ liệu xa nhất từ tâm của một cụm lớn làm tâm cụm thay thế. Ở đây, cụm lớn có nghĩa là tổng bình phương sai số lớn.

Cụm cong

- Giải thuật k-mean không phù hợp để khám phá các cụm không phải là hình cầu/lồi (như cụm cong).
- Ví dụ 7: Hình 23(A) hiển thị tập dữ liệu 2 chiều. Có hai cụm có hình dạng không đều. Tuy nhiên, hai cụm này không phải là hình cầu và giải thuật K-means sẽ không thể tìm được các cụm này. Thay vào đó, nó có thể tìm thấy hai cụm sai như trong Hình 23(B).

Dữ liệu lớn và thay đổi

- Thuận toán K-means phù hợp với dữ liệu kích thước nhỏ tới trung bình. Đối với dữ liệu có dữ liệu lớn và thay đổi, có thể cần các biến thể nâng cao, ví dụ Giải thuật Online K-means (tham khảo nội dung Online K-Means ở phần sau).
- Đối với dữ liệu có số chiều lớn, thước đo dựa trên khoảng cách có thể hội tụ về một giá trị không đổi đối với các cặp dữ liệu (các giá trị các chiều bù trừ lẫn nhau). Một trong các giải pháp là áp dụng các phương pháp giảm chiều dữ liệu (ví dụ PCA).

1. Choose an **centroid c_1** , randomly from D .

2. Choose the next **centroid c_i** :

- Select $c_i = x' \in X$ with probability $\frac{dist(x')^2}{\sum_{x \in X} dist(x)^2}$
- $dist(x)$ denote the shortest distance from x to the closest centroid

3. Repeat Step 2 until a total of k centers has been chosen.

Hình 24: Giải thuật K-means++

2.2.4 Khởi tạo với K-means++

Giải thuật

Vấn đề với khởi tạo của K-means được trình bày ở Phần 2.2.3. Thuật toán K-means rất nhạy cảm với việc khởi tạo các centroids ban đầu. Tập khởi tạo khác nhau có thể tạo ra các cụm khác nhau.

Ý tưởng của K-mean++ [Arthur and Vassilvitskii, 2007] là khởi tạo các trọng tâm ban đầu cách xa nhau. Điều này làm tăng khả năng các centroid nằm trong các cụm khác nhau. Vì trọng tâm được chọn từ các điểm dữ liệu nên mỗi trọng tâm có ít nhất một điểm dữ liệu liên kết với nó.

Giải thuật K-means++ được trình bày ở Hình 24.

Điễn giải giải thuật K-means++:

- Xác xuất $\frac{dist(x')^2}{\sum_{x \in X} dist(x)^2}$ tỷ lệ thuận với khoảng cách điểm dữ liệu đến tâm cụm gần nhất.
- Điểm có khoảng cách lớn nhất có nhiều khả năng được chọn nhất. Điều này làm tăng cơ hội chọn được các tâm nằm ở các cụm khác nhau.
- Các tâm được chọn với một mức độ ngẫu nhiên nào đó.

Đánh giá giải thuật

Thực nghiệm để đánh giá K-mean++ sử dụng 2 tập dữ liệu, Norm 25 và Spam:

- Norm 25: Tập dữ liệu tổng hợp, bao gồm 25 trung tâm (centroids) từ một siêu khối 15 chiều có chiều dài cạnh 500, 1000 điểm dữ liệu.
- Spam: tập dữ liệu thư rác thực tế, sử dụng cho hệ thống phát hiện thư rác, bao gồm 4601 điểm dữ liệu với 58 chiều.

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$1.365 \cdot 10^5$	8.47%	$1.174 \cdot 10^5$	0.93%	0.12	46.72%
25	$4.233 \cdot 10^4$	99.96%	$1.914 \cdot 10^4$	99.92%	0.90	87.79%
50	$7.750 \cdot 10^3$	99.81%	$1.474 \cdot 10^1$	0.53%	2.04	-1.62%

Hình 25: Giải thuật K-means++ với dữ liệu Norm 16

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$3.698 \cdot 10^4$	49.43%	$3.684 \cdot 10^4$	54.59%	2.36	69.00%
25	$3.288 \cdot 10^4$	88.76%	$3.280 \cdot 10^4$	89.58%	7.36	79.84%
50	$3.183 \cdot 10^4$	95.35%	$2.384 \cdot 10^4$	94.30%	12.20	75.76%

Hình 26: Giải thuật K-means++ với dữ liệu Norm 16

Đối với mỗi tập dữ liệu, thực hiện thuật toán K-means truyền thống (viết tắt là K-means) và K-means với khởi tạo bằng K-means++ (viết tắt là K-means++), với các giá trị $k = 10, 25$ và 50 . Các chỉ số đánh giá bao gồm:

- Φ : tổng bình phương khoảng cách giữa mỗi điểm và tâm gần nhất của nó.
- T: thời gian thực hiện

Kết quả thực nghiệm trình bày ở Hình 25 (tập dữ liệu Norm 25) và 26 (tập dữ liệu Spam). Đối với k-means, Hình vẽ liệt kê Φ và thời gian tính bằng giây. Đối với K-means++, Hình vẽ liệt kê phần trăm cải thiện so với K-means.

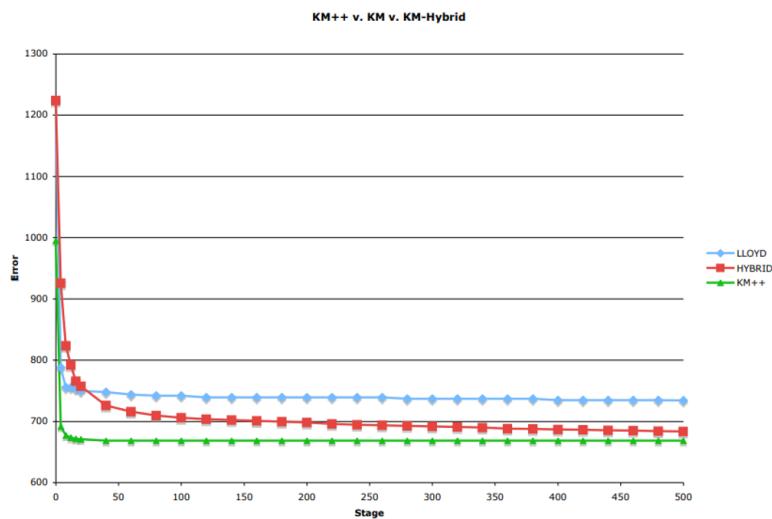
Về tính hội tụ, từ so sánh ở Hình 27 ta thấy K-means++ giúp K-means hội tụ và đạt được điểm cực trị nhanh chóng. Kết quả cũng cho thấy K-means++ đạt sai số nhỏ hơn so với phương pháp K-means truyền thống.

2.2.5 Phân cụm dữ liệu lớn với Online K-means

Tuần tự hóa K-means

Vấn đề về cơ chế hoạt động của K-means

- Tại mỗi lần lặp, K-means đọc tất cả dữ liệu huấn luyện để tính toán các trọng tâm (centroids) mới và gán lại các điểm dữ liệu cho các trọng tâm này. Vì thế, K-means gặp khó khăn khi làm việc với các bộ dữ liệu lớn.
- Ngoài ra, cũng do yêu cầu đọc toàn bộ dữ liệu huấn luyện mỗi vòng lặp, nó không thể làm việc với dữ liệu luồng trong đó dữ liệu xuất hiện theo thứ tự.



Hình 27: So sánh K-means++ và K-mean

K-means thực ra đang tìm các tâm cụm C_i để mà có sai số E nhỏ nhất, trong đó:

$$E = \sum_{i=1}^k \sum_{x \in C_i} dist(x, cen_i)^2 = \sum_{x \in D} dist(x, cen_i)^2 \quad (18)$$

Gọi $w(x_i)$ là tâm của cụm chứa điểm x_i . Ta có thể viết lại công thức sai số dưới dạng:

$$Q_{k-means} = \sum_{i=1}^M (x_i - w(x_i))^2 \quad (19)$$

Như vậy việc phân cụm theo K-means có thể được thực hiện theo phương pháp giảm đạo hàm, với γ_t là hằng số tốc độ học (learning rate):

$$w_{t+1} = w_t + \gamma_t \sum_{i=1}^M (x_i - w_t(x_i)) \quad (20)$$

Ở mỗi lần lặp, đạo hàm tính toán dựa trên tất cả dữ liệu đào tạo:

$$Q'_t = \sum_{i=1}^M (x_i - w(x_i)) \quad (21)$$

Như vậy việc này sẽ rất tốn kém khi tập dữ liệu lớn. Hơn nữa, ý tưởng đó sẽ không khả thi đối với luồng dữ liệu, khi mà tại mỗi thời điểm ta chỉ có thể thu thập được một tập nhỏ dữ liệu, chứ không phải tập đầy đủ.

Phương pháp Online K-means giảm thiểu tính toán Q' bằng cách sử dụng thông tin đạo hàm ngẫu nhiên (Stochastic gradient). Ở mỗi lần lặp, Online K-means chỉ sử dụng

- Initialize k centroids randomly.
- Update the centroids as an instance comes
 - At iteration t , take an instance x_t .
 - Find the nearest centroid w_t to x_t
 - Update w_t as follows:

$$w_{t+1} = w_t + \gamma_t(x_t - w_t)$$

Hình 28: Giải thuật Online K-means

thông tin liên quan một điểm dữ liệu x_t từ công thức Q' . Như vậy việc cập nhật trong giải thuật học sẽ đơn giản hơn nhiều:

$$w_{t+1} = w_t + \gamma_t(x_t - w_t(x_t)) \quad (22)$$

Thuật toán Online K-means

Giải thuật Online K-means được trình bày ở Hình 28.

Tốc độ học (learning rate)

- Tốc độ học là hằng số dương và thỏa mãn $\sum_{i=0}^{\infty} \gamma_t = \infty; \sum_{i=0}^{\infty} \gamma_t^2 < \infty$
- Một trong những lựa chọn phổ biến của tốc độ học là $\gamma_t = (t + \tau)^{-\kappa}$ trong đó τ, κ là các hằng số dương.
- $\kappa \in (0, 5, 1]$ được gọi là tốc độ quên. κ lớn có nghĩa là giải thuật ghi nhớ quá khứ lâu hơn và quan sát mới đóng vai trò ít quan trọng hơn khi t tăng.

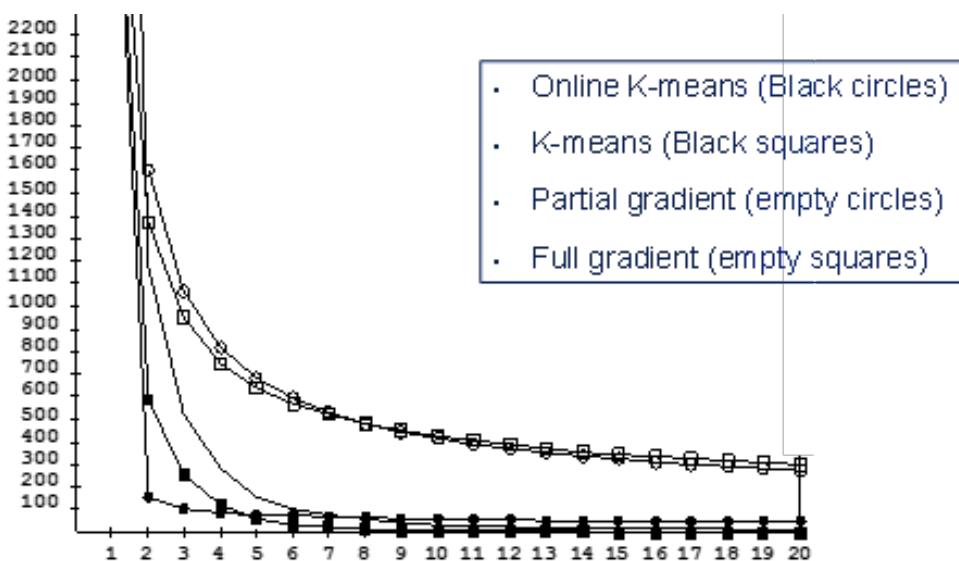
Ví dụ về hiệu suất Online K-means chỉ ra ở Hình 29.

2.2.6 Phân cụm phân cấp

Các phương pháp phân cụm phân cấp

Một số trường hợp, người sử dụng có thể có thể muốn phân vùng dữ liệu thành các nhóm ở các cấp độ khác nhau. Phương pháp phân cụm phân cấp hoạt động bằng cách nhóm các đối tượng dữ liệu thành một hệ thống phân cấp hoặc “cây” của các cụm.

Phương pháp phân cụm phân cấp có thể là phân cụm tổng hợp hoặc phân chia, tùy thuộc vào các cụm được hình thành theo kiểu từ dưới lên (hợp nhất các cụm thành một



Hình 29: Ví dụ về hiệu suất Online K-means

cụm lớn hơn) hay từ trên xuống (tách các cụm thành các cụm nhỏ hơn).

Phương pháp phân cụm phân cấp tổng hợp (Agglomerative Hierarchical clustering) sử dụng chiến lược từ dưới lên. Nó thường bắt đầu bằng cách cho phép mỗi đối tượng tạo thành cụm riêng của nó và lặp đi lặp lại việc hợp nhất các cụm thành các cụm lớn hơn và lớn hơn cho đến khi tất cả các đối tượng nằm trong một cụm duy nhất hoặc các điều kiện kết thúc nhất định được thỏa mãn. Đối với bước hợp nhất, nó tìm hai cụm gần nhau nhất (theo một số thước đo tương tự) và kết hợp cả hai để tạo thành một cụm. Bởi vì hai cụm được hợp nhất trong mỗi lần lặp, trong đó mỗi cụm chứa ít nhất một đối tượng, nên một phương pháp kết tụ yêu cầu tối đa n lần lặp:

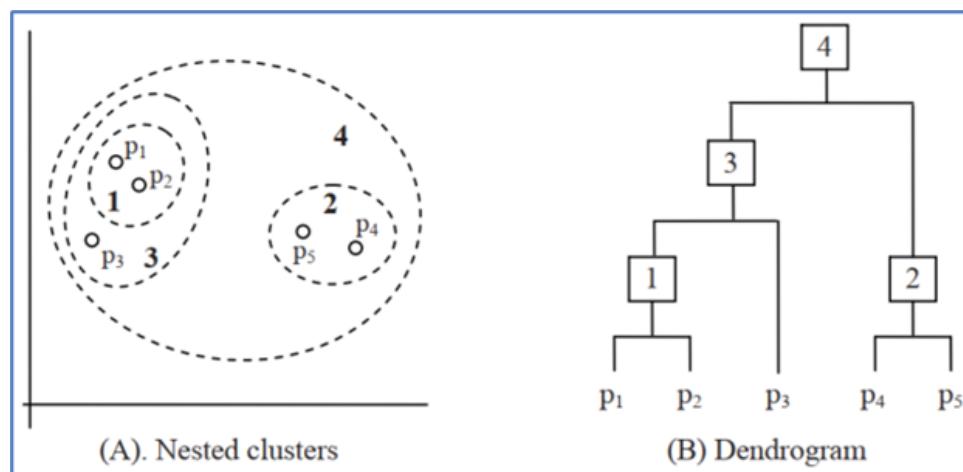
- Xây dựng dendrogram (cây) từ cấp dưới cùng
- Hợp nhất cặp cụm giống nhau nhất
- Tiếp tục cho đến khi tất cả các điểm được hợp nhất thành một cụm

Trong phân cụm phân cấp tổng hợp, yếu tố quan trọng là xác định khoảng cách giữa các cụm để nhóm các cụm tương đồng trong các lần lặp. Các phương pháp phân cụm phân cấp tổng hợp sử dụng các độ đo khác nhau để tính khoảng cách giữa các cụm.

Phương pháp phân cụm phân cấp phân chia (Divisive Hierarchical clustering) sử dụng chiến lược từ trên xuống. Nó bắt đầu bằng cách đặt tất cả các đối tượng vào một cụm chung duy nhất. Sau đó, nó chia cụm gốc thành các cụm con nhỏ hơn và tiếp tục phân chia các cụm đó thành các cụm nhỏ hơn nữa. Quá trình phân vùng tiếp tục cho đến khi

1. Make each data point in the data set D a cluster,
2. Compute all pair-wise distances of $x_1, x_2, \dots, x_n \in D$;
3. Repeat
 4. Find **two clusters** that are nearest to each other;
 5. Merge the two clusters form a new cluster c ;
 6. Compute the distance from c to all other clusters;
7. Until there is only one cluster left

Hình 30: Giải thuật phân cụm phân cấp



Hình 31: Ví dụ về phân cụm phân cấp

mỗi cụm ở mức thấp nhất chỉ chứa một điểm dữ liệu hoặc các đối tượng trong một cụm đều giống nhau. Trong phân cụm phân cấp kết tụ hoặc phân chia, người dùng có thể chỉ định số lượng cụm mong muốn làm điều kiện kết thúc.

- Bắt đầu với tất cả các điểm dữ liệu trong một cụm (cấp cao nhất)
- Chia một cụm thành các cụm nhỏ hơn dựa trên một số tiêu chí
- Tiếp tục cho đến khi mỗi cụm là một điểm duy nhất.

Giải thuật của phân cụm phân cấp phân chia tương tự giải thuật của phân cụm tổng hợp với việc hình thành các cụm theo hướng ngược lại. Tại mỗi lần lặp, các cụm phân chia thành các cụm nhỏ hơn thay vì tổng hợp các cụm nhỏ thành cụm lớn hơn.

Các phương pháp tính khoảng cách giữa các cụm

Phương pháp Single-link

Phương pháp này đo khoảng cách giữa hai cụm tính bằng khoảng cách nhỏ nhất của hai phần tử dữ liệu thuộc hai cụm. Được gọi là giải thuật phân cụm lân cận gần nhất hoặc giải thuật liên kết đơn (quá trình phân cụm bị chấm dứt khi khoảng cách giữa các cụm gần nhất vượt quá ngưỡng do người dùng xác định)

Nếu chúng ta xem các điểm dữ liệu dưới dạng các nút của đồ thị, việc nhóm các cụm tương ứng với việc thêm một cạnh giữa cặp nút gần nhất trong các cụm đó. Giải thuật phân cụm phân cấp sử dụng khoảng cách nhỏ nhất tương tự như bài toán “minimal spanning tree algorithm” tìm cách kết nối các điểm dữ liệu sao cho tổng các cạnh tạo nên là nhỏ nhất.

Đặc điểm

- Khoảng cách cụm: khoảng cách giữa hai điểm gần nhất trong hai cụm
- Hợp nhất hai cụm với khoảng cách cặp tối thiểu nhỏ nhất.
- Có thể nhạy cảm với nhiễu trong dữ liệu (có hiệu ứng dây chuyền)

Phương pháp Complete-Link

Phương pháp này sử dụng khoảng cách tối đa giữa hai điểm của hai cụm để đo khoảng cách giữa các cụm. Nó còn được gọi là giải thuật phân cụm lân cận xa nhất hay giải thuật liên kết hoàn chỉnh (Complete-Link Method).

Minimum distance: $dist_{min}(C_i, C_j) = \min_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} \{|\mathbf{p} - \mathbf{p}'|\}$

Maximum distance: $dist_{max}(C_i, C_j) = \max_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} \{|\mathbf{p} - \mathbf{p}'|\}$

Mean distance: $dist_{mean}(C_i, C_j) = |\mathbf{m}_i - \mathbf{m}_j|$

Average distance: $dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} |\mathbf{p} - \mathbf{p}'|$

Hình 32: Các phương pháp tính khoảng cách cụm

Do khoảng cách giữa hai cụm được xác định bởi các nút ở xa nhất trong hai cụm, phương pháp này có xu hướng giảm thiểu sự gia tăng đường kính của các cụm tại mỗi lần lặp. Nếu các cụm thực tế khá nhỏ gọn và có kích thước xấp xỉ bằng nhau thì phương pháp này sẽ tạo ra các cụm có chất lượng cao.

Đặc điểm

- Khoảng cách cụm: khoảng cách giữa hai điểm xa nhất trong hai cụm
- Hợp nhất hai cụm với khoảng cách cặp tối đa nhỏ nhất.
- Có thể nhạy cảm với tiếng ồn (không gắp vấn đề về hiệu ứng dây chuyền)

Các phương pháp khác

- Phương pháp liên kết trung bình: Khoảng cách giữa hai cụm là khoảng cách trung bình của tất cả các khoảng cách theo cặp giữa hai cụm.
- Phương pháp centroid: Khoảng cách giữa hai cụm là khoảng cách giữa các trọng tâm của chúng.
- Phương pháp Ward: Khoảng cách giữa hai cụm là sự gia tăng tổng bình phương (khoảng cách) từ sai số của hai cụm đến sai số của một cụm được hợp nhất

CHƯƠNG 3. PHÂN LOẠI

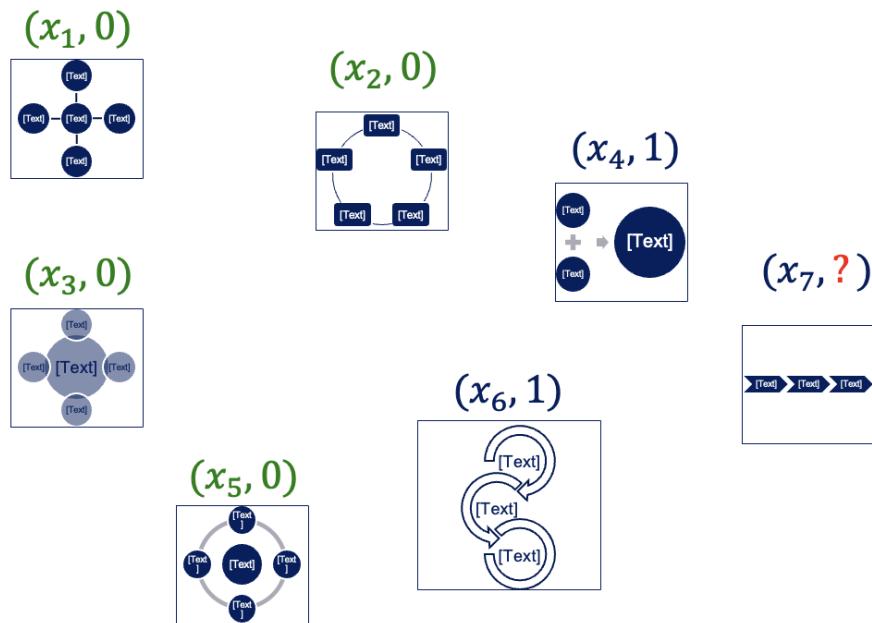
3.1 Học dựa trên láng giềng

3.1.1 Bài toán học có giám sát

Xét lớp bài toán học trên tập dữ liệu có nhãn $\{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}, 1 \leq i \leq M\}$, trong đó \mathcal{X} là không gian đầu vào, và \mathcal{Y} là không gian nhãn đầu ra. Chúng ta cần học hàm y^* kết nối giữa \mathbf{x} và y .

Dựa vào đặc điểm của không gian đầu ra, chúng ta có một số nhánh con thường thấy của bài toán học có giám sát:

- $\mathcal{Y} = \{0, 1\}$: Bài toán phân loại nhị phân
- $\mathcal{Y} = \{1, 2, \dots, C\}$: Bài toán phân loại nhiều lớp
- $\mathcal{Y} \in \mathbb{R}$: Bài toán hồi quy



Hình 33: Ví dụ về dự đoán trên dữ liệu có nhãn.

Chúng ta sẽ bắt đầu với một trong những thuật toán cơ bản nhất của học có giám sát. Lấy ví dụ về một bài toán dự đoán trên dữ liệu có nhãn như hình 33, trong đó $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_5$ là các dữ liệu với 5 thành phần con được gán nhãn 0. $\mathbf{x}_4, \mathbf{x}_6$ là dữ liệu với 3 thành phần con được gán nhãn 1. Như vậy \mathbf{x}_7 với 3 thành phần con nên được gán nhãn nào? Ta thấy tính chất của \mathbf{x}_7 gần giống với tính chất của $\mathbf{x}_4, \mathbf{x}_6$ nhất, vì vậy ta có thể

gán nhãn cho x_7 tương tự như x_4, x_6 là 1. Đây là một minh họa của phương pháp học dựa trên hàng xóm.

3.1.2 Học dựa trên hàng xóm

Ý tưởng cơ bản của phương pháp học dựa trên hàng xóm là các mẫu có thuộc tính tương tự nhau sẽ có nhãn tương tự nhau. Cụ thể, để thực hiện được phương pháp học này, chúng ta cần:

- Lưu trữ tất cả các mẫu dữ liệu trong tập học D .
- Để dự đoán cho một điểm dữ liệu mới x , chúng ta cần tìm những điểm dữ liệu trong D có thuộc tính tương tự (hang xóm) như x . Nhãn dự đoán của x được xác định bằng cách tổng hợp các nhãn của tập hàng xóm đã tìm được.

Để tìm được tập hàng xóm, chúng ta cần hai thành phần:

- Một độ đo để đánh giá độ tương tự: $d(\mathbf{u}, \mathbf{v}) \in R^+, \mathbf{u}, \mathbf{v} \in \mathcal{X}$
- Tập hàng xóm được xác định bằng độ đo trên.

Các tính chất của thuật toán học dựa trên hàng xóm

Một thuật toán học dựa trên hàng xóm thường có các tính chất sau:

- Không cần học một tham số nào (non-parametric).
- Học bằng các lưu trữ tất cả các dữ liệu cung cấp để huấn luyện.
- Tốn chi phí cho việc dự đoán. Mỗi lần dự đoán cần truy vấn lại các dữ liệu đã có. Điều này dẫn tới cần chi phí cho lưu trữ hoặc tính toán song song hiệu quả.
- Có ít thông tin về cấu trúc bên trong của dữ liệu.
- Hỗ trợ một cách tự nhiên với dữ liệu tăng dần theo thời gian.

Với những đặc điểm trên, học dựa trên hàng xóm còn được gọi với một số tên khác như: học dựa trên bộ nhớ (memory-based learning), học chậm (lazy learning), học theo điểm dữ liệu (instance-based learning).

3.1.3 Thuật toán k lóng giềng gần nhất (kNN)

Khi ta giới hạn số lượng hàng xóm (lóng giềng) của dữ liệu mới là k , ta được thuật toán k lóng giềng gần nhất kNN. Khi đó, thuật toán kNN nói chung được mô tả như sau:

Đầu vào: Tập học \mathcal{D} chứa các cặp dạng $(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}$; độ đo tương đồng $d(\mathbf{u}, \mathbf{v}) \in \mathbb{R}^+, \mathbf{u}, \mathbf{v} \in \mathcal{X}$; số hàng xóm $k > 0$.

Học: Lưu trữ toàn bộ \mathcal{D}

Dự đoán:

- Với mỗi điểm dữ liệu mới \mathbf{x} , tính tập hàng xóm $NB_k(\mathbf{x})$ là chỉ mục (index) sao cho $d(\mathbf{x}_i, \mathbf{x}), i \in NB_k(\mathbf{x})$ thuộc top k giá trị tương đồng tốt nhất (khoảng cách nhỏ nhất).
- Nhãn dự đoán của \mathbf{x} là tổng hợp nhãn $y = Aggr(y_i | i \in NB_k(\mathbf{x}))$, trong đó $Aggr$ là một hàm kết tập tùy thuộc vào không gian của \mathcal{Y} .

kNN trong bài toán phân loại

Khi \mathcal{Y} là tập nhãn rời rạc, thì hàm $Aggr$ thường được sử dụng là hàm đề xuất nhiều nhất (max-voting), tức là nhãn được xuất hiện nhiều nhất trong $NB_k(\mathbf{x})$ sẽ được sử dụng là nhãn của \mathbf{x} .

$$y = LabelAppearTheMost(\{y_i | i \in NB_k(\mathbf{x})\})$$

kNN trong bài toán hồi quy

Khi \mathcal{Y} là miền liên tục, thì hàm $Aggr$ thường được sử dụng là hàm trung bình cộng.

$$y = \frac{1}{|NB_k(\mathbf{x})|} \sum_{i \in NB_k(\mathbf{x})} y_i$$

3.1.4 Hai thành phần cơ bản trong kNN

Chúng ta xem xét hai thành phần cơ bản trong thuật toán kNN: độ đo tương đồng và số lượng hàng xóm k .

i) Độ đo tương đồng

Độ đo tương đồng đóng một vai trò quan trọng trong thuật toán kNN khi nó đi kèm với giả thiết về phân bố của dữ liệu, và một khi đã xác định sẽ không thay đổi. Sử dụng

độ đo tương đồng nào thường phụ thuộc vào bài toán và kiểu dữ liệu cụ thể. Một số độ đo tương đồng thường dùng là:

- Khoảng cách hình học (Geometric distance), (như khoảng cách Euclid). Độ đo này thường được sử dụng ở bài toán với dữ liệu thực tế trên miền liên tục.
- Khoảng cách Hamming, được sử dụng với dữ liệu có thuộc tính nhị phân
- Một số đánh giá cách như tương đồng Cosine.

Một số ví dụ cụ thể về khoảng cách hình học: Kí hiệu $\mathbf{x}[i]$ là phần tử thứ i của \mathbf{x} (phân biệt với \mathbf{x}_i - phần tử thứ i trong tập dữ liệu).

- Minkowski (L_p -norm):

$$d(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^N |\mathbf{x}[i] - \mathbf{z}[i]|^p \right)^{1/p}$$

- Manhattan (L_1 -norm):

$$d(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^N |\mathbf{x}[i] - \mathbf{z}[i]| \right)$$

- Euclid (L_2 -norm):

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^N (\mathbf{x}[i] - \mathbf{z}[i])^2}$$

- Chebyshev (max norm):

$$d(\mathbf{x}, \mathbf{z}) = \lim_{p \rightarrow \infty} \left(\sum_{i=1}^N |\mathbf{x}[i] - \mathbf{z}[i]|^p \right)^{1/p} = \max_i |\mathbf{x}[i] - \mathbf{z}[i]|$$

Với khoảng cách Hamming:

$$d(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^K Difference(\mathbf{x}[i], \mathbf{z}[i])$$

Chẳng hạn, cho thuộc tính nhị phân thì *Difference* có thể định nghĩa:

$$\text{Difference}(a, b) = \begin{cases} 1, & \text{nếu } a \neq b \\ 0, & \text{nếu } a = b \end{cases}$$

Với khoảng cách *Cosine*:

$$d(\mathbf{x}, \mathbf{z}) = \frac{\mathbf{x}^T \mathbf{z}}{\|\mathbf{x}\| \cdot \|\mathbf{z}\|}$$

Số lượng hàng xóm k

Một câu hỏi được đặt ra, chúng ta nên đặt k bằng bao nhiêu. Thực tế k nên chọn bằng các phương pháp đánh giá mô hình trong một tập các giá trị của k để chọn giá trị tốt nhất. Chúng ta nên bắt đầu từ khoảng $k > 1$ nhưng không quá lớn. Một số trường hợp k có thể lên tới vài trăm.

Có một số lưu ý:

- Dữ liệu đôi khi bị nhiễu, vì vậy để giảm thiểu ảnh hưởng của nhiễu và giảm sai số, k thông thường lớn hơn 1.
- Sử dụng quá nhiều k sẽ có thể không tập trung vào miền thuộc tính chính của dữ liệu mà lan sang những miền ít liên quan, khiến việc dự đoán bị tệ đi.

3.1.5 Một số phương pháp để cải thiện hiệu quả của kNN

Chúng ta thường sử dụng một số phương án ở bước xử lý thuộc tính đầu vào để cải thiện hiệu quả của phương pháp kNN.

1. Chuẩn hoá thuộc tính

Chuẩn hoá các thuộc tính đôi khi là một yếu tố quan trọng để có được dự đoán tốt trong kNN.

- Việc không chuẩn hoá dẫn tới biên độ của các thuộc tính sẽ đóng một vai trò quan trọng, và có khả năng lấn át vai trò của các thuộc tính khác. Chẳng hạn xét ví dụ sau:

Ta cần tính khoảng cách để chọn hàng xóm với khoảng cách Euclid:

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^N (\mathbf{x}[i] - \mathbf{z}[i])^2}$$

Với trường dữ liệu với 3 thuộc tính (Age, Income, Height) lần lượt là

- $\mathbf{x} = (20, 12000, 1.68)$
- $\mathbf{z} = 40, 1300, 1.75)$

Khi đó:

$$d(\mathbf{x}, \mathbf{z}) = [(20 - 40)^2 + (12000 - 1300)^2 + (1.68 - 1.75)^2)]^{0.5}$$

Thì trường *Income* mặc nhiên lần át hai trường còn lại. Vì vậy việc chuẩn hoá các trường dữ liệu đôi khi là cần thiết.

Một số phương pháp chuẩn hoá thường dùng:

- Chuẩn hoá [min, max]. Cho tất cả các giá trị của các trường nằm trong khoảng cho trước, chẳng hạn [-1,1] hoặc [0, 1]
- Chuẩn hoá theo z-score: Cho các trường dữ liệu đều có giá trị trung bình là 0, và variance là một giá trị cho trước (chẳng hạn 1)

2. Đánh trọng số cho thuộc tính

Khác với chuẩn hoá các trường thuộc tính khi chúng ta có xu hướng cho các thuộc tính có vai trò gần như nhau, thì trong một số trường hợp chúng ta muốn ưu tiên độ quan trọng của một số trường và giảm độ quan trọng của các trường khác. Khi đó chúng ta sẽ dùng phương pháp đánh trọng số cho thuộc tính.

Chúng ta có thể đánh lại trọng số trong công thức tính độ tương đồng (khoảng cách) của các điểm dữ liệu. Chẳng hạn, chúng ta sẽ sửa đổi khoảng cách Euclid thành:

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^n w_i (\mathbf{x}[i] - \mathbf{z}[i])^2}$$

Việc xác định trọng số phù hợp sẽ phụ thuộc vào kiến thức chuyên môn liên quan tới

tính chất dữ liệu. Trong một số trường hợp, có thể sử dụng việc học ra các trọng số phù hợp với dữ liệu.

3. Đánh trọng số cho hàng xóm

Chúng ta xem lại công thức kết tập nhãn cho việc dự đoán đã cho ở các phần trên:

$$y = \frac{1}{|NB_k(\mathbf{x})|} \sum_{i \in NB_k(\mathbf{x})} y_i$$

Khi này, các hàng xóm trong k hàng xóm đều được đánh giá vai trò ngang bằng nhau, tuy nhiên theo logic thông thường, những hàng xóm gần hơn nên có ảnh hưởng lớn hơn. Vì vậy thực tế, khi kết tập thông tin nhãn của các hàng xóm, chúng ta sẽ ưu tiên gán ảnh hưởng cao hơn cho hàng xóm gần hơn.

Đặt $v(\mathbf{x}, \mathbf{z})$ phụ thuộc nghịch đảo của khoảng cách $d(\mathbf{x}, \mathbf{z})$, chẳng hạn:

$$\begin{aligned} v(\mathbf{x}, \mathbf{z}) &= \frac{1}{\alpha + d(\mathbf{x}, \mathbf{z})} \text{ or} \\ v(\mathbf{x}, \mathbf{z}) &= \frac{1}{\alpha + [d(\mathbf{x}, \mathbf{z})]^2} \text{ or} \\ v(\mathbf{x}, \mathbf{z}) &= e^{\frac{-d(\mathbf{x}, \mathbf{z})^2}{\sigma^2}} \end{aligned}$$

Gọi $NB(z)$ là tập hàng xóm được chọn gần với z nhất. Với bài toán hồi quy, ta có thể gán nhãn:

$$y_z = \frac{\sum_{x \in NB(z)} v(\mathbf{x}, \mathbf{z}) y_x}{\sum_{x \in NB(z)} v(\mathbf{x}, \mathbf{z})}$$

Với bài toán phân lớp, ta có thể gán nhãn:

$$c_z = \arg \max_{c_j \in C} \sum_{x \in NB(z)} v(\mathbf{x}, \mathbf{z}) \text{Identical}(c_j, c_x)$$

Với:

$$\text{Identical}(a, b) = \begin{cases} 1, & \text{nếu } a = b \\ 0, & \text{nếu } a \neq b \end{cases}$$

3.1.6 Ưu điểm, nhược điểm của kNN

kNN có một số ưu điểm sau:

- Tốn ít chi phí cho việc học.
- Có nhiều tuỳ chọn cho việc đánh giá độ tương đồng (khoảng cách).
- Có khả năng loại bỏ một số ảnh hưởng của nhiễu khi k lớn.
- Một số nghiên cứu lí thuyết có chỉ ra kNN có thể tiến tới hiệu quả tốt nhất trong các phương pháp hồi quy dưới một số điều kiện xác định.

Đi kèm với đó, kNN có một số nhược điểm như:

- Đôi khi khó khăn trong việc lựa chọn độ đo tương đồng phù hợp.
- Tốn chi phí khi thực hiện dự đoán.

3.2 Cây quyết định và Rừng ngẫu nhiên

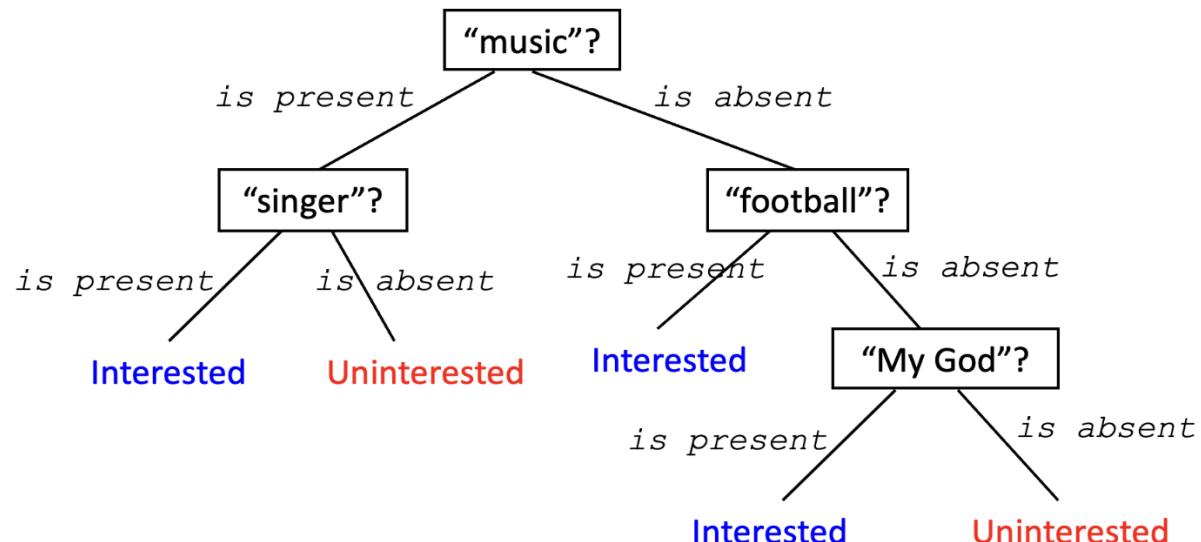
3.2.1 Giới thiệu

Đối với bài toán học có giám sát, chúng ta thường cần tìm hàm y^* từ một tập dữ liệu \mathcal{D} đã có nhãn. Do không biết hàm y^* , nên ta thường chọn một lớp mô hình \mathcal{H} để xấp xỉ nó. Chương 2 đã mô tả một ví dụ về lớp mô hình tuyến tính, tức là ta xấp xỉ y^* bởi một hàm dạng tuyến tính. Với cách làm đó, dạng hàm \mathcal{H} thường được chọn trước và khá chi tiết về hình dáng (ví dụ hàm tuyến tính). Cách làm này rất phổ biến trong ML, và thường được gọi là những phương pháp có tham số (parametric methods).

Ở đây chúng ta sẽ tìm hiểu một cách tiếp cận hoàn toàn khác. Ta sẽ không chọn trước hình dáng của hàm, mà sẽ dùng cấu trúc cây (tree structure) để biểu diễn các hàm. Mỗi cây sẽ biểu diễn một hàm cụ thể. Như vậy việc học sẽ đi tìm một cây mà có thể xấp xỉ y^* tốt. Những cây như thế được gọi là *Cây quyết định* (*Decision tree*).

Cụ thể hơn, cây quyết định là một cấu trúc dạng cây trong đó mỗi nút trong biểu diễn một thuộc tính cần kiểm tra giá trị đối với các mẫu, mỗi nhánh từ một nút sẽ tương ứng với một giá trị có thể của thuộc tính gắn với nút đó và mỗi nút lá biểu diễn một lớp hay là một dự đoán cuối cùng. Mục đích của ta là tạo ra một mô hình dự đoán giá trị của biến đích bằng cách học các luật quyết định IF-THEN đơn giản được suy ra từ các thuộc tính của dữ liệu. Một cây quyết định học được sẽ dự đoán một mẫu dữ liệu bằng cách duyệt cây từ nút gốc đến một nút lá, trong đó nhãn lớp hay giá trị gắn với nút lá đó sẽ được dùng để dự đoán. Như vậy, một cây sẽ biểu diễn một hàm phân loại hoặc hồi qui nào đó.

Các mô hình cây trong đó biến đích lấy một tập giá trị rời rạc được gọi là cây phân loại; trong các cấu trúc cây này, mỗi đường đi (path) từ nút gốc đến một nút lá sẽ tương ứng với một kết hợp (conjunction) của các kiểm tra giá trị thuộc tính (attribute tests). Cây quyết định (bản thân nó) chính là một phép tuyển (disjunction) của các kết hợp (conjunctions) này. Cây quyết định trong đó biến đích lấy các giá trị liên tục (thường là số thực) được gọi là cây hồi quy. Cây quyết định là một trong những thuật toán học máy phổ biến nhất nhờ tính dễ hiểu và đơn giản của chúng.



$$\begin{aligned}
 & [(\text{"music"} \text{ is present}) \wedge (\text{"singer"} \text{ is present})] \vee \\
 & [(\text{"music"} \text{ is absent}) \wedge (\text{"football"} \text{ is present})] \vee \\
 & [(\text{"music"} \text{ is absent}) \wedge (\text{"football"} \text{ is absent}) \wedge (\text{"My God"} \text{ is present})]
 \end{aligned}$$

Hình 34: Ví dụ về DT. DT được dùng để biểu diễn tri thức về thói quen xem các chương trình truyền hình của một người.

3.2.2 ID3

Ý tưởng

Trong phần này, chúng ta sẽ làm quen với một thuật toán xây dựng cây quyết định ra đời từ rất sớm. ID3 (Iterative Dichotomiser 3) do Ross Quinlan đề xuất năm 1986, là một thuật toán xây dựng/học cây quyết định được áp dụng cho các bài toán phân loại mà tất cả các thuộc tính đều có kiểu định danh/phạm trù.

Trong ID3, chúng ta cần xác định thứ tự của thuộc tính cần được xem xét tại mỗi bước. Với các bài toán có nhiều thuộc tính và mỗi thuộc tính có nhiều giá trị khác nhau, việc tìm được cây tối ưu thường là không khả thi. Thay vào đó, một phương pháp đơn giản thường được sử dụng là tại mỗi bước, một thuộc tính tốt nhất sẽ được chọn ra dựa trên một tiêu chuẩn nào đó. Việc chọn ra thuộc tính tốt nhất ở mỗi bước như thế này được gọi là cách chọn tham lam (greedy). Cách chọn này có thể không phải là tối ưu, nhưng trực giác cho chúng ta thấy rằng cách làm này sẽ gần với cách làm tối ưu. Ngoài ra, cách làm này khiến cho bài toán cần giải quyết trở nên đơn giản hơn.

ID3 thực hiện tìm kiếm tham lam trên không gian các cây quyết định để xây dựng một cây quyết định theo chiến lược top-down, bắt đầu từ nút gốc. Cụ thể:

- Ở mỗi nút, chọn thuộc tính kiểm tra tốt nhất - là thuộc tính có khả năng phân loại tốt nhất đối với các mẫu học gắn với nút đó.
- Tạo mới một cây con của nút hiện tại cho mỗi giá trị có thể của thuộc tính kiểm tra, và tập học sẽ được tách ra thành các tập con tương ứng với cây con vừa tạo.

Quá trình phát triển cây quyết định sẽ tiếp tục cho đến khi:

- Cây quyết định phân loại hoàn toàn các mẫu học, hoặc
- Tất cả các thuộc tính đã được sử dụng

Trong quá trình xây dựng một cây quyết định, với mỗi thuộc tính được chọn, ta chia dữ liệu tại nút đang xét vào các nút con tương ứng với tất cả các giá trị có thể của thuộc tính đó rồi tiếp tục áp dụng phương pháp này cho mỗi nút con. Do vậy, mỗi thuộc tính chỉ được phép xuất hiện tối đa 1 lần đối với bất kỳ một đường đi nào trong cây.

Giải thuật ID3

ID3_alg(Training_Set, Class_Labels, Attributes)

Tạo nút Root của cây quyết định

If tất cả các ví dụ của Training_Set thuộc cùng lớp c, Return Cây quyết định có nút Root được gắn với (có nhãn) lớp c

If Tập thuộc tính Attributes là rỗng, Return Cây quyết định có nút Root được gắn với nhãn lớp ≡ **Majority_Class_Label**(Training_Set)

A ← Thuộc tính trong tập Attributes có khả năng phân loại “tốt nhất” đối với Training_Set

Thuộc tính kiểm tra cho nút Root ← A

For each Giá trị có thể v của thuộc tính A

Bổ sung một nhánh cây mới dưới nút Root, tương ứng với trường hợp: “Giá trị của A là v”

Xác định Training_Set_v = {ví dụ x | x ⊆ Training_Set, x_A=v}

If (Training_Set_v là rỗng) Then

 Tạo một nút lá với nhãn lớp ≡ **Majority_Class_Label**(Training_Set)

 Gắn nút lá này vào nhánh cây mới vừa tạo

Else Gắn vào nhánh cây mới vừa tạo một cây con sinh ra bởi **ID3_alg**(Training_Set_v, Class_Labels, {Attributes \ A})

Return Root

Hình 35: Giải thuật ID3.

Để thực hiện ID3, tại nút đang xét, chúng ta cần chọn được thuộc tính có khả năng phân loại tốt nhất đối với các mẫu học gắn với nút đó để chia dữ liệu vào các nút con của nó. Bằng trực giác, một phép phân chia là tốt nhất nếu dữ liệu trong mỗi nút con hoàn toàn thuộc vào một lớp—khi đó nút con này có thể được coi là một nút lá, tức ta không cần phân chia thêm nữa. Nếu dữ liệu trong các nút con vẫn lẩn vào các lớp khác nhau theo tỉ lệ lớn, ta coi rằng phép phân chia đó chưa thực sự tốt. Từ nhận xét này, ta cần có một hàm số đo độ thuần khiết (purity) của một phép phân chia. Hàm số này sẽ cho giá trị thấp nhất nếu dữ liệu trong mỗi nút con nằm trong cùng một lớp (tinh khiết nhất), và cho giá trị cao nếu mỗi nút con có chứa dữ liệu thuộc nhiều lớp khác nhau. Entropy là một độ đo trong Lý thuyết thông tin được sử dụng để đo mức độ hỗn tạp của một tập và Entropy=0 nếu tất cả các mẫu thuộc cùng một lớp. Vì vậy, thuộc tính có khả năng phân loại tốt nhất đối với các mẫu học gắn với nút đang xét là thuộc tính sao cho nếu dùng thuộc tính đó để phân chia, entropy sẽ giảm đi một lượng lớn nhất. Chúng ta gọi Information Gain của một thuộc tính đối với một tập S các mẫu dữ liệu là độ đo dùng để đo mức độ giảm Entropy hay lượng thông tin (trung bình) bị mất nếu chia S theo tất cả các giá trị có thể của thuộc tính đó.

Information Gain của thuộc tính A đối với tập S được tính như sau:

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

trong đó $Values(A)$ là tập các giá trị có thể của thuộc tính A, và

$$S_v = \{x : x \in S \text{ và thuộc tính } A \text{ trong } x \text{ có giá trị là } v\}$$

Trong công thức trên, thành phần thứ 2 thể hiện giá trị Entropy sau khi tập S được phân chia bởi các giá trị của thuộc tính A.

Chiến lược tìm kiếm của giải thuật ID3

ID3 thực hiện tìm kiếm tham lam trên không gian các cây quyết định để xây dựng một cây quyết định theo chiến lược top-down, bắt đầu từ nút gốc. Do vậy, ID3 chỉ đảm bảo tìm được lời giải tối ưu cục bộ (locally optimal solution) chứ không đảm bảo tìm được lời giải tối ưu tổng thể (globally optimal solution). Đặc biệt, một khi một thuộc tính được chọn là thuộc tính kiểm tra cho một nút, ID3 không bao giờ cân nhắc lại lựa chọn này. ID3 luôn chọn cây quyết định phù hợp đầu tiên tìm thấy trong quá trình tìm kiếm của nó và ưu tiên các cây quyết định đơn giản (chiều cao cây thấp) trong đó một thuộc tính có giá trị Information Gain càng lớn thì sẽ là thuộc tính kiểm tra của một nút càng gần nút gốc.

Ví dụ

Để mọi thứ được rõ ràng hơn, chúng ta cùng xem ví dụ với dữ liệu huấn luyện được cho trong Bảng dưới đây. Bảng dữ liệu này được lấy từ cuốn sách D Mining: Practical Machine Learning Tools and Techniques, trang 11. Đây là một bảng dữ liệu được sử dụng rất nhiều trong các bài giảng về cây quyết định. Bảng dữ liệu này mô tả mối quan hệ giữa thời tiết trong 14 ngày (bốn cột đầu, không tính cột id) và việc một người chơi (không chơi) tennis (cột cuối cùng). Nói cách khác, ta phải dự đoán giá trị ở cột cuối cùng nếu biết giá trị của bốn cột còn lại.

Theo giải thuật ID3, tại nút gốc, chúng ta cần xác định được thuộc tính nào trong số Outlook, Temperature, Humidity, Wind nên được chọn là thuộc tính kiểm tra? Chúng ta hãy tính giá trị Information Gain của 4 thuộc tính trên đối với tập học S các mẫu từ bảng dữ liệu trên. Cụ thể, chúng ta sẽ tính $Gain(S, Wind)$?

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Thuộc tính Wind có 2 giá trị có thể là Weak và Strong. $\mathcal{S} = \{9 \text{ ví dụ lớp Yes và } 5 \text{ ví dụ lớp No}\}$. Gọi

$\mathcal{S}_{Weak} = \{6 \text{ ví dụ lớp Yes và } 2 \text{ ví dụ lớp No: giá trị của thuộc tính Wind là Weak}\}$,

$\mathcal{S}_{Strong} = \{3 \text{ ví dụ lớp Yes và } 3 \text{ ví dụ lớp No: giá trị của thuộc tính Wind là Strong}\}$.

Như vậy, chúng ta có:

$$\begin{aligned}
 Gain(\mathcal{S}, Wind) &= Entropy(\mathcal{S}) - \sum_{v \in \{Strong, Weak\}} \frac{|\mathcal{S}_v|}{|\mathcal{S}|} Entropy(\mathcal{S}_v) \\
 &= Entropy(\mathcal{S}) - \frac{8}{14} Entropy(\mathcal{S}_{Weak}) - \frac{6}{14} Entropy(\mathcal{S}_{Strong}) \\
 &= 0.94 - \frac{8}{14} \times 0.81 - \frac{6}{14} \times 1 \\
 &= 0.048
 \end{aligned}$$

Tương tự, chúng ta tính giá trị Gain của 3 thuộc tính còn lại {Outlook, Temperature, Humidity}:

- $Gain(\mathcal{S}, Outlook) = \dots = 0.246$
- $Gain(\mathcal{S}, Temperature) = \dots = 0.029$
- $Gain(\mathcal{S}, Humidity) = \dots = 0.151$
- Và $Gain(\mathcal{S}, Wind) = \dots = 0.048$

Vì vậy, Outlook được chọn là thuộc tính kiểm tra cho nút gốc!

3.2.3 Một số vấn đề của ID3

Vấn đề quá khớp

Trong các thuật toán cây học quyết định nói chung và ID3 nói riêng, nếu ta tiếp tục phân chia các nút chưa thuần khiết, ta sẽ thu được một cây mà mọi điểm trong tập huấn luyện đều được dự đoán đúng. Khi đó, cây có thể sẽ rất phức tạp với nhiều nút và nhiều nút lá chỉ có một vài điểm dữ liệu. Như vậy, nhiều khả năng quá khớp sẽ xảy ra.

Để tránh quá khớp, phương pháp ngừng học sớm có thể được sử dụng. Tại một nút, nếu một trong số các điều kiện sau đây xảy ra, ta không tiếp tục phân chia nút đó và coi nó là một nút lá:

- nếu nút đó có entropy bằng 0, tức mọi điểm trong nút đều thuộc một lớp.
- nếu nút đó có số phần tử nhỏ hơn một ngưỡng nào đó. Trong trường hợp này, ta chấp nhận có một số mẫu bị phân lớp sai để tránh quá khớp. Lớp cho nút lá này có thể được xác định dựa trên lớp chiếm đa số trong nút .
- nếu khoảng cách từ nút đó đến nút gốc đạt tới một giá trị nào đó. Việc hạn chế chiều sâu của cây này làm giảm độ phức tạp của cây và phần nào giúp tránh quá khớp.
- nếu tổng số nút lá vượt quá một ngưỡng nào đó.
- nếu việc phân chia nút đó không làm giảm entropy quá nhiều (gain nhỏ hơn một ngưỡng nào đó).

Ngoài phương pháp trên, một phương pháp phổ biến khác được sử dụng để tránh quá khớp là cắt tỉa (pruning).

Cắt tỉa là một kỹ thuật hiệu chỉnh để tránh quá khớp cho cây quyết định nói chung. Trong phương pháp cắt tỉa, một cây quyết định sẽ được xây dựng tối khi mọi điểm trong tập luyện (training set) đều được phân lớp đúng. Sau đó, các nút lá có chung một nút trong sẽ được cắt tỉa và nút trong đó trở thành một nút lá, với lớp tương ứng với lớp chiếm đa số trong số các mẫu được phân vào nút đó. Việc cắt tỉa cây quyết định này có thể được xác định dựa vào các cách sau:

1. Dựa vào một tập tối ưu (validation set): Trước tiên, tập luyện được tách ra thành một tập luyện nhỏ hơn và một tập tối ưu. Cây quyết định được xây dựng trên tập luyện cho tới khi mọi điểm trong tập luyện được phân lớp đúng. Sau đó, đi ngược từ các nút lá, cắt tỉa các nút anh em của nó và giữ lại nút bố mẹ nếu độ chính xác trên validation set được cải thiện. Khi nào độ chính xác trên tập tối ưu không được cải thiện nữa, quá trình cắt tỉa dừng lại. Phương pháp này còn được gọi là reduced error pruning.
2. Dựa vào toàn bộ tập luyện. Trong phương pháp này, ta không tách tập luyện ban đầu ra mà sử dụng toàn bộ dữ liệu trong tập này cho việc xây dựng cây quyết định. Một ví dụ cho việc này là cộng thêm một đại lượng hiệu chỉnh vào hàm mất mát. Đại lượng hiệu chỉnh L sẽ lớn nếu số nút lá là lớn. Việc tối ưu có thể được thực hiện thông qua cắt tỉa như sau. Trước hết, xây dựng một cây quyết định mà mọi điểm trong tập huấn luyện đều được phân loại đúng (tất cả các entopy của các nút lá bằng 0). Lúc này loss gốc bằng 0 nhưng L có thể lớn do số nút lá là lớn. Sau đó, ta có thể tỉa dần các nút lá sao cho L giảm. Việc cắt tỉa được lặp lại đến khi L không thể giảm được nữa.

Ngoài phương pháp cắt tỉa trên cây quyết định như trên, chúng ta có thể thực hiện cắt tỉa các luật quyết định. Phương pháp này còn được gọi là Rule post-pruning. Trong phương pháp cắt tỉa luật này, một cây quyết định được phát triển hoàn toàn phù hợp với tập huấn luyện. Sau đó, chúng ta chuyển biểu diễn cây quyết định học được thành một tập các luật tương ứng, có nghĩa là chúng ta tạo một luật cho mỗi đường đi từ nút gốc đến một nút lá. Tiếp theo, chúng ta rút gọn hay tổng quát hóa mỗi luật một cách độc lập với các luật khác bằng cách loại bỏ bất kỳ điều kiện nào trong luật giúp mang lại sự cải thiện về hiệu quả phân loại của luật đó. Cuối cùng, chúng ta sắp xếp các luật đã rút gọn theo khả năng hay hiệu quả phân loại của luật, và sử dụng thứ tự này cho việc phân loại các mẫu trong tương lai.

Vấn đề lựa chọn thuộc tính kiểm tra

Giải thuật ID3 sử dụng Gain để lựa chọn thuộc tính kiểm tại một nút và luôn ưu tiên các cây quyết định có chiều cao cây thấp trong đó một thuộc tính có giá trị Gain càng lớn thì sẽ là thuộc tính kiểm tra của một nút càng gần nút gốc. Rõ ràng giá trị Gain của một thuộc tính có nhiều giá trị hơn sẽ cao hơn so với các thuộc tính có ít giá trị.

Nếu một thuộc tính có thể nhận rất nhiều giá trị, cây quyết định thu được có thể sẽ có rất nhiều nút. Chẳng hạn, thuộc tính Day trong bảng dữ liệu trên có số lượng rất lớn các giá trị có thể. Khi đó, thuộc tính này sẽ có giá trị Gain cao nhất và thuộc tính này có thể phân loại hoàn hảo toàn bộ tập huấn luyện, cụ thể thuộc tính này phân chia tập học thành rất nhiều các tập con có kích thước 1 mẫu. Thuộc tính này được chọn là thuộc tính kiểm tra ở nút gốc của cây quyết định chỉ có mức độ sâu bằng 1, nhưng rất rộng, rất nhiều phân nhánh.

Để giảm ảnh hưởng của các thuộc tính có rất nhiều giá trị, thay vì sử dụng Gain, chúng ta sẽ sử dụng một độ đo khác được là Gain Ratio:

$$GainRatio(\mathcal{S}, A) = \frac{Gain(\mathcal{S}, A)}{SplitInformation(\mathcal{S}, A)}$$

trong đó $SplitInformation(\mathcal{S}, A) = -\sum_{v \in Values(A)} \frac{|\mathcal{S}_v|}{|\mathcal{S}|} \log_2 \frac{|\mathcal{S}_v|}{|\mathcal{S}|}$.

Vấn đề thuộc tính liên tục

Khi một thuộc tính nhận giá trị liên tục, chẳng hạn temperature không còn thuộc {hot, mild, cool} nữa mà là các giá trị thực liên tục, chúng ta vẫn có một cách để áp dụng ID3. Ta có thể rời rạc hoá các thuộc tính liên tục để tạo thành các thuộc tính có giá trị rời rạc bằng cách chia khoảng giá trị liên tục của thuộc tính này thành một tập các khoảng (intervals) không giao nhau, mỗi khoảng có số lượng mẫu dữ liệu tương đương, hoặc cũng có thể dùng các thuật toán phân cụm (clustering) đơn giản cho một thuộc tính dữ liệu để chia thuộc tính thành các cụm nhỏ. Lúc này, thuộc tính liên tục được chuyển về thuộc tính dạng định danh (categorical).

Hạn chế lớn nhất của ID3 và cây quyết định nói chung là việc nếu một điểm dữ liệu mới rơi vào nhầm nhánh ở ngay những lần phân chia đầu tiên, kết quả cuối cùng sẽ khác đi rất nhiều. Việc rơi vào nhầm nhánh này rất dễ xảy ra trong trường hợp thuộc tính liên tục được chia thành nhiều nhóm nhỏ, vì hai điểm có thuộc tính tương ứng rất gần nhau có thể rơi vào hai nhóm khác nhau.

3.2.4 Rừng ngẫu nhiên (Random Forests)

Ý tưởng

Khi xây dựng cây quyết định, nếu để độ sâu tùy ý thì cây sẽ phân loại đúng hết các

dữ liệu trong tập học, dẫn đến mô hình có thể dự đoán tệ trên tập kiểm thử. Khi đó mô hình bị quá khớp.

Thuật toán Random Forests gồm nhiều cây quyết định, mỗi cây quyết định đều có những yếu tố ngẫu nhiên:

1. Lấy ngẫu nhiên dữ liệu để xây dựng cây quyết định.
2. Lấy ngẫu nhiên các thuộc tính để đặt vào một đinh.

Do mỗi cây quyết định trong thuật toán học Random Forests không dùng tất cả dữ liệu luyện, cũng như không dùng tất cả các thuộc tính của dữ liệu để xây dựng cây nên mỗi cây có thể sẽ dự đoán không tốt, khi đó mỗi mô hình cây quyết định không bị quá khớp mà có thể bị kém khớp, hay nói cách khác là mô hình có high bias. Tuy nhiên, kết quả cuối cùng của thuật toán Random Forests lại tổng hợp từ nhiều cây quyết định, thế nên thông tin từ các cây sẽ bổ sung thông tin cho nhau, dẫn đến mô hình có độ lệch và phương sai bé, hay mô hình có kết quả dự đoán tốt.

Random Forest là một thuật toán học máy phổ biến thuộc về phương pháp học có giám sát. Nó có thể được sử dụng cho cả bài toán Phân loại và Hồi quy trong Học máy. Nó dựa trên khái niệm học kết hợp, là một quá trình kết hợp nhiều bộ phân loại để giải quyết một vấn đề phức tạp và để cải thiện hiệu suất của mô hình.

Rừng ngẫu nhiên cũng là một thuật toán học máy linh hoạt, dễ sử dụng, tạo ra kết quả tuyệt vời ngay cả khi không điều chỉnh siêu tham số. Nó cũng là một trong những thuật toán được sử dụng nhiều nhất, do tính đơn giản và đa dạng của nó. Như tên gọi của phương pháp học máy này, “Rừng ngẫu nhiên là một bộ phân loại chứa một số cây quyết định trên các tập con khác nhau của tập dữ liệu cũng như các tập con thuộc tính khác nhau và lấy giá trị dự đoán trung bình để cải thiện độ chính xác dự đoán của tập dữ liệu đó.”

Thuật toán Random Forests

Giả sử tập dữ liệu huấn luyện \mathbf{D} gồm có n mẫu dữ liệu và mỗi dữ liệu có d thuộc tính. Chúng ta học K cây quyết định như sau:

1. Tạo K cây, mỗi cây được sinh ra như sau:
 - (a) Xây dựng một tập con \mathbf{D}_i bằng cách lấy ngẫu nhiên (có trùng lặp) từ \mathbf{D} .
 - (b) Học cây thứ i từ \mathbf{D}_i như sau:

- (c) Tại mỗi nút trong quá trình xây dựng cây:
- chọn ngẫu nhiên một tập con các thuộc tính
 - phân nhánh cây dựa trên tập thuộc tính đó.
- (d) Cây này sẽ được sinh ra với cỡ lớn nhất, không dùng cắt tỉa.
2. Mỗi phán đoán về sau thu được bằng cách lấy trung bình các phán đoán từ tất cả các cây.

Để xây dựng một tập con D_i từ D , chúng ta lấy ngẫu nhiên n dữ liệu từ tập dữ liệu huấn luyện với kĩ thuật Bootstrapping, hay còn gọi là lấy mẫu ngẫu nhiên có lặp lại (random sampling with replacement), có nghĩa là, chúng ta lấy mẫu được 1 mẫu dữ liệu thì không bỏ dữ liệu đấy ra mà vẫn giữ lại trong tập dữ liệu ban đầu, rồi tiếp tục lấy mẫu cho tới khi có đủ n mẫu dữ liệu. Khi dùng kĩ thuật này thì tập mới có thể có những mẫu dữ liệu bị trùng nhau.

Do quá trình xây dựng mỗi cây quyết định đều có yếu tố ngẫu nhiên (random) nên kết quả là các cây quyết định trong thuật toán Random Forests có thể khác nhau. Thuật toán Random Forests sẽ bao gồm nhiều cây quyết định, mỗi cây được xây dựng dùng thuật toán Cây quyết định trên tập dữ liệu khác nhau và dùng tập thuộc tính khác nhau. Sau đó kết quả dự đoán của thuật toán Random Forests sẽ được tổng hợp từ các cây quyết định đã xây dựng được.

Khi dùng thuật toán Random Forests, chúng ta nên chú ý đến các siêu tham số như: số lượng cây quyết định sẽ xây dựng, số lượng thuộc tính dùng để xây dựng cây,...

3.3 Máy véc-tơ hỗ trợ (SVM)

Máy vectơ hỗ trợ (Support vector machine - SVM) được đề xuất bởi V. Vapnik và các đồng nghiệp của ông vào những năm 1970s ở Nga, và sau đó đã trở nên nổi tiếng và phổ biến vào những năm 1990s. SVM là một phương pháp phân lớp tuyến tính (linear classifier), với mục đích xác định một siêu phẳng (hyperplane) để phân tách hai lớp của dữ liệu. Ví dụ: lớp có nhãn dương (positive) và lớp có nhãn âm (negative). Sau đó, các hàm nhân (kernel functions), cũng được gọi là các hàm biến đổi (transformation functions), được dùng cho các trường hợp phân lớp phi tuyến. SVM có nền tảng lý thuyết chặt chẽ, được xây dựng dựa trên lý thuyết toán học và thống kê.

Về mặt thực nghiệm, SVM thường được coi là một phương pháp tốt đối với những bài toán phân lớp có không gian rất nhiều chiều. Điều này có lợi ích đặc biệt khi dữ liệu đặc trưng của các đối tượng là một tập rất lớn các thuộc tính. Đặc biệt, SVM đã được biết đến là một trong số các phương pháp phân lớp tốt nhất đối với các bài toán phân lớp văn bản (text classification). Trong phân loại văn bản, dữ liệu thường có số chiều lớn, với mỗi từ hoặc đặc trưng là một chiều. SVM hiệu quả trong việc xử lý không gian chiều cao này và có khả năng tạo ra siêu phẳng phân loại tốt giữa các danh mục văn bản khác nhau.

3.3.1 Mô hình hóa

Đầu tiên, xem xét bài toán phân loại nhị phân có tập huấn luyện gồm r điểm (\mathbf{x}_i, y_i) với $i = \{1, \dots, r\}$ trong đó $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in})^\top$ là vectơ n chiều, $y_i \in \{-1, 1\}$ là lớp của điểm \mathbf{x}_i . Ở đây, chúng ta có hai lớp: lớp dương ($y_i = 1$) và lớp âm ($y_i = -1$).

Giả thiết rằng dữ liệu ban đầu của chúng ta là có thể phân tách tuyến tính, chúng ta muốn tìm một siêu phẳng có thể phân tách các lớp âm và lớp dương. Siêu phẳng có dạng:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \quad (23)$$

trong đó, \mathbf{w} là vectơ trọng số, b là hệ số bias, $\langle \mathbf{w}, \mathbf{x} \rangle$ là tích vô hướng của \mathbf{x} và \mathbf{w} . Siêu phẳng phân tách 2 lớp dương và âm phải thỏa mãn ràng buộc:

$$y_i = \begin{cases} +1, & \text{nếu } \langle \mathbf{x}_i, \mathbf{w} \rangle + b \geq +1 \\ -1, & \text{nếu } \langle \mathbf{x}_i, \mathbf{w} \rangle + b \leq -1 \end{cases} \quad (24)$$

Siêu phẳng (H_0) phân tách lớp dương và lớp âm có dạng: $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$. H_0 còn được gọi là ranh giới quyết định (decision boundary). Thực tế, khi dữ liệu học có thể phân tách tuyến tính, có vô số các siêu phẳng phân tách (Hình 36). Câu hỏi đặt ra là siêu phẳng phân tách nào là tốt nhất? SVM lựa chọn mặt siêu phẳng phân tách có **lề (margin) lớn nhất**. Lý thuyết học máy đã chỉ ra rằng *một mặt siêu phẳng phân tách như thế sẽ tối thiểu hóa giới hạn lỗi (phân lớp) mắc phải (so với mọi siêu phẳng khác)*.

Chúng ta sẽ xem xét chi tiết khái niệm cực đại hóa mức lề. Giả sử rằng tập dữ liệu huấn luyện có thể phân tách được một cách tuyến tính. Xét một quan sát của lớp dương

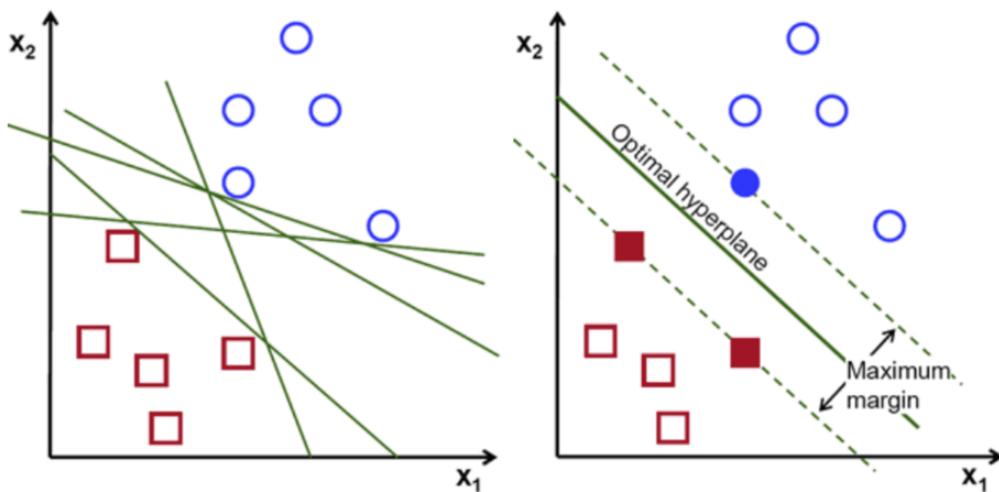
$(\mathbf{x}^+, 1)$ và một quan sát của lớp âm $(\mathbf{x}^-, -1)$ gần nhất đối với siêu phẳng phân tách $(H_0)(\langle \mathbf{w}, \mathbf{x} \rangle + b = 0)$. Định nghĩa **2 siêu phẳng lề** song song với nhau:

- H_+ đi qua \mathbf{x}^+ , và song song với H_0
- H_- đi qua \mathbf{x}^- , và song song với H_0

Phương trình mô tả cho H_+ và H_- được viết như sau:

$$H_+ : \langle \mathbf{w}, \mathbf{x} \rangle + b = 1$$

$$H_- : \langle \mathbf{w}, \mathbf{x} \rangle + b = -1$$



Hình 36: Hình minh họa các siêu phẳng phân tách và siêu phẳng phân tách tối ưu khi cựu đại hóa mức lề.

Mức lề (margin) là khoảng cách giữa 2 siêu phẳng lề H_+ và H_- . Chúng ta có một số khái niệm:

- d_+ là khoảng cách giữa H_+ và H_0
- d_- là khoảng cách giữa H_- và H_0
- $(d_+ + d_-)$ là mức lề

Trong không gian vectơ, **khoảng cách** từ một điểm \mathbf{x}_i đến siêu phẳng $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ là:

$$\frac{|\langle \mathbf{w}, \mathbf{x}_i \rangle + b|}{\|\mathbf{w}\|} \quad (25)$$

trong đó, $\|\mathbf{w}\|$ là độ dài của vectơ \mathbf{w} :

$$\|\mathbf{w}\| = \sqrt{\langle \mathbf{w}, \mathbf{w} \rangle} = \sqrt{\sum_{i=1}^n w_i^2} \quad (26)$$

Áp dụng công thức trên, chúng ta có thể tính được d_+ : khoảng cách từ \mathbf{x}^+ đến ($\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$):

$$d_+ = \frac{|\langle \mathbf{w}, \mathbf{x}^+ \rangle + b|}{\|\mathbf{w}\|} = \frac{|1|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (27)$$

Tương tự, chúng ta tính toán d_- : khoảng cách từ \mathbf{x}^- đến ($\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$):

$$d_- = \frac{|\langle \mathbf{w}, \mathbf{x}^- \rangle + b|}{\|\mathbf{w}\|} = \frac{|-1|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (28)$$

Từ đó, mức lề có thể tính được như sau:

$$\text{margin} = d_+ + d_- = \frac{2}{\|\mathbf{w}\|} \quad (29)$$

Việc học trong SVM tương đương với giải quyết **bài toán cực tiểu hóa có ràng buộc**. Chúng ta phải cực tiểu hóa: $\frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2}$ với điều kiện:

$$\begin{cases} \langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq 1, \text{ nếu } y_i = 1 \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b \leq -1, \text{ nếu } y_i = -1 \end{cases}$$

Bài toán này tương đương với:

$$\begin{aligned} \text{Cực tiểu hóa hàm:} \quad & \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} \\ \text{với điều kiện:} \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \forall i \in \{1, \dots, r\} \end{aligned} \quad (30)$$

Đây là bài toán học ra hàm tuyến tính trong SVM.

3.3.2 Tối ưu hóa có ràng buộc

Trong phần này, chúng ta xem xét lại vấn đề tối ưu hóa có ràng buộc và áp dụng cho bài toán học mức lề cực đại trong SVM.

Bài toán cực tiểu hóa có ràng buộc đẳng thức: Cực tiểu hóa $f(x)$, với điều kiện $g(x)=0$.

Điều kiện cần để x_0 là một lời giải:

$$\begin{cases} \frac{\partial}{\partial x}(f(x) + \alpha g(x)) \Big|_{x=x_0} = 0 \\ g(x) = 0 \end{cases}$$

với α là một hệ số nhân (multiplier) Lagrange. Trong trường hợp có nhiều ràng buộc đẳng thức $g_i(x)=0$ ($i=1..r$), cần một hệ số nhân Lagrange cho mỗi ràng buộc:

$$\begin{cases} \frac{\partial}{\partial x}(f(x) + \sum_{i=0}^r \alpha_i g_i(x)) \Big|_{x=x_0} = 0 \\ g_i(x) = 0 \end{cases}$$

Trong khi đó, bài toán cực tiểu hóa có các ràng buộc bất đẳng thức: Cực tiểu hóa $f(x)$, với các điều kiện $g_i(x) \leq 0$.

Điều kiện cần để x_0 là một lời giải:

$$\begin{cases} \frac{\partial}{\partial x}(f(x) + \sum_{i=0}^r \alpha_i g_i(x)) \Big|_{x=x_0} = 0 & \text{với } \alpha_i \geq 0 \\ g_i(x) \leq 0 \end{cases}$$

Hàm: $L = f(x) + \sum_{i=1}^r \alpha_i g_i(x)$ được gọi là hàm Lagrange.

Học SVM được quy về giải bài toán cực tiểu hóa có ràng buộc bất đẳng thức. Biểu thức Lagrange

$$L_p(\mathbf{w}, b, \alpha) = \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} - \sum_{i=1}^r \alpha_i [y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1] \quad (31)$$

trong đó $\alpha_i \geq 0$ là các hệ số nhân Lagrange.

Lý thuyết tối ưu chỉ ra rằng một lời giải tối ưu cho (31) phải thỏa mãn các điều kiện nhất định, được gọi là **các điều kiện Karush-Kuhn-Tucker** (là các điều kiện cần, nhưng không phải là các điều kiện đủ). Các điều kiện Karush-Kuhn-Tucker đóng vai

trò trung tâm trong cả lý thuyết và ứng dụng của lĩnh vực tối ưu có ràng buộc. Tập điều kiện Karush-Kuhn-Tucker cho bài toán tối ưu (31):

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^r \alpha_i y_i \mathbf{x}_i = 0 \quad (32)$$

$$\frac{\partial L_p}{\partial b} = - \sum_{i=1}^r \alpha_i y_i = 0 \quad (33)$$

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 \geq 0, \forall \mathbf{x}_i \quad (i = 1..r) \quad (34)$$

$$\alpha_i \geq 0, \quad i = 1..r \quad (35)$$

$$\alpha_i[y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1] = 0, \quad i = 1..r \quad (36)$$

Trong đó, (34) chính là tập các ràng buộc ban đầu. Điều kiện bổ sung (36) chỉ ra rằng chỉ những ví dụ (điểm dữ liệu) thuộc các mặt siêu phẳng lề (H_+ và H_-) mới có $\alpha_i \geq 0$ bởi vì với những ví dụ đó thì $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 = 0$. Những ví dụ này được gọi là **các vectơ hỗ trợ**. Trong đó, đối với các ví dụ khác thì $\alpha_i = 0$.

Trong trường hợp tổng quát, các điều kiện Karush-Kuhn-Tucker là *cần* đối với một lời giải tối ưu, nhưng *chưa đủ*. Tuy nhiên đối với SVM, bài toán cực tiểu hóa có hàm mục tiêu lồi (convex) và các ràng buộc tuyến tính, thì các điều kiện Karush-Kuhn-Tucker là cần và đủ đối với một lời giải tối ưu. Nhưng giải quyết bài toán tối ưu này vẫn là một nhiệm vụ khó khăn, do sự tồn tại của các ràng buộc bất đẳng thức. Phương pháp Lagrange giải quyết bài toán tối ưu hàm lồi dẫn đến một bài toán **đối ngẫu (dual)** của bài toán tối ưu. Bài toán đối ngẫu dễ giải quyết hơn so với bài toán tối ưu **ban đầu (primal)**.

Để thu được biểu thức đối ngẫu từ biểu thức ban đầu trong SVM, chúng ta gán giá trị bằng 0 đối với các đạo hàm bộ phận của biểu thức Lagrange trong (31) đối với các biến ban đầu \mathbf{w} và b . Sau đó, áp dụng các quan hệ thu được đối với biểu thức Lagrange. Nghĩa là áp dụng các biểu thức (32) và (33) vào biểu thức Lagrange ban đầu (31) để loại bỏ các biến ban đầu \mathbf{w} và b , chúng ta sẽ thu được biểu thức đối ngẫu L_D :

$$L_d(\alpha) = \sum_{i=1}^r \alpha_i - \frac{1}{2} \sum_{i,j=1}^r \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (37)$$

Cả hai biểu thức L_P và L_D đều là các biểu thức Lagrange. Chúng đều dựa trên cùng một hàm mục tiêu – nhưng với các ràng buộc khác nhau. Lời giải tìm được, bằng cách cực tiểu hóa L_P hoặc cực đại hóa L_D .

Bài toán tối ưu đối ngẫu cực đại hóa: $L_d(\alpha) = \sum_{i=1}^r \alpha_i - \frac{1}{2} \sum_{i,j=1}^r \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ với điều kiện:

$$\begin{cases} \sum_{i=1}^r \alpha_i y_i = 0 \\ \alpha_i \geq 0, \forall i = 1..r \end{cases}$$

Đối với hàm mục tiêu là hàm lồi và các ràng buộc tuyến tính, giá trị cực đại của L_D xảy ra tại cùng các giá trị của w , b và α_i giúp đạt được giá trị cực tiểu của L_P . Giải bài toán trên, ta thu được các hệ số nhân Lagrange α_i (các hệ số α_i này sẽ được dùng để tính w và b). Giải bài toán trên cần đến các phương pháp lặp (để giải quyết bài toán tối ưu hàm lồi bậc hai có các ràng buộc tuyến tính). Chi tiết các phương pháp này nằm ngoài phạm vi của bài giảng.

Sau khi giải bài toán tối ưu đối ngẫu thu được α , chúng ta có thể tính được các giá trị w^* và b^* . Gọi SV là tập các vectơ hỗ trợ. SV là tập con của tập r các ví dụ huấn luyện ban đầu và thỏa mãn $\alpha_i > 0$ với các vectơ hỗ trợ \mathbf{x}_i . Đồng thời, $\alpha_i = 0$ với các vectơ không phải vectơ hỗ trợ \mathbf{x}_i . Sử dụng biểu thức (32), ta có thể tính được giá trị w^*

$$w^* = \sum_{i=1}^r \alpha_i y_i \mathbf{x}_i = \sum_{\mathbf{x}_i \in SV} \alpha_i y_i \mathbf{x}_i \quad (38)$$

vì $\forall \mathbf{x}_i \notin SV: \alpha_i = 0$. Mặt khác, sử dụng biểu thức (36) và (bất kỳ) một vectơ hỗ trợ x_k , ta có: $\alpha_k [y_k (\langle w^*, \mathbf{x}_k \rangle + b^*) - 1] = 0$. Chú ý rằng $\alpha_k > 0$ với mọi vectơ hỗ trợ \mathbf{x}_k . Vì vậy: $y_k (\langle w^*, \mathbf{x}_k \rangle + b^*) - 1 = 0$. Từ đây, ta được: $b^* = \frac{1}{y_k} - \langle w^*, \mathbf{x}_k \rangle$. Từ đó, chúng ta thu được ranh giới quyết định phân lớp:

$$f(\mathbf{x}) = \langle w^*, \mathbf{x} \rangle + b^* = \sum_{\mathbf{x}_i \in SV} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^* = 0 \quad (39)$$

Khi làm việc với một ví dụ cần phân lớp \mathbf{z} , chúng ta cần tính giá trị:

$$\text{sign}(\langle \mathbf{w}^*, \mathbf{z} \rangle + b^*) = \text{sign}\left(\sum_{\mathbf{x}_i \in SV} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{z} \rangle + b^*\right) \quad (40)$$

Nếu biểu thức (40) trả về giá trị 1, thì ví dụ \mathbf{z} được phân vào lớp có nhãn dương (positive); ngược lại, được phân vào lớp có nhãn âm (negative). Việc phân lớp này chỉ phụ thuộc vào các vectơ hỗ trợ và chỉ cần giá trị tích vô hướng (tích trong) của 2 vectơ (chứ không cần biết giá trị của 2 vectơ đó).

3.3.3 Soft-margin SVM

Thực tế, trường hợp phân lớp tuyến tính và phân tách được là lý tưởng (ít xảy ra). Tập dữ liệu có thể chứa nhiễu, lỗi (vd: một số ví dụ được gán nhãn lớp sai). Đối với trường hợp không phân tách được, bài toán (30) không có lời giải.

Để làm việc với các dữ liệu chứa nhiễu, chúng ta cần nới lỏng các điều kiện lề (margin constraints) bằng cách sử dụng các biến bù $\xi_i \geq 0$:

- $\langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq 1 - \xi_i$ với các giá trị $y_i = 1$
- $\langle \mathbf{w}, \mathbf{x}_i \rangle + b \leq 1 - \xi_i$ với các giá trị $y_i = -1$

Với một ví dụ nhiễu/lỗi thì $\xi_i > 1$. Các điều kiện mới đối với trường hợp (phân lớp tuyến tính) không thể phân tách được:

$$\begin{cases} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \forall i = 1..r \\ \xi_i \geq 0, \forall i = 1..r \end{cases}$$

Chúng ta có thể tích hợp lỗi trong hàm tối ưu mục tiêu bằng cách gán giá trị chi phí (cost) cho các lỗi, và tích hợp chi phí này trong hàm mục tiêu mới. Cụ thể, chúng ta sẽ cực tiểu hóa:

$$\frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} + C \sum_{i=1}^r \xi_i \quad (41)$$

trong đó $C \geq 0$ là tham số xác định mức độ phạt (penalty degree) đối với các lỗi. C có

giá trị càng lớn, mức phạt càng cao với các lỗi. Chúng ta có bài toán:

$$\text{Tìm cực tiểu của hàm: } \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} + C \sum_{i=1}^r \xi_i \quad (42)$$

với điều kiện: $\begin{cases} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \forall i = 1..r \\ \xi_i \geq 0, \forall i = 1..r \end{cases}$

Bài toán tối ưu mới này được gọi là **Soft-margin SVM**. Giải bài toán này tương đương với việc cực tiểu hóa hàm:

$$\left[\frac{1}{r} \sum_{i=1}^r \max(0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)) \right] + \lambda \|\mathbf{w}\|_2^2 \quad (43)$$

trong đó $\max(0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))$ thường được gọi là Hinge loss và λ là tham số.

Chúng ta cũng xây dựng được biểu thức tối ưu Lagrange cho bài toán tối ưu có ràng buộc bất đẳng thức:

$$L_p = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{i=1}^r \xi_i - \sum_{i=1}^r \alpha_i [y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^r \mu_i \xi_i \quad (44)$$

trong đó $\alpha_i (\geq 0)$ và $\mu (\geq 0)$ là các hệ số nhân Lagrange. Tập điều kiện Karush-Kuhn-

Tucker cho bài toán này trình bày như sau:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^r \alpha_i y_i \mathbf{x}_i = 0 \quad (45)$$

$$\frac{\partial L_p}{\partial b} = - \sum_{i=1}^r \alpha_i y_i = 0 \quad (46)$$

$$\frac{\partial L_p}{\partial \xi_i} = C - \alpha_i - \mu_i = 0, \forall i = 1..r \quad (47)$$

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 + \xi_i \geq, \forall i = 1..r \quad (48)$$

$$\xi_i \geq 0 \quad (49)$$

$$\alpha_i \geq 0 \quad (50)$$

$$\mu_i \geq 0 \quad (51)$$

$$\alpha_i(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 + \xi_i) = 0 \quad (52)$$

$$\mu_i \xi_i = 0 \quad (53)$$

Giống như với trường hợp dữ liệu có thể phân tách được, chúng ta chuyển biểu thức Lagrange từ dạng ban đầu (primal formulation) về dạng đối ngẫu (dual formulation). Từ biểu thức (47), ta có: $C - \alpha_i - \mu_i = 0$ và bởi vì: $\mu_i \geq 0$ nên ta suy ra điều kiện $\alpha_i \leq C$.

Chúng ta có biểu thức đối ngẫu

$$L_D(\alpha) = \sum_{i=1}^r \alpha_i - \frac{1}{2} \sum_{i,j=1}^r \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (54)$$

với điều kiện:

$$\begin{cases} \sum_{i=1}^r \alpha_i y_j = 0 \\ 0 \leq \alpha_i \leq C, \forall i = 1..r \end{cases}$$

và ξ_i và các hệ số nhân Lagrange của chúng (μ_i) không xuất hiện trong biểu thức đối ngẫu. Đặc biệt, hàm mục tiêu giống hệt như đối với bài toán phân lớp tuyến tính phân tách được (separable linear classification). Khác biệt duy nhất là tập các ràng buộc mới: $\alpha_i \leq C$.

Bài toán đối ngẫu (54) được giải quyết bằng các phương pháp lặp (để giải quyết bài toán tối ưu hàm lồi bậc hai có các ràng buộc tuyến tính). Các giá trị (hệ số nhân

Lagrange) α_i được sử dụng để tính toán \mathbf{w}^* và b^* . \mathbf{w}^* được xác định sử dụng biểu thức (45). b^* được xác định sử dụng các điều kiện bổ sung Karush-Kuhn-Tucker trong (52) và (53). Cụ thể, từ (47) và (53), ta suy ra được: $\xi_i = 0$ nếu $\alpha_i < C$. Vì vậy, ta có thể sử dụng một ví dụ học x_k thỏa mãn điều kiện ($0 < \alpha_k < C$) và (52) (với $\xi_k = 0$) để tính toán b^* . Đến đây, việc tính toán b^* tương tự như với trường hợp phân lớp tuyến tính phân tách được.

Từ các biểu thức (47) tới (53), ta có thể suy ra :

Nếu $\alpha_i = 0$ thì $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$, và $\xi_i = 0$

Nếu $0 < \alpha_i < C$ thì $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$, và $\xi_i = 0$

Nếu $\alpha_i = C$ thì $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) < 1$, và $\xi_i > 0$

Biểu thức trên thể hiện một đặc điểm rất quan trọng của SVM. Lời giải được xác định dựa trên rất ít các giá trị α_i và rất nhiều ví dụ học nằm ngoài khoảng lề (margin area), chúng có giá trị α_i bằng 0. Các ví dụ nằm trên lề $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$ – chính là các vectơ hỗ trợ, thì có giá trị α_i khác không ($0 < \alpha_i < C$). Các ví dụ nằm trong khoảng lề ($y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) < 1$ - là các ví dụ nhiễu/lỗi), thì có giá trị $\alpha_i = C$. Nếu không có đặc điểm thưa thớt (sparsity) này, thì phương pháp SVM không thể hiệu quả đối với các tập dữ liệu lớn. Một điều đáng chú ý rằng: tham số C trong hàm tối ưu mục tiêu có thể ảnh hưởng rất lớn tới độ chính xác của thuật toán và thường được xác định bằng cách sử dụng một tập dữ liệu tối ưu (validation set).

Sau khi giải quyết bài toán tối ưu, chúng ta thu được ranh giới quyết định phân lớp là siêu phẳng:

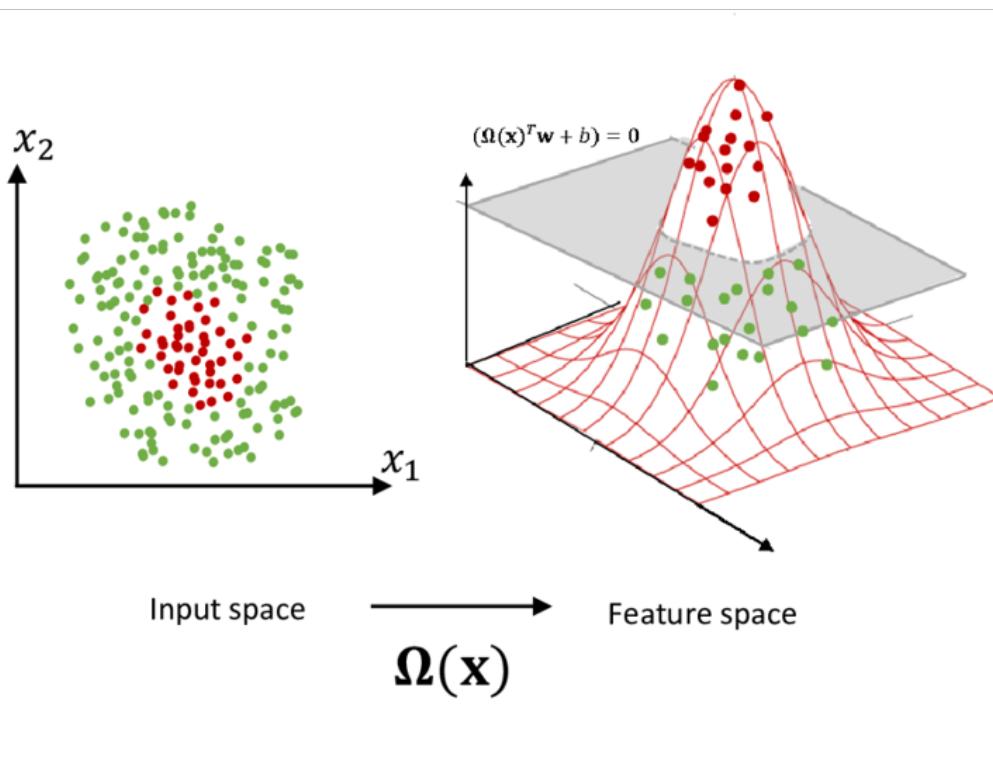
$$\langle \mathbf{w}^*, \mathbf{x} \rangle + b^* = \sum_{i=1}^r \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b^* = 0$$

Đối với một ví dụ cần phân loại \mathbf{z} , nó được phân loại bởi:

$$\text{sign}(\langle \mathbf{w}^*, \mathbf{z} \rangle + b^*)$$

3.3.4 Hàm nhân (kernel function)

Các công thức trong phương pháp SVM đòi hỏi tập dữ liệu phải có thể phân lớp tuyến tính (có thể có nhiều ít). Trong nhiều bài toán thực tế, thì các tập dữ liệu có thể là phân lớp phi tuyến (non-linearly separable). Phương pháp phân loại SVM phi tuyến (Non-linear SVM) có thể được thực hiện qua 2 bước. Bước 1, chúng ta chuyển đổi không gian biểu diễn đầu vào ban đầu sang một không gian khác (thường có số chiều lớn hơn nhiều). Trên không gian mới, dữ liệu được biểu diễn có thể phân lớp tuyến tính (linearly separable). Bước 2, chúng ta áp dụng lại các công thức và các bước như trong phương pháp phân lớp SVM tuyến tính.



Hình 37: Hình minh họa phép biến đổi không gian giúp dữ liệu có thể phân tách trên không gian mới.

Ý tưởng cơ bản của chuyển đổi không gian biểu diễn (Hình 37) là việc ánh xạ (chuyển đổi) biểu diễn dữ liệu từ không gian ban đầu \mathcal{X} sang một không gian khác \mathcal{F} bằng cách áp dụng một hàm ánh xạ phi tuyến Φ

$$\Phi : \mathcal{X} \longmapsto \mathcal{F}$$

$$\mathbf{x} \longmapsto \Phi(\mathbf{x})$$

Trong không gian đã chuyển đổi, tập các ví dụ học ban đầu $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_r, y_r)\}$

được biểu diễn (ánh xạ) tương ứng:

$$\{(\Phi(\mathbf{x}_1), y_1), (\Phi(\mathbf{x}_2), y_2), \dots, (\Phi(\mathbf{x}_r), y_r)\}$$

Sau quá trình chuyển đổi không gian biểu diễn, bài toán tối ưu:

$$\text{Cực tiểu hoá: } L_p = \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} + C \sum_{i=1}^r \xi_i \quad (55)$$

$$\text{Với điều kiện: } \begin{cases} y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \forall i = 1..r \\ \xi_i \geq 0, \forall i = 1..r \end{cases} \quad (56)$$

và bài toán (tối ưu) đối ngẫu tương ứng:

$$\text{Cực đại hoá: } L_D = \sum_{i=1}^r \alpha_i - \frac{1}{2} \sum_{i,j=1}^r \alpha_i \alpha_j y_i y_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \quad (57)$$

$$\text{Với điều kiện: } \begin{cases} \sum_{i=1}^r \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, \forall i = 1..r \end{cases} \quad (58)$$

Ranh giới quyết định phân lớp là siêu phẳng phân tách:

$$f(\mathbf{z}) = \langle \mathbf{w}^*, \Phi(\mathbf{z}) \rangle + b^* = \sum_{i=1}^r \alpha_i y_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{z}) \rangle + b^* = 0 \quad (59)$$

Tuy nhiên, việc chuyển đổi không gian một cách trực tiếp có thể gặp vấn đề về số chiều không gian quá lớn (curse of dimensionality). Ngay cả với một không gian ban đầu có số chiều không lớn, một hàm chuyển đổi (ánh xạ) thích hợp có thể trả về một không gian mới có số chiều rất lớn. Chú ý, “thích hợp” ở đây mang ý nghĩa là hàm chuyển đổi cho phép xác định không gian mới mà trong đó tập dữ liệu có thể phân lớp tuyến tính. Chúng ta phải đổi mặt với chi phí tính toán quá lớn đối với việc chuyển đổi không gian trực tiếp hoặc không tìm được phép biến đổi đủ tốt. Rõ ràng, dữ liệu dù thu thập được lớn đến đâu thì cũng là quá nhỏ so với không gian của chúng.

Trong biểu thức đối ngẫu và trong biểu thức siêu phẳng phân tách, việc xác định trực tiếp giá trị $\Phi(\mathbf{x})$ và $\Phi(\mathbf{z})$ là không cần thiết. Chúng ta chỉ cần tính giá trị tích vô hướng vectơ $\langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle$. Nếu có thể tính được tích vô hướng vectơ $\langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle$ trực

tiếp từ các vectơ \mathbf{x} và \mathbf{z} , thì chúng ta không cần phải xác định (không cần biết) vectơ đặc trưng (trong không gian sau chuyển đổi) $\Phi(\mathbf{x})$, và hàm chuyển đổi (ánh xạ) Φ . Trong phương pháp SVM, mục tiêu này đạt được thông qua việc sử dụng các hàm nhân (kernel functions), được ký hiệu là K :

$$K(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle \quad (60)$$

Một số hàm nhân thường dùng:

- Đa thức

$$K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + \Phi)^d; \text{ trong đó } \Phi \in \mathbb{R}, d \in \mathbb{N}$$

- Gaussian RBF (Gaussian radial basis function)

$$K(\mathbf{x}, \mathbf{z}) = e^{\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}}; \text{ trong đó : } \sigma > 0$$

- Sigmoid kernel

$$K(\mathbf{x}, \mathbf{z}) = \tanh(\beta \langle \mathbf{x}, \mathbf{z} \rangle + \lambda);$$

trong đó : $\beta, \lambda \in \mathbb{R}$

3.4 Học dựa trên xác suất

Trong mục này chúng ta sẽ tìm hiểu một cách tiếp cận hoàn toàn khác để giải quyết bài toán học. Đó là các mô hình xác suất (probabilistic models).

3.4.1 Giới thiệu

Đối với một bài toán học, chúng ta cần tìm một hàm từ một tập dữ liệu \mathcal{D} đã có. Tập dữ liệu này chứa một vài mâu rác về đầu vào (và đầu ra) của một hàm y^* nào đó mà chúng ta không biết. Cách làm thường thấy trong ML gồm hai bước chính:

- Chọn một kiểu mô hình $\mathcal{H} = \{f(\mathbf{x}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$. Nghĩa là chúng ta đã có một **giả thuyết** (assumption) ngầm rằng hàm y^* cần tìm có thể được xấp xỉ tốt bởi một hàm nào đó trong \mathcal{H} . Trong đó $\boldsymbol{\theta}$ là tham số để mô tả mỗi hàm trong \mathcal{H} .
- Huấn luyện để tìm ra một hàm $h \in \mathcal{H}$ từ tập \mathcal{D} , mà có thể xấp xỉ tốt y^* .

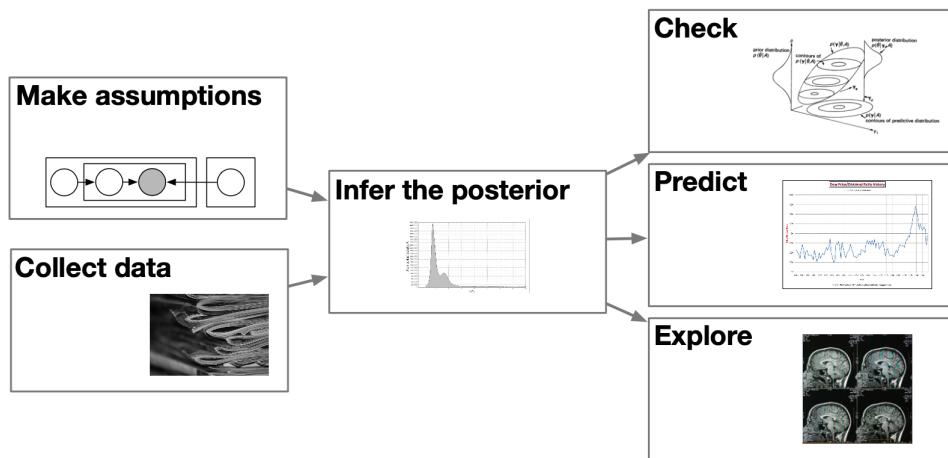
Trong thực tế, sự không chắc chắn (uncertainty) trong việc học và suy diễn (phán đoán) có thể diễn ra thường xuyên. Lý do là vì sự xuất hiện của nhiễu/lỗi trong tập dữ liệu, vì khả năng của lớp mô hình \mathcal{H} , và vì khả năng của một giải thuật học. Tuy nhiên, các phương pháp đã trình bày ở các mục trước thường bỏ qua yếu tố không chắc chắn này. Việc bỏ qua như thế có thể dẫn đến kết quả không tốt.

Để làm việc với sự không chắc chắn, chúng ta có thể mượn các công cụ trong lĩnh vực Xác suất (Probability theory) và Thống kê (Statistics). Khi đó, chúng ta cần dùng các khái niệm và quy luật xác suất để giải quyết bài toán học/suy diễn. Quy trình mô hình hoá cơ bản, được minh họa trong Hình 38, bao gồm các bước sau:

1. *Tạo giả thuyết (Make assumptions)*: Chúng ta cần đưa ra một số giả thuyết về quá trình các mẫu dữ liệu được sinh ra. Việc này cũng tương tự như việc chọn kiểu mô hình mà ta đã biết trước đây.
2. *Thu thập dữ liệu (Collect data)*: Thiết kế hệ thống và thu thập dữ liệu thực tế, để phục vụ việc huấn luyện.
3. *Huấn luyện (Infer the posterior)*: Sử dụng một số phương pháp để huấn luyện, tìm ra mô hình. Ví dụ bằng cách cực đại hóa phân bố hậu nghiệm của một (vài) biến nào đó.
4. *Kiểm tra (Check)*: Sau khi đã học, chúng ta cần kiểm tra chất lượng của mô hình thu được. Nếu nó có chất lượng tốt thì có thể dùng cho các bước tiếp theo.
5. *Dự đoán (Predict)*: Dùng mô hình đã học để thực hiện dự đoán.
6. *Khám phá (Explore)*: Dùng mô hình đã học để khám phá và tìm ra tri thức mới.

3.4.2 Nhắc lại một số khái niệm trong xác suất

Trong thực tế đôi khi chúng ta thấy một sự kiện nào đó mà có đầu ra ngẫu nhiên. Chẳng hạn sau khi quay xổ số, chúng ta thu được một con số nào đó xuất hiện một cách



Hình 38: Các bước chính khi làm việc với một mô hình xác suất [Blei, 2012].

ngẫu nhiên; khi quay 5 lần thì sẽ thu được 5 con số ngẫu nhiên. Trong xác suất, một sự kiện như thế thường được gọi là *sự kiện ngẫu nhiên*.

Để biểu diễn một sự kiện ngẫu nhiên, chúng ta có thể dùng một biến và nó thường được gọi là *biến ngẫu nhiên* (random variable). Mỗi biến ngẫu nhiên thường được gắn với một xác suất xảy ra. Ví dụ $\Pr(x = 3)$ mô tả xác suất thu được số 3, trong đó x mô tả một lần quay xổ số.

Xét hai sự kiện ngẫu nhiên A và B .

- Sự xuất hiện của A có thể phụ thuộc vào B hoặc ngược lại. Khả năng cùng xảy ra của cả A và B được gọi là *xác suất kết hợp* (joint probability) giữa chúng, và được ký hiệu là $\Pr(A, B)$.
- Sự xuất hiện của A có thể phụ thuộc vào B . Khả năng xảy ra của A mà phụ thuộc vào B được gọi là *xác suất có điều kiện* (conditional probability), và được ký hiệu là $\Pr(A|B)$.
- Sự xuất hiện của A có thể độc lập với B . Khi đó chúng ta nói rằng hai sự kiện này *độc lập*. Khi đó $\Pr(A|B) = \Pr(A)$.
- *Độc lập có điều kiện* (conditionally independent): Hai sự kiện A và C có thể phụ thuộc vào nhau, nhưng khi có sự xuất hiện của B thì chúng độc lập với nhau, tức là $\Pr(A|B, C) = \Pr(A|B)$. Khi đó chúng được gọi là *độc lập có điều kiện*.

Tính chất 1 Xét hai biến ngẫu nhiên $x \in \mathcal{X}$ và $y \in \mathcal{Y}$. Khi đó:

$$\Pr(x, y) = \Pr(x|y) \Pr(y) \quad (\text{Luật tích})$$

$$\Pr(x) = \sum_{y \in \mathcal{Y}} \Pr(x, y) \quad (\text{Luật tổng})$$

Đây là hai tính chất rất căn bản của các biến ngẫu nhiên. Chúng được dùng thường xuyên khi làm việc với các mô hình xác suất. Chú ý rằng nếu \mathcal{Y} là một miền liên tục thì đại lượng tổng ở trên sẽ được thay bằng tích phân $\int_{\mathcal{Y}} p(x, y) dy$, trong đó $p(x, y)$ là hàm mật độ (density function).

Dưới đây là một tính chất rất quan trọng từ Định lý Bayes.

Tính chất 2 (Luật Bayes) *Đối với hai biến ngẫu nhiên D và $\boldsymbol{\theta}$ bất kỳ:*

$$\Pr(\boldsymbol{\theta}|D) = \frac{\Pr(D|\boldsymbol{\theta}) \Pr(\boldsymbol{\theta})}{\Pr(D)} \quad (61)$$

Để thấy được ý nghĩa của tính chất này, chúng ta hãy lấy ví dụ trong trường hợp D là (mẫu/tập) dữ liệu và $\boldsymbol{\theta}$ mô tả một mô hình nào đó. Khi đó:

- $\Pr(D)$: xác suất để ta quan sát được dữ liệu D , và thường được gọi là **Xác suất tiên nghiệm (prior probability)**.
- $\Pr(\boldsymbol{\theta})$: xác suất tiên nghiệm của $\boldsymbol{\theta}$. Nó mô tả tri thức trước đó về mô hình $\boldsymbol{\theta}$, mà độc lập với dữ liệu. Ví dụ một số hiểu biết hoặc kinh nghiệm trong quá khứ.
- $\Pr(D|\boldsymbol{\theta})$: xác suất để ta quan sát được dữ liệu D nếu biết trước mô hình $\boldsymbol{\theta}$. Xác suất này thường được gọi là **Cơ hội xảy ra (likelihood)**.
- $\Pr(\boldsymbol{\theta}|D)$: khả năng mô hình $\boldsymbol{\theta}$ là đúng nếu ta quan sát được dữ liệu D . Nó thường được gọi là **Xác suất hậu nghiệm (posterior probability)**.

Trong kỷ nguyên của dữ liệu lớn, việc thu thập dữ liệu thường khá dễ dàng. Do đó việc dùng dữ liệu để giúp chúng ta suy luận, học, phán đoán là một con đường rất hữu ích để giải quyết nhiều bài toán thực tế. Do đó đại lượng $\Pr(\boldsymbol{\theta}|D)$ thường được quan tâm, bởi mô hình $\boldsymbol{\theta}$ thường ẩn hoặc không biết trước.

3.4.3 Mô hình xác suất

Ở mục 3.4.1, chúng ta đã biết rằng để giải quyết các bài toán học, một kiểu mô hình thường được sử dụng và mỗi mô hình thường ở dạng $f(\mathbf{x}; \boldsymbol{\theta})$, trong đó $\boldsymbol{\theta}$ là tham số để

mô tả mô hình đó. Việc huấn luyện về cơ bản sẽ đi tìm một θ^* cụ thể dựa trên tập học D đã cho.

Nếu ta coi x và θ là các biến ngẫu nhiên thì về cơ bản chúng ta sẽ làm việc với một mô hình xác suất. Lúc này hàm $f(x; \theta)$ đôi khi được viết là $f(x|\theta)$ và có thể ở dạng không tưởng minh. Theo luật Bayes, việc tìm $f(x|\theta)$ có thể thông qua việc tìm $f(\theta|x)$ hoặc $f(x, \theta)$. Do đó cách học và suy diễn lúc này sẽ rất linh động.

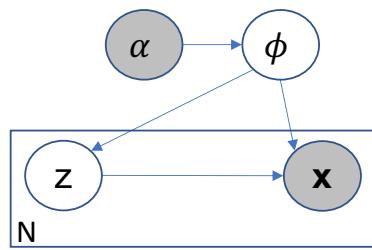
Mỗi mô hình xác suất (probabilistic model) sẽ chứa một số giả thuyết (assumption) của chúng ta. Điều này cũng tương tự như các mô hình đã được trình bày trước đây. Ví dụ, mỗi lựa chọn dạng mô hình \mathcal{H} đã ngầm giả thuyết rằng tồn tại một hàm $f \in \mathcal{H}$ mà có thể xấp xỉ tốt hàm không biết. *Đối với mỗi mô hình xác suất, các giả thuyết của chúng ta thường mô tả quá trình sinh ra các mẫu dữ liệu trong thực tế.* Ví dụ, để nói ra một câu có ý nghĩa, bộ não chúng ta có thể trải qua hai bước chính: (a) chọn ý (chủ đề) của câu cần nói; (b) sinh ra các từ lần lượt để mô tả được ý đó.

Trong một mô hình xác suất, một số loại biến ngẫu nhiên có thể xuất hiện:

- *Biến quan sát được (observed variable):* là biến mà ta có thể thu thập được các quan sát cụ thể về nó. Ví dụ để mô tả các từ, ta có thể dùng biến x và là biến quan sát được.
- *Biến ẩn (latent variable):* là biến mà ta không thể thu thập trực tiếp được các quan sát cụ thể về nó. Ví dụ ý của một câu không thể thu thập trực tiếp được.
- *Biến địa phương (local variable):* là biến dành trong việc mô tả một mẫu dữ liệu cụ thể.
- *Biến toàn cục (global variable):* là biến có tác động tới quá trình sinh nhiều mẫu dữ liệu khác nhau.

Đặc biệt, các biến trong một mô hình thường có liên hệ về mặt xác suất với nhau. Ví dụ trong Hình 39, biến x phụ thuộc vào biến z và ϕ . Việc này sẽ giúp chúng ta mô tả các mối quan hệ của các đối tượng khác nhau trong thực tế. Chẳng hạn, việc tạo ra một câu nói sẽ phụ thuộc vào ý muốn nói. Ngoài ra, mỗi biến trong một mô hình thường tuân theo một phân bố xác suất (probability distribution) cụ thể nào đó.

Rất nhiều mô hình xác suất đã và đang được dùng trong thực tế. Ví dụ mô hình Latent Dirichlet allocation (LDA) [Blei et al., 2003], mô hình Markov ẩn (HMM) [Baum et al.,



Hình 39: Ví dụ về một mô hình xác suất, có các biến (z, x, ϕ) và tham số α .

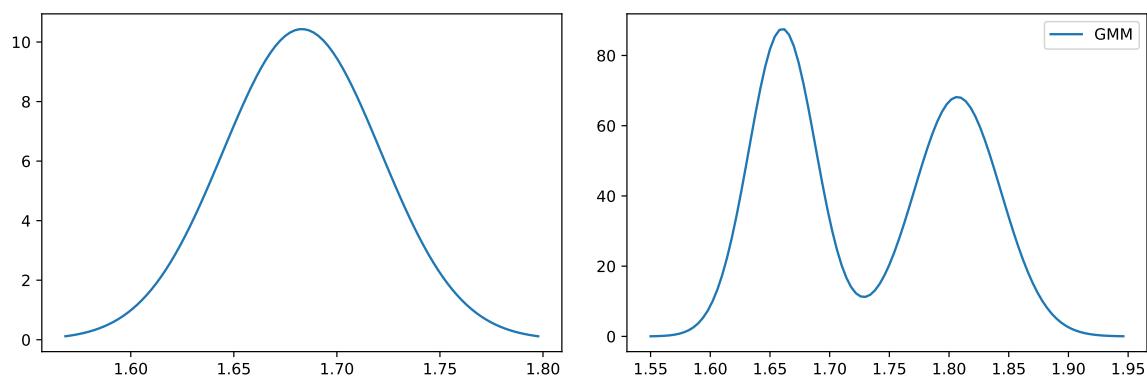
1970], Conditional Random Field (CRF) [Lafferty et al., 2001], mô hình khuếch tán (Diffusion models) [Ho et al., 2020, Rombach et al., 2022, Croitoru et al., 2023]. Trong đó lớp mô hình khuếch tán đã và đang tạo đột phá lớn trong ML nói riêng, và AI nói chung.

Mô hình đồ thị xác suất (Probabilistic graphical model - PGM): Để giúp ta dễ hình dung về một mô hình xác suất và mối quan hệ của các biến bên trong nó, ta có thể dùng một đồ thị để mô tả, trong đó:

- Mỗi đỉnh mô tả một biến.
- Mỗi cạnh mô tả sự phụ thuộc giữa hai biến. Ví dụ ở Hình 39, biến x phụ thuộc vào z .
- Đỉnh màu xám mô tả một biến quan sát được.
- Đỉnh giống mô tả một biến ẩn.
- Một khung chữ nhật để mô tả một nhóm biến trong một mẫu dữ liệu nào đó.

Như vậy các mô hình đồ thị xác suất là kết quả của việc kết hợp giữa lý thuyết đồ thị và xác suất. Đây là một lớp rất lớn các mô hình khác nhau trong thực tế. Tuỳ vào đặc trưng khác nhau mà người ta có thể chia thành nhóm mô hình nhỏ hơn. Ví dụ mô hình biến ẩn (latent variable model) chứa ít nhất một biến ẩn, mô hình Bayes nếu nó dùng phân bố tiên nghiệm về các tham số, mô hình PGM vô hướng nếu các cạnh không có hướng. Hình 39 chứa ví dụ về một mô hình biến ẩn.

Ví dụ 3 (Mô hình Gauss đơn biến) Gọi x là một biến ngẫu nhiên để mô tả chiều cao của một người. Giả sử x tuân theo phân bố chuẩn với kỳ vọng μ và phương sai σ^2 . Phân bố này có hàm mật độ là $\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$.



Hình 40: Hàm mật độ của phân bố chuẩn (bên trái) và mô hình hỗn hợp Gauss (bên phải).

Đây là một kiểu mô hình rất đơn giản và có giả thuyết đơn giản. Ở đây $\mathcal{N}(x; \mu, \sigma)$ là một lớp các phân bố, mà mỗi cặp (μ, σ) mô tả một phân bố cụ thể. Hình 40 chứa ví dụ về một phân bố chuẩn. Việc huấn luyện mô hình này thực ra là tìm tham số $\boldsymbol{\theta} = (\mu, \sigma)$, dựa trên một tập học cho trước.

Ví dụ 4 (Mô hình hỗn hợp Gauss - GMM) Xét một mô hình mà trong đó mỗi mẫu dữ liệu $\mathbf{x} \in \mathbb{R}^n$ được sinh ra bởi một trong số K phân bố chuẩn khác nhau, mà phân bố thứ i có kỳ vọng $\boldsymbol{\mu}_i$ và ma trận hiệp phương sai (covariance matrix) $\boldsymbol{\Sigma}_i$.

Đây chính là mô hình hỗn hợp Gauss (Gaussian mixture model - GMM). Mô hình này giả thuyết rằng mỗi mẫu dữ liệu được sinh ra bởi một **quá trình sinh** (generative process) như sau:

1. Chọn phân bố thứ $z \sim Cat(\phi)$
2. Sinh ra mẫu $\mathbf{x} \sim Normal(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$

trong đó $Cat(\phi)$ là phân bố Phân loại (categorical distribution) với tham số $\phi = (\phi_1, \dots, \phi_K)$, $Normal(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ là phân bố chuẩn với hai tham số $(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$, và ký hiệu \sim mô tả sự "tuân theo". Mỗi ϕ_k mô tả khả năng xảy ra của giá trị $k \in \{1, 2, \dots, K\}$, tức là $\phi_k = \Pr(z = k | \phi)$, và cần thoả mãn $1 = \sum_{k=1}^K \phi_k$. Các ràng buộc này về ϕ nói rằng mỗi mẫu dữ liệu được sinh bởi phân bố Cat sẽ chỉ nhận một trong số K giá trị khác nhau. Hình 40 chứa ví dụ về GMM với hai phân bố chuẩn.

Hàm mật độ của GMM này là

$$f(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi) = \sum_{k=1}^K \phi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (62)$$

trong đó $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_k)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$ là hàm mật độ của phân bố chuẩn thứ k .

Chúng ta thấy rằng GMM chứa K phân bố chuẩn khác nhau và có thể có tâm đặt ở các vùng khác nhau trong không gian của \mathbf{x} . Hình 40 minh họa về hai phân bố chuẩn trong GMM. Như vậy, K phân bố này mô tả đặc trưng của dữ liệu trong các vùng khác nhau. Đồng thời chúng có thể giúp ta phân hoạch không gian dữ liệu thành nhiều vùng, mà mỗi phân bố mô tả một vùng. Do đó GMM có thể được dùng để phân cụm dữ liệu, trong đó $\boldsymbol{\mu}_k$ là tâm cụm thứ k .

3.4.4 Học và suy diễn

Có hai bài toán cơ bản khi chúng ta sử dụng một mô hình xác suất: *học và suy diễn*. Hai bài toán này tương tự như trong các mô hình trước đây. Nhưng bước suy diễn trong mô hình xác suất thường phức tạp hơn nhiều.

Suy diễn (inference) chính là bước vận dụng một mô hình đã có (đã huấn luyện) để xử lý (phán đoán) một mẫu dữ liệu \mathbf{x} cụ thể. Đối với mô hình xác suất, chúng ta cần sử dụng các quy tắc của xác suất để thực hiện suy diễn. Nghĩa là, ta sẽ cần ước lượng xác suất hoặc phân bố của một (vài) biến địa phương nào đó. Ví dụ:

- Xét mô hình trong Hình 39, ta cần tính $\Pr(z|\mathbf{x}, \alpha)$ hoặc $\Pr(z, \mathbf{x}|\alpha)$ hoặc tìm phân bố đầy đủ $p(z, \mathbf{x}|\alpha)$.
- Xét GMM, ta muốn biết \mathbf{x} nên được gán vào cụm nào. Khi đó ta có thể tìm cụm thứ z mà có $\Pr(z|\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$ lớn nhất.

Học (training hoặc estimation) là giai đoạn mà chúng ta cần tìm một mô hình phù hợp nhất từ một lớp mô hình đã chọn, dựa trên tập học đã có. Nói cách khác, chúng ta cần tìm bộ tham số mô tả mô hình đó. Ví dụ, cho trước tập học $\mathbf{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$,

- Xét mô hình trong Hình 39, ta cần tìm α hoặc tìm phân bố đầy đủ $p(z, \mathbf{x}, \phi|\alpha)$.
- Xét GMM, ta tìm bộ tham số $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$.

3.4.5 Một số phương pháp suy diễn

Tiếp theo chúng ta hãy tìm hiểu một số phương pháp phổ biến để giải quyết bài toán học và suy diễn.

Giả sử chúng ta đã chọn một kiểu/lớp mô hình \mathcal{H} , mà mỗi $h \in \mathcal{H}$ thường được gọi là một giả thuyết (mô hình, bộ tham số, một số biến, ...) cần quan tâm, và \mathbf{D} là (tập/mẫu) dữ liệu đã có.

Định nghĩa 7 (Maximum Likelihood Estimation - MLE) *Chúng ta sẽ tìm một giả thuyết h^* mà có khả năng $\Pr(\mathbf{D}|h^*)$ lớn nhất. Tức là*

$$h^* = \arg \max_{h \in \mathcal{H}} \Pr(\mathbf{D}|h)$$

Phương pháp này được gọi là *Cực đại hoá khả năng* hoặc *Khả năng lớn nhất*. Ta có thể hiểu rằng, MLE sẽ tìm một mô hình mà có khả năng cao là nó sinh ra dữ liệu \mathbf{D} . Ở đây, chúng ta chỉ cần quan tâm đến mô hình cụ thể đó mà không cần quan tâm đến phân bố của nó.

Ngược lại, **suy diễn Bayes** (Bayes inference) thường quan tâm đến việc biến đổi tri thức tiên nghiệm $p(h)$ của chúng ta, thông qua dữ liệu \mathbf{D} , vào tri thức hậu nghiệm $p(h|\mathbf{D})$. Để làm được điều đó, chúng ta có thể dùng luật Bayes. Đây là một phương pháp có thể dùng tri thức bên ngoài vào quá trình suy diễn. Nó sẽ rất hữu ích trong thực tế khi chúng ta có tri thức tốt về h .

Để dùng được tri thức đã có về h , chúng ta cần mã hoá nó vào phân bố tiên nghiệm $p(h)$. Ví dụ, chúng ta có thể biết trước rằng "spam" emails thường có tần suất xuất hiện cao hơn 9 lần các emails thường trong bài toán phân loại thư rác. Nếu h là nhãn lớp trong bài toán này thì ta có thể mã hoá $\Pr(h = \text{"spam"}) = 9/10$ và $\Pr(h \neq \text{"spam"}) = 1/10$. Một cách mã hoá tốt có thể giúp chúng ta tận dụng tốt tri thức bên ngoài.

Định nghĩa 8 (Maximum a Posteriori Estimation - MAP) *Chúng ta sẽ tìm một giả thuyết h^* mà có xác suất hậu nghiệm $\Pr(h^*|\mathbf{D})$ lớn nhất. Tức là*

$$h^* = \arg \max_{h \in \mathcal{H}} \Pr(h|\mathbf{D})$$

Phương pháp này thường được gọi là *Cực đại hóa hậu nghiệm*. Ở đó, chúng ta chỉ cần tìm một mô hình có xác suất hậu nghiệm lớn nhất. Gọi $p(h|\mathbf{D})$ là phân bố hậu nghiệm của h . Khi đó h^* được tìm theo MAP chính là điểm cực đại của phân bố này. Như vậy, MAP chỉ cần tìm ra một điểm/mẫu của phân bố, chứ không phải tìm phân bố đầy đủ.

Theo luật Bayes thì $\Pr(h|\mathbf{D}) = \Pr(\mathbf{D}|h) \Pr(h) / \Pr(\mathbf{D})$. Khi ta cần tìm h thì đại lượng $\Pr(\mathbf{D})$ là hằng số. Do đó

$$h^* = \arg \max_{h \in \mathcal{H}} \Pr(h|\mathbf{D}) = \arg \max_{h \in \mathcal{H}} \Pr(\mathbf{D}|h) \Pr(h) / \Pr(\mathbf{D}) = \arg \max_{h \in \mathcal{H}} \Pr(\mathbf{D}|h) \Pr(h) \quad (63)$$

Như vậy MLE là một trường hợp đặc biệt của MAP nếu h tuân theo phân bố đều, bởi vì $\Pr(h)$ giống nhau cho mọi h . Vì MAP sử dụng tri thức tiên nghiệm vào quá trình suy diễn, nên nó là một ví dụ cụ thể của suy diễn Bayes. Hiệu quả của MAP phụ thuộc nhiều vào chất lượng của tri thức tiên nghiệm.

Đôi khi chúng ta muốn tìm được phân bố hậu nghiệm $p(h|\mathbf{D})$ ở dạng đầy đủ. Việc biết dạng đầy đủ của phân bố này có thể mang lại nhiều lợi ích về sau, chẳng hạn lấy mẫu mới (sampling) từ phân bố đó. Tuy nhiên, việc tìm ra $p(h|\mathbf{D})$ không dễ đối với một số mô hình phức tạp.

Chú ý: MLE và MAP là các phương pháp khác nhau để tìm ra một giả thuyết h . Chúng là các hướng rất tổng quát và có thể được dùng để giải quyết bài toán học hoặc suy diễn cho các mô hình xác suất.

Ví dụ 5 (Học mô hình Gauss) Xét mô hình phân bố chuẩn mà trong đó có biến x mô tả chiều cao của một người. Nhiệm vụ là tìm phân bố đó từ tập học $\mathbf{D} = \{1.6, 1.7, 1.65, 1.63, 1.75, 1.71, 1.68, 1.72, 1.77, 1.62\}$.

Giả sử phân bố chuẩn có kỳ vọng μ và phương sai σ^2 . Hàm mật độ của nó là $\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$. Việc tìm mô hình này về bản chất là tìm các tham số (μ, σ) . Chúng ta sẽ sử dụng MLE để học. Nghĩa là ta cần tìm

$$(\mu_*, \sigma_*) = \arg \max_{\mu, \sigma} \Pr(\mathbf{D}|\mu, \sigma) = \arg \max_{\mu, \sigma} \Pr(x_1, \dots, x_{10}|\mu, \sigma)$$

trong đó x_1, \dots, x_{10} là các mẫu dữ liệu trong \mathbf{D} . Ở bài toán này, chúng ta cần tính xác suất kết hợp của 10 biến. Điều này có thể rất khó.

Giả thuyết i.i.d (Independent and identically distributed): *Giả sử các mẫu dữ liệu trong \mathbf{D} được sinh ra một cách độc lập và đồng nhất từ một phân bố.*

Với giả thuyết này, ta có $\Pr(x_1, \dots, x_{10}|\mu, \sigma) = \prod_{i=1}^{10} \Pr(x_i|\mu, \sigma)$. Nghĩa là ta có thể

viết lại bài toán trên như sau:

$$(\mu_*, \sigma_*) = \arg \max_{\mu, \sigma} \Pr_{\mu, \sigma}(x_1, \dots, x_{10} | \mu, \sigma) \quad (64)$$

$$= \arg \max_{\mu, \sigma} \prod_{i=1}^{10} \Pr(x_i | \mu, \sigma) \quad (\text{Giả sử i.i.d.}) \quad (65)$$

$$= \arg \max_{\mu, \sigma} \prod_{i=1}^{10} \mathcal{N}(x_i; \mu, \sigma) \quad (66)$$

$$= \arg \max_{\mu, \sigma} \log \prod_{i=1}^{10} \mathcal{N}(x_i; \mu, \sigma) \quad (\text{Mẹo logarit}) \quad (67)$$

$$= \arg \max_{\mu, \sigma} \sum_{i=1}^{10} \log(\mathcal{N}(x_i; \mu, \sigma)) \quad (68)$$

$$= \arg \max_{\mu, \sigma} \sum_{i=1}^{10} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{1}{2\sigma^2}(x_i - \mu)^2 \right) \right) \quad (69)$$

$$= \arg \max_{\mu, \sigma} \sum_{i=1}^{10} \left(-\log \sqrt{2\pi\sigma^2} - \frac{1}{2\sigma^2}(x_i - \mu)^2 \right) \quad (70)$$

Ta đã sử dụng tính chất đơn điệu tăng của hàm logarit để chuyển bài toán (66) về bài toán (67), lý do là điểm cực đại của hàm $f(x)$ nào đó cũng chính là điểm cực đại của $\log[f(x)]$.

Ở đây chúng ta quan tâm đến các tham số (μ, σ) . Do hàm mục tiêu trong (70) là hàm lồi theo các tham số này nên ta có thể lấy đạo hàm và tìm điểm làm cho đạo hàm này bằng 0. Sau khi giải ta sẽ thu được nghiệm là

$$\mu_* = \frac{1}{10} \sum_{i=1}^{10} x_i = 1.683; \quad \sigma_*^2 = \frac{1}{10} \sum_{i=1}^{10} (x_i - \mu_*)^2 = 0.0015$$

Như vậy chúng ta đã thấy MLE được sử dụng để huấn luyện mô hình Gauss trong Ví dụ 5. Tiếp theo chúng ta sẽ xét hai mô hình dành cho bài toán phân loại.

3.4.6 Naive Bayes cho phân loại

Xét một bài toán phân loại, với tập học $\mathbf{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_M, y_M)\}$. Ta cần học một mô hình mà có thể gán nhãn lớp y cho mỗi mẫu dữ liệu \mathbf{x} , dựa vào tập học \mathbf{D} . Mỗi \mathbf{x} là một vectơ trong không gian n chiều, và tổng số nhãn lớp là C .

Ví dụ 6 (Gaussian Naive Bayes - GNB) Giả sử tồn tại C phân bố Gauss khác nhau, mà mỗi phân bố thứ c sinh ra các mẫu dữ liệu có nhãn c . Cụ thể hơn, mỗi mẫu dữ liệu được sinh ra bởi quá trình sinh như sau:

1. Chọn nhãn thứ $c \sim Cat(\phi)$

2. Sinh ra mẫu $\mathbf{x} \sim Normal(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$

trong đó $Cat(\phi)$ là phân bố Phân loại (categorical distribution) với tham số $\phi = (\phi_1, \dots, \phi_K)$, $Normal(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ là phân bố chuẩn với hai tham số $(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. Tham số ϕ cần thoả mãn $\phi_k = \Pr(c = k | \phi)$ với mỗi chỉ số k và $1 = \sum_{k=1}^K \phi_k$.

Ta thấy rằng mô hình GNB có giả thuyết rất đơn giản. Các phân bố trong mô hình này tách biệt nhau, và việc sinh ra mẫu dữ liệu ở một lớp không bị ảnh hưởng bởi các lớp khác. Do đó việc huấn luyện có thể dễ.

Giả sử ta có tập học \mathbf{D} và các mẫu dữ liệu thoả mãn giả thuyết i.i.d. Chúng ta sẽ dùng phương pháp MAP để thực hiện huấn luyện và suy diễn. Đối với mô hình này, các tham số cần tìm là $\boldsymbol{\theta} = \{\phi_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \dots, \phi_C, \boldsymbol{\mu}_C, \boldsymbol{\Sigma}_C\}$. Mỗi mẫu dữ liệu $\mathbf{x}_i \in \mathbf{D}$ có một nhãn lớp y_i . Gọi c là biến mô tả nhãn lớp, và là biến ẩn cần tìm khi ta cần phân loại cho mỗi \mathbf{x} . Xác suất hậu nghiệm của nhãn là $\Pr(c = y_i | \mathbf{x}_i, \boldsymbol{\theta})$ và ta sẽ dùng ký hiệu $\Pr(y_i | \mathbf{x}_i, \boldsymbol{\theta})$ cho đơn giản hơn.

Để huấn luyện, ta hãy quan tâm đến xác suất kết hợp $\Pr(y_1, \dots, y_M | \mathbf{x}_1, \dots, \mathbf{x}_M, \boldsymbol{\theta})$. Theo MAP, việc huấn luyện sẽ đưa về bài toán tìm

$$\boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} \Pr(y_1, \dots, y_M | \mathbf{x}_1, \dots, \mathbf{x}_M, \boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^M \Pr(y_i | \mathbf{x}_i, \boldsymbol{\theta}) \quad (\text{Do i.i.d.}) \quad (71)$$

$$= \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^M \Pr(\mathbf{x}_i | y_i, \boldsymbol{\theta}) \Pr(y_i | \boldsymbol{\theta}) \quad (\text{Luật Bayes}) \quad (72)$$

$$= \arg \max_{\boldsymbol{\theta}} \sum_{i=1}^M [\log \Pr(\mathbf{x}_i | y_i, \boldsymbol{\theta}) + \log \Pr(y_i | \boldsymbol{\theta})] \quad (\text{Mẹo logarit}) \quad (73)$$

$$= \arg \max_{\boldsymbol{\theta}} \sum_{k=1}^C \sum_{\mathbf{x} \in \mathbf{D}_k} [\log \Pr(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \log \Pr(c = k | \phi_k)] \quad (74)$$

$$= \arg \max_{\boldsymbol{\theta}} \sum_{k=1}^C \sum_{\mathbf{x} \in \mathbf{D}_k} [\log \Pr(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \log \phi_k] \quad (75)$$

$$= \arg \max_{\boldsymbol{\theta}} \sum_{k=1}^C \sum_{\mathbf{x} \in \mathbf{D}_k} \log \Pr(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \sum_{k=1}^C |\mathbf{D}_k| \log \phi_k \quad (76)$$

trong đó \mathbf{D}_k là tập chứa các mẫu dữ liệu có nhãn k của \mathbf{D} , và $|\mathbf{D}_k|$ là số lượng mẫu có nhãn k .

Trước tiên ta hãy tìm ϕ_k trong bài toán tối ưu này. Chú ý rằng hàm mục tiêu trong (76) có tính tách biệt giữa ϕ_k và $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Hơn nữa ϕ cần thoả mãn $\sum_{j=1}^C \phi_j = 1, \phi_k \geq 0, \forall k$. Do đó ta có thể tìm ϕ_k bằng cách giải bài toán

$$\max \sum_{k=1}^C |\mathbf{D}_k| \log \phi_k \quad \text{sao cho } \sum_{j=1}^C \phi_j = 1, \phi_k \geq 0, \forall k \quad (77)$$

Bằng phương pháp nhân tử Lagrange, ta có thể dễ dàng tìm được nghiệm tối ưu là

$$\phi_k^* = \frac{|\mathbf{D}_k|}{M} \quad (78)$$

Tiếp theo ta có thể tìm các tham số $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ khá dễ dàng trong bài toán (76). Chú ý rằng ta có thể tìm từng cặp

$$(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^*) = \arg \max_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k} \sum_{\mathbf{x} \in \mathbf{D}_k} \log \Pr(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \arg \max_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k} \sum_{\mathbf{x} \in \mathbf{D}_k} \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (79)$$

$$= \arg \max_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k} \sum_{\mathbf{x} \in \mathbf{D}_k} \log \left[\frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_k)}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right) \right] \quad (80)$$

$$= \arg \max_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k} \sum_{\mathbf{x} \in \mathbf{D}_k} \left[-\log \sqrt{\det(2\pi\boldsymbol{\Sigma}_k)} - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right] \quad (81)$$

Bằng cách tính đạo hàm của hàm mục tiêu và tìm điểm làm cho đạo hàm bằng 0, ta sẽ tìm được nghiệm như sau:

$$\boldsymbol{\mu}_k^* = \frac{1}{|\mathbf{D}_k|} \sum_{\mathbf{x} \in \mathbf{D}_k} \mathbf{x}; \quad \boldsymbol{\Sigma}_k^* = \frac{1}{|\mathbf{D}_k|} \sum_{\mathbf{x} \in \mathbf{D}_k} (\mathbf{x} - \boldsymbol{\mu}_k^*)(\mathbf{x} - \boldsymbol{\mu}_k^*)^T \quad (82)$$

Như vậy quá trình huấn luyện GNB sẽ tính các tham số $(\phi_k^*, \boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^*)$ cho mỗi lớp k theo công thức (78) và (82). Sau khi huấn, ta có thể dùng chúng để thực hiện phán đoán nhãn cho mẫu dữ liệu mới \mathbf{z} một cách dễ dàng. Cụ thể, ta sẽ phán đoán nhãn c_z cho mẫu

đó, với

$$c_z = \arg \max_{c \in \{1, \dots, C\}} \Pr(c | \mathbf{z}, \phi_c^*, \boldsymbol{\mu}_c^*, \boldsymbol{\Sigma}_c^*) \quad (83)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \Pr(\mathbf{z} | \boldsymbol{\mu}_c^*, \boldsymbol{\Sigma}_c^*) \Pr(c | \phi_c^*) \quad (84)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \log \Pr(\mathbf{z} | \boldsymbol{\mu}_c^*, \boldsymbol{\Sigma}_c^*) + \log \Pr(c | \phi_c^*) \quad (85)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \left[-\log \sqrt{\det(2\pi \boldsymbol{\Sigma}_c^*)} - \frac{1}{2} (\mathbf{z} - \boldsymbol{\mu}_c^*)^T (\boldsymbol{\Sigma}_c^*)^{-1} (\mathbf{z} - \boldsymbol{\mu}_c^*) + \log \phi_c^* \right] \quad (86)$$

Ở đây, chúng ta đã dùng phương pháp MAP để thực hiện phán đoán (suy diễn). Nếu dùng MLE thì chúng ta có thể bỏ qua việc tìm phân bố tiên nghiệm $p(c)$ của nhãn lớp.

Chú ý: Trong ví dụ trên chúng ta đã dùng phân bố chuẩn đầy đủ cho mỗi lớp. Để phán đoán, ta cần tính ma trận nghịch đảo của mỗi $\boldsymbol{\Sigma}_c^*$. Điều này thường tốn kém với bài toán có số chiều n cao. Đổi lại, ta có thể tìm ra được mối tương quan giữa các thuộc tính. Nếu ta giảm tính toán thì có thể dùng giả thuyết thêm rằng *các thuộc tính độc lập*, khi đó các ma trận hiệp phương sai $\boldsymbol{\Sigma}_c$ sẽ ở dạng đường chéo. Khi đó việc tính toán và lưu trữ có thể dễ dàng và đơn giản hơn so với trường hợp đầy đủ. Tuy nhiên việc dùng thêm giả thuyết độc lập này sẽ làm cho mô hình có khả năng yếu hơn.

Ví dụ 7 (Multinomial Naive Bayes - MNB) Giả sử tồn tại C phân bố đa thức khác nhau, mà mỗi phân bố thứ c sinh ra các mẫu dữ liệu có nhãn c . Cụ thể hơn, mỗi mẫu dữ liệu $\mathbf{x} = (x_1, \dots, x_n)^T$ được sinh ra bởi quá trình sinh như sau:

1. Chọn nhãn thứ $c \sim \text{Cat}(\phi)$
2. Sinh ra mẫu $\mathbf{x} \sim \text{Multinomial}(\boldsymbol{\theta}_c)$

trong đó $\text{Cat}(\phi)$ là phân bố Phân loại (categorical distribution) với tham số $\phi = (\phi_1, \dots, \phi_K)$, và $\text{Multinomial}(\boldsymbol{\theta})$ là phân bố Đa thức với tham số $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$. Tham số ϕ cần thoả mãn $\phi_k = \Pr(c = k | \phi)$ với mỗi chỉ số k và $1 = \sum_{k=1}^K \phi_k$. Tương tự vậy, θ_k mô tả khả năng xảy ra của $x_k \in \mathbb{N}$ với mỗi chỉ số k , tức là $\theta_k = \Pr(x_k \neq 0 | \boldsymbol{\theta})$, và thoả mãn $1 = \sum_{k=1}^K \theta_k$.

Có thể thấy mô hình MNB chỉ khác GNB ở việc dùng phân bố khác để sinh ra một mẫu dữ liệu, còn quá trình sinh không thay đổi. Tương tự như thế, nếu ta thay phân bố này (Multinomial) bằng một phân bố khác thì ta sẽ thu được một mô hình khác. Do đó ta có thể tạo ra rất nhiều mô hình khác nhau một cách đơn giản.

Đối với một phân bố đa thức, hàm xác suất có dạng:

$$f(x_1, \dots, x_n; \theta_1, \dots, \theta_n) = \frac{\Gamma(1 + \sum_{j=1}^n x_j)}{\sum_{j=1}^n \Gamma(1 + x_j)} \prod_{k=1}^n \theta_k^{x_k} \quad (87)$$

trong đó Γ là hàm gamma. Ở đây chúng ta không cần quan tâm hình dạng cụ thể của hàm gamma này vì việc huấn luyện và phán đoán không cần dùng giá trị cụ thể của hàm đó.

Để huấn luyện MNB từ một tập học D cho trước, ta có thể làm tương tự như đối với GNB. Theo (78) ta có nghiệm tối ưu

$$\phi_c^* = \frac{|\mathbf{D}_c|}{M}, \quad 1 \leq c \leq C, \quad (88)$$

trong đó \mathbf{D}_c là tập chứa tất cả các mẫu dữ liệu có nhãn c trong tập học. Có thể tìm $\boldsymbol{\theta}_c = (\theta_{c1}, \dots, \theta_{cn})$ cho mỗi lớp c một cách độc lập. Khi đó ta tìm

$$\boldsymbol{\theta}_c^* = \arg \max_{\boldsymbol{\theta}_c} \sum_{\mathbf{x} \in \mathbf{D}_c} \log \Pr(\mathbf{x} | \boldsymbol{\theta}_c) = \arg \max_{\boldsymbol{\theta}_c} \sum_{\mathbf{x} \in \mathbf{D}_c} \log f(\mathbf{x}; \boldsymbol{\theta}_c) \quad (89)$$

$$= \arg \max_{\boldsymbol{\theta}_c} \sum_{\mathbf{x} \in \mathbf{D}_c} \log \left[\frac{\Gamma(1 + \sum_{j=1}^n x_j)}{\sum_{j=1}^n \Gamma(1 + x_j)} \prod_{k=1}^n \theta_{ck}^{x_k} \right] \quad (90)$$

$$= \arg \max_{\boldsymbol{\theta}_c} \sum_{\mathbf{x} \in \mathbf{D}_c} \left[\sum_{k=1}^n x_k \log \theta_{ck} \right] \quad (\text{Bỏ đi các hằng số}) \quad (91)$$

$$= \arg \max_{\boldsymbol{\theta}_c} \sum_{k=1}^n \left[\sum_{\mathbf{x} \in \mathbf{D}_c} x_k \right] \log \theta_{ck} \quad (92)$$

Chú ý rằng $\boldsymbol{\theta}_c$ cần thoả mãn $1 = \sum_{k=1}^K \theta_{ck}$ và $\boldsymbol{\theta}_c \geq \mathbf{0}$. Do đó việc tìm điểm cực đại của hàm $\sum_{k=1}^n [\sum_{\mathbf{x} \in \mathbf{D}_c} x_k] \log \theta_{ck}$ cần thoả mãn các ràng buộc này. Đây là bài toán tối ưu có ràng buộc. Bằng phương pháp nhân tử Lagrange, ta có thể dễ dàng tìm được nghiệm tối ưu là

$$\theta_{ck}^* = \frac{1}{Z_c} \sum_{\mathbf{x} \in \mathbf{D}_c} x_k, \quad 1 \leq k \leq n, \quad Z_c = \sum_{j=1}^n \sum_{\mathbf{x} \in \mathbf{D}_c} x_j \quad (93)$$

Như vậy việc huấn luyện MNB chỉ cần tính các bộ tham số $(\phi_c^*, \boldsymbol{\theta}_c^*)$ cho mỗi nhãn lớp c theo các công thức (88) và (93). Công thức (93) sẽ tính cho mỗi thuộc tính thứ k .

Sau khi học, chúng ta có thể dùng mô hình để phán đoán cho mỗi mẫu dữ liệu

$\mathbf{x} = (x_1, \dots, x_n)^T$ mới. Phương pháp MAP sẽ tìm nhãn lớp

$$c_x = \arg \max_{c \in \{1, \dots, C\}} \Pr(c | \mathbf{x}, \phi_c^*, \boldsymbol{\theta}_c^*) = \arg \max_{c \in \{1, \dots, C\}} \Pr(\mathbf{x} | \boldsymbol{\theta}_c^*) \Pr(c | \phi_c^*) \quad (94)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \log \Pr(\mathbf{x} | \boldsymbol{\theta}_c^*) + \log \Pr(c | \phi_c^*) \quad (\text{MNB.1}) \quad (95)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \left[\log \left[\frac{\Gamma(1 + \sum_{j=1}^n x_j)}{\sum_{j=1}^n \Gamma(1 + x_j)} \prod_{k=1}^n (\theta_{ck}^*)^{x_k} \right] + \log \phi_c^* \right] \quad (96)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \left[\log \prod_{k=1}^n (\theta_{ck}^*)^{x_k} + \log \phi_c^* \right] \quad (\text{Bỏ các hằng số}) \quad (97)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \left[\log \prod_{k=1}^n \Pr(x_k | \theta_{ck}^*) + \log \Pr(c | \phi_c^*) \right] \quad (\text{MNB.2}) \quad (98)$$

$$= \arg \max_{c \in \{1, \dots, C\}} \left[\sum_{k=1}^n x_k \log \theta_{ck}^* + \log \phi_c^* \right] \quad (99)$$

Từ phân tích trên về việc phán đoán, ta có thể thấy:

- Khi dùng MNB, chúng ta đã **ngầm giả thuyết rằng các thuộc tính độc lập với nhau** (có điều kiện của nhãn lớp). Điều này có thể thấy khi so sánh công thức (MNB.1) và (MNB.2).
- Theo (99), việc phán đoán cần tính $\log \theta_{ck}^*$ cho mỗi thuộc tính k và lớp c . Chú ý rằng trong thực tế một thuộc tính j nào đó không xuất hiện trong một lớp c , và khi đó công thức nghiệm (93) gợi ý rằng $\theta_{cj}^* = 0$ và làm cho biểu thức $\log \theta_{cj}^*$ không xác định. Do đó, để tránh trường hợp này, một số thư viện thường làm mịn nghiệm trong (93). Ví dụ một cách làm mịn như sau

$$\theta_{ck}^* = \frac{1}{Z_c} \sum_{\mathbf{x} \in D_c} x_k + \frac{\alpha}{Z_c}, \quad 1 \leq k \leq n, \quad Z_c = \alpha n + \sum_{j=1}^n \sum_{\mathbf{x} \in D_c} x_j \quad (100)$$

trong đó α là một hằng số dương và nhỏ.

Tóm lại, chúng ta đã tìm hiểu hai mô hình Naive Bayes dành cho bài toán phân loại. Chúng đều là những mô hình đơn giản, bởi các giả thuyết khá ngây thơ. Các giả thuyết ấy đôi khi không phù hợp với đặc trưng của dữ liệu trong thực tế. Chẳng hạn, hai thuộc tính nào đó có thể có sự phụ thuộc với nhau, hoặc các mẫu dữ liệu có thể được sinh ra bởi một phân bố phức tạp (chứ không phải phân bố chuẩn hay đa thức). Tuy nhiên, do

tính đơn giản nên Naive Bayes có lợi về tính toán và đôi khi làm việc rất tốt trong một số miền bài toán. Ví dụ, người ta thường thấy MNB thường cho kết quả phán đoán tốt khi phân loại các văn bản.

3.4.7 Bàn luận thêm

Trong mục trước, chúng ta đã thấy việc dùng MLE hoặc MAP vào giai đoạn huấn luyện và học của một số mô hình cơ bản. Đôi với các mô hình đó, chúng ta có thể tìm ra công thức nghiệm một cách tường minh cho các tham số. Nghĩa là các mô hình đó có thể được huấn luyện nhanh.

Tuy nhiên, nhiều mô hình PGM trong thực tế rất phức tạp. Việc học và suy diễn đối với chúng đôi khi là bất khả thi hoặc rất khó. Chẳng hạn, nếu chúng ta dùng MLE để học mô hình GMM từ một tập \mathbf{D} thì sẽ cần tìm điểm cực đại của hàm log-likelihood sau:

$$L(\mathbf{D}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi) = \sum_{\mathbf{x} \in \mathbf{D}} \log \sum_{k=1}^K \phi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (101)$$

trong đó $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_k)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$ là hàm mật độ của phân bô chuẩn thứ k . Rõ ràng hàm $L(\mathbf{D}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$ rất phức tạp đối với các tham số $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$. Do đó, các cách làm đơn thuần như ở mục trước sẽ không đi đến một công thức nghiệm cụ thể và tường minh được. Khi đó các giải thuật xấp xỉ sẽ cần được dùng.

Một số PGM không có công thức hàm mật độ tường minh. Khi đó việc học còn phức tạp hơn nữa. Ở các ví dụ trước, chúng ta biết công thức cụ thể của hàm mật độ (hoặc xác suất). Khi đó MAP và MLE có thể dùng để học hoặc suy diễn trực tiếp. Tuy nhiên, điều này không đúng khi ta không biết công thức cụ thể về hàm mật độ. Vài mô hình ví dụ gồm LDA [Blei et al., 2003], CRF [Lafferty et al., 2001], Latent Diffusion [Rombach et al., 2022]. Đôi với các mô hình đó, chúng ta cần dùng một số phương pháp phức tạp hơn, chẳng hạn Cực đại hóa kỳ vọng (Expectation maximization) [Hastie et al., 2017], suy diễn biến phân (Variational inference) [Blei et al., 2017], ...

3.5 Đánh giá hiệu năng và lựa chọn tham số

3.5.1 Giới thiệu về Đánh giá hiệu năng hệ thống học máy

Khi xây dựng một mô hình học máy, chúng ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình khác nhau. Trong phần này, chúng ta sẽ tìm hiểu về các phương pháp đánh giá đối với các mô hình phân loại và hồi qui. Sau khi xây dựng một mô hình học máy và huấn luyện nó trên một tập dữ liệu, điều tiếp theo chúng ta nên làm là đánh giá hiệu năng của mô hình trên tập dữ liệu mới. Việc đánh giá mô hình giúp chúng ta trả lời những câu hỏi sau:

- Mức độ thành công của mô hình huấn luyện thế nào?
- Khi nào nên dừng quá trình huấn luyện?
- Khi nào nên cập nhật mô hình?

Trả lời được các câu hỏi trên, chúng ta có thể quyết định mô hình này có thực sự phù hợp cho bài toán đang xem xét hay không. Tất nhiên, để thu được một đánh giá đáng tin cậy về hiệu năng của một mô hình, chúng ta cần thực hiện một phương pháp đánh giá đúng đắn cũng như lựa chọn tham số tốt cho mô hình. Hiệu năng một mô hình học máy có thể được đánh giá từ góc độ lý thuyết hay đánh giá qua thực nghiệm, trong đó:

- Đánh giá lý thuyết (theoretical evaluation) thực hiện nghiên cứu các khía cạnh lý thuyết của một mô hình mà có thể chứng minh được các yếu tố về:
 - Tốc độ học, thời gian học,
 - Bao nhiêu mẫu học là đủ?
 - Độ chính xác trung bình của hệ thống,
 - Khả năng chống nhiễu,...
- Đánh giá thực nghiệm (experimental evaluation) thực hiện quan sát hệ thống làm việc trong thực tế, sử dụng một hoặc nhiều tập dữ liệu và các tiêu chí đánh giá sau đó tổng hợp đánh giá từ các quan sát đó.

Trong phần tiếp theo, chúng ta sẽ nghiên cứu cách đánh giá thực nghiệm và vấn đề đánh giá mô hình (model assessment) được hiểu là đánh giá hiệu năng của một mô hình học máy A, chỉ dựa trên bộ dữ liệu có nhãn D. Hơn nữa, việc đánh giá hiệu năng của hệ thống được thực hiện một cách tự động, sử dụng một tập dữ liệu mà không cần sự tham

gia hay can thiệp của người dùng. Đánh giá một mô hình có tốt hay không thường được thực hiện trên dữ liệu mà mô hình chưa được huấn luyện. Tỷ lệ thường thấy của một tập dữ liệu huấn luyện so với tập dữ liệu kiểm thử là 70% và 30%. Chúng ta sử dụng dữ liệu mới khi đánh giá mô hình nhằm giảm thiểu khả năng quá khớp (overfitting) đối với tập huấn luyện. Đôi khi sẽ hữu ích khi đánh giá mô hình và cùng lúc huấn luyện nó để tìm ra các thông số được thiết lập tốt nhất của một mô hình. Tuy nhiên, chúng ta không thể sử dụng tập kiểm thử để thực hiện đánh giá này.

Để có thể đánh giá một cách tin cậy một mô hình phân loại cũng như lựa chọn được một giá trị tốt đối với mỗi tham số trong mô hình khi xây dựng và hiệu chỉnh nó, chúng ta cần có một tập dữ liệu đã gán nhãn lớp hay còn được gọi là tập dữ liệu có nhãn. Từ tập dữ liệu này, chúng ta có thể tạo ra một tập dữ liệu con mà được biết đến là tập dữ liệu tối ưu. Một tỷ lệ chia tập dữ liệu điển hình là 60% cho tập huấn luyện, 20% cho tập tối ưu và 20% cho tập kiểm thử, trong đó:

- Tập huấn luyện: Được dùng để huấn luyện mô hình.
- Tập tối ưu: Tùy chọn và được dùng để tối ưu các tham số của mô hình.
- Tập kiểm thử: Được dùng để đánh giá hiệu năng của mô hình đã được huấn luyện. Chúng ta có thể dễ dàng nhận thấy, hiệu năng của một mô hình học máy không chỉ phụ thuộc vào giải thuật học máy mà còn phụ thuộc vào nhiều nhân tố liên quan tới dữ liệu như:
 - Phân bố lớp (Class distribution)
 - Kích thước của tập huấn luyện (Size of the training set)
 - Kích thước tập kiểm thử (Size of the testing set)
 - Tính đại diện của tập luyện trên toàn bộ không gian mẫu

Thông thường, tập huấn luyện càng lớn thì hiệu năng của mô hình học càng tốt trong khi đó tập kiểm thử càng lớn thì việc đánh giá hiệu năng của nó càng chính xác. Mặc dù vậy, chúng ta rất khó hay hiếm khi có thể có được các tập dữ liệu có nhãn rất lớn. Một điều rất quan trọng là độ hỗn loạn của tập dữ liệu có thể được tăng lên đáng kể khi bạn xáo trộn dữ liệu trước khi chia tập dữ liệu ra thành nhiều phần nhỏ. Điều này giúp

cho mỗi tập dữ liệu con có thể thể hiện rõ ràng hơn tính chất đặc thù của tập dữ liệu lớn.

3.5.2 Các phương pháp đánh giá

Phần này sẽ trình bày các phương pháp cho phép đánh giá hiệu năng của một mô hình học máy:

- Hold-out (chia đôi)
- Stratified sampling (lấy mẫu phân tầng)
- Repeated hold-out (chi đôi nhiều lần)
- Cross-validation (đánh giá chéo)
- Leave-one-out
- Bootstrap sampling

1. Hold-out

Toàn bộ tập dữ liệu có nhãn D sẽ được chia thành 2 tập con D_{train} và D_{test} không giao nhau với kích thước của tập luyện khá lớn so với tập kiểm thử, có nghĩa là: $|D_{train}| \gg |D_{test}|$.

- Tập huấn luyện D_{train} được sử dụng để huấn luyện hệ thống
- Tập kiểm thử D_{test} được sử dụng để đánh giá hiệu năng của hệ thống sau khi đã được huấn luyện

Các yêu cầu đối với tập huấn luyện và kiểm thử:

- Dữ liệu thuộc tập kiểm thử D_{test} không được sử dụng trong quá trình huấn luyện hệ thống để đảm bảo các mẫu kiểm thử trong D_{test} cho một đánh giá không thiên vị đối với hiệu năng của hệ thống.
- Dữ liệu thuộc tập huấn luyện D_{train} không được sử dụng trong quá trình đánh giá hệ thống sau khi huấn luyện.

Ví dụ: Ta có thể chọn $|D_{train}| = (2/3).|D|$ và $|D_{test}| = (1/3).|D|$

Cách đánh giá này phù hợp khi chúng ta thu thập được một tập dữ liệu có nhãn (D) có kích thước lớn.

2. Stratified sampling

Khi tập dữ liệu có nhãn có kích thước nhỏ với một số lớp hoặc không cân xứng (unbalanced dataset) đối với các lớp, chẳng hạn tập dữ liệu có ít, hoặc không có các mẫu với một số lớp nhất định, chúng ta sẽ sử dụng phương pháp đánh giá này.

Mục tiêu của phương pháp đánh giá này là đảm bảo phân bố lớp (Class distribution) trong tập huấn luyện và tập kiểm thử phải xấp xỉ như trong tập toàn bộ các mẫu D. Do vậy, chúng ta có:

- Stratified sampling là một phương pháp để cân xứng về phân bố lớp
- Đảm bảo tỉ lệ phân bố lớp trong tập huấn luyện và tập kiểm thử sẽ là xấp xỉ nhau
Phương pháp này không áp dụng được cho bài toán học máy hồi quy vì giá trị đầu ra của bài toán là một giá trị số, không phải là một nhãn lớp.

3. Repeated hold-out

Đây là phương pháp sẽ áp dụng phương pháp đánh giá Hold-out nhiều lần, để sinh ra các tập huấn luyện và kiểm thử khác nhau. Trong mỗi lần lặp, một tỉ lệ nhất định của tập dữ liệu D được chọn ngẫu nhiên nhằm tạo nên tập huấn luyện và có thể kết hợp với phương pháp Stratified sampling khi tập dữ liệu có nhãn không cân xứng.

Tập các giá trị lỗi hoặc các giá trị đo được tương ứng với các tiêu chí đánh giá khác, ghi nhận được trong các bước lặp này sẽ được lấy trung bình cộng để xác định lỗi tổng thể. Tuy nhiên phương pháp này chưa được tốt vì mỗi bước lặp ta lại có một tập kiểm thử khác và có một số mẫu dữ liệu sẽ được dùng nhiều lần trong các tập kiểm thử khác nhau này.

4. Cross validation

Để tránh việc trùng lặp giữa các tập kiểm thử khi một số mẫu dữ liệu cùng xuất hiện trong các tập kiểm thử khác nhau, chúng ta sử dụng phương pháp đánh giá cross-validation.

(i) k-fold cross-validation

Phương pháp này gồm các bước như sau:

- Tập dữ liệu D được chia thành k tập con không giao nhau gọi là “fold” có kích thước xấp xỉ nhau.

- Mỗi lần lặp, một tập con trong k tập sẽ được dùng để làm tập kiểm thử, (k-1) tập còn lại sẽ được sử dụng làm tập huấn luyện.
- k giá trị lỗi trong các lần lặp, trong đó mỗi giá trị lỗi tương ứng với mỗi “fold” sẽ được tính trung bình cộng để thu được giá trị lỗi tổng thể.

Ví dụ: ta có thể chia D thành 10 hoặc 5 folds ($k = 10$ hoặc $k = 5$).

Thông thường, mỗi tập con (fold) được lấy mẫu phân tầng xấp xỉ phân bố lớp trước khi áp dụng quá trình đánh giá Cross-validation. Cách đánh giá này phù hợp khi ta có tập dữ liệu D vừa và nhỏ.

(ii) leave-one-out cross-validation

Phương pháp này là trường hợp đặc biệt của phương pháp Cross-validation với mục đích khai thác tối đa tập dữ liệu ban đầu và không cần có bước lấy mẫu ngẫu nhiên nữa. Cụ thể, trong leave-one-out cross-validation:

- Số lượng các tập con (folds) bằng kích thước của tập dữ liệu ($k = |D|$)
- Mỗi fold chỉ bao gồm 1 mẫu

Phương pháp này thực hiện lặp nhiều lần tương ứng với kích thước của tập dữ liệu nên chi phí tính toán rất cao và chỉ phù hợp khi ta có tập dữ liệu D rất nhỏ. Vả lại, việc áp dụng lấy mẫu phân tầng (stratification) không còn phù hợp khi tập dữ liệu mất cân xứng vì ở mỗi bước lặp, tập kiểm thử chỉ gồm có một mẫu.

5. Bootstrap sampling

Phương pháp Cross-validation sử dụng việc lấy mẫu không lặp lại (sampling without replacement) trong đó, đối với mỗi mẫu, một khi đã được chọn để sử dụng, thì không thể được chọn sử dụng lại cho tập huấn luyện. Phương pháp này sử dụng việc lấy mẫu lặp lại để tạo nên tập huấn luyện. Giả sử toàn bộ tập D bao gồm n mẫu, chúng ta lấy mẫu có lặp lại n lần đối với tập D để tạo nên một tập huấn luyện D_{train} gồm n mẫu như sau:

- Từ tập D, lấy ngẫu nhiên một mẫu x (nhưng không loại bỏ x khỏi D)
- Đưa dữ liệu x vào trong tập huấn luyện
- Lặp lại các bước trên n lần, ta có n mẫu dữ liệu trong tập D_{train}

Sau đó, chúng ta sử dụng các dữ liệu trong tập D_{train} để huấn luyện hệ thống và sử dụng tất cả các dữ liệu thuộc D nhưng không thuộc tập huấn luyện D_{train} để tạo nên tập D_{test} .

Như vậy, trong mỗi bước lặp để tạo nên tập huấn luyện, một mẫu có xác suất $(1 - 1/n)$ để không được lựa chọn đưa vào tập huấn luyện. Vì vậy, xác suất để một mẫu sau quá trình lấy mẫu lặp lại – bootstrap sampling được đưa vào tập kiểm thử là: $(1 - 1/n)^n \approx e^{-1} \approx 0.368$.

Điều này có nghĩa là một tập huấn luyện (có kích thước = n) bao gồm xấp xỉ 63.2% các mẫu trong D và tất nhiên, một mẫu thuộc tập D có thể xuất hiện nhiều lần trong tập D_{train} này. Trong khi đó, tập kiểm thử (có kích thước $< n$) bao gồm xấp xỉ 36.8% các mẫu trong D với một mẫu thuộc tập D chỉ có thể xuất hiện tối đa 1 lần trong tập D_{test} này. Cách đánh giá này phù hợp khi ta có một tập dữ liệu D có kích thước rất nhỏ.

3.5.3 Lựa chọn tham số

Trong các phương pháp học máy khác nhau, nhiều phương pháp có một vài tham số (hyperparameters) mà buộc người dùng phải thiết lập giá trị cho chúng trước khi huấn luyện mô hình. Ví dụ: Ridge regression có tham số λ và Linear SVM có tham số C

Vấn đề lựa chọn giá trị tốt nhất cho các tham số được gọi là bài toán lựa chọn mô hình (model selection). Chúng ta có thể phát biểu bài toán này như sau: Cho một tập học D và phương pháp học A, cần lựa chọn bộ tham số (mô hình) trong phương pháp học A này sao cho hàm học được có khả năng tổng quát hóa cao.

Để đảm bảo hệ thống được huấn luyện tốt nhất từ D, một tập tối ưu (validation set) nhỏ được sử dụng để tối ưu giá trị các tham số trong giải thuật học máy được lấy từ tập D, trong đó, đối với một tham số, giá trị tối ưu là giá trị giúp sinh ra hiệu năng cực đại đối với tập tối ưu. Điều này có nghĩa là, hàm học được chỉ dự đoán tốt trên tập tối ưu và chúng ta xấp xỉ lỗi tổng quát trên toàn bộ không gian dữ liệu bằng cách chỉ sử dụng một tập tối ưu nhỏ.

Sử dụng phương pháp Hold-out, chúng ta có thể lựa chọn tham số λ cho một phương pháp học A từ tập quan sát D như sau:

1. Chọn tập hữu hạn S mà chứa các giá trị tiềm năng cho λ .
2. Chọn độ đo P để đánh giá hiệu năng cho phương pháp học A.

3. Chia D thành 2 tập rời nhau: D_{train} và $T_{validation}$
4. Với mỗi giá trị $\lambda \in S$: Học A từ tập học D_{train} với tham số đầu vào λ . Đo hiệu năng trên tập $T_{validation}$, ta thu được giá trị P_λ
5. Chọn λ^* mà có P_λ tốt nhất.

Chúng ta cũng có thể học lại A từ D với tham số λ^* để hệ thống thu được kết quả tốt hơn. Bên cạnh đó, có thể thay Hold-out bằng các kỹ thuật chia tập dữ liệu khác khác (chẳng hạn như sampling, cross-validation).

3.5.4 Đánh giá và lựa chọn mô hình

Trong thực tế, sẽ là hữu ích hơn khi đánh giá mô hình và cùng lúc huấn luyện nó để tìm ra các giá trị được thiết lập tốt nhất cho các tham số trong mô hình. Tuy nhiên, chúng ta không thể sử dụng tập kiểm thử để thực hiện đánh giá này. Chúng ta có thể phát biểu bài toán đánh giá và lựa chọn mô hình như sau: Cho trước tập quan sát D, ta cần lựa chọn tham số λ (model selection) cho phương pháp học A và đánh giá (assessment) chất lượng tổng thể của A. Sử dụng phương pháp Hold-out, chúng ta có thể lựa chọn tham số λ cho một phương pháp học A và đánh giá chất lượng tổng thể của A từ tập quan sát D như sau:

1. Chọn tập hữu hạn S mà chứa các giá trị tiềm năng cho λ .
2. Chọn độ đo P để đánh giá hiệu năng phương pháp học A .
3. Chia tập D thành 3 tập rời nhau: D_{train} , $T_{validation}$, và T_{test}
4. Với mỗi giá trị $\lambda \in S$: Học A từ tập học D_{train} với tham số đầu vào λ . Đo hiệu năng trên tập $T_{validation}$, ta thu được giá trị P_λ
5. Chọn λ^* mà có P_λ tốt nhất.
6. Huấn luyện A trên tập $D_{train} \cup T_{validation}$, với tham số đầu vào λ^* .
7. Đo hiệu năng của hệ thống trên tập T_{test} .

Chúng ta có thể thay Hold-out bằng các kỹ thuật chia tập dữ liệu khác khác (chẳng hạn như sampling, cross-validation). Một tỷ lệ chia tập dữ liệu điển hình từ tập quan sát D là 60% cho tập huấn luyện, 20% cho tập tối ưu và 20% cho tập kiểm thử.

3.5.5 Các tiêu chí đánh giá

Để một hệ thống học máy có thể sử dụng được trong thực tiễn, chúng ta cần đánh giá hệ thống qua các tiêu chí sau:

- Độ chính xác (Accuracy): phần trăm các dự đoán đúng trên tập kiểm thử
- Tính hiệu quả (Efficiency): Chi phí theo thời gian và bộ nhớ khi học/dự đoán
- Khả năng mở rộng (Scalability): Hiệu năng của hệ thống (vd: tốc độ học/phân loại) thay đổi như thế nào đối với kích thước của tập dữ liệu
- Khả năng diễn giải (Interpretability): Mức độ dễ hiểu (đối với người sử dụng) của các kết quả và hoạt động của hệ thống
- Mức độ phức tạp (Complexity): Mức độ phức tạp của mô hình hệ thống (hàm mục tiêu) học được

Trong phần tiếp theo, chúng ta sẽ tìm hiểu các tiêu chí đánh giá được sử dụng trong cách đánh giá thực nghiệm. Đây là các độ đo để đánh giá hiệu năng của mô hình học máy A chỉ dựa trên bộ dữ liệu có nhãn D. Hơn nữa, việc đánh giá hiệu năng của hệ thống được thực hiện một cách tự động, sử dụng một tập dữ liệu mà không cần sự tham gia hay can thiệp của người dùng.

Độ chính xác trong bài toán phân loại và hồi qui.

Khi xây dựng mô hình phân đoán chúng ta sẽ muốn biết một cách khái quát tỷ lệ các trường hợp được dự đoán đúng trên tổng số các trường hợp trong tập dữ liệu kiểm thử là bao nhiêu. Tỷ lệ đó được gọi là độ chính xác. Độ chính xác giúp ta đánh giá hiệu quả dự đoán của mô hình trên một bộ dữ liệu. Độ chính xác càng cao thì mô hình của chúng ta càng chuẩn xác.

- Đối với bài toán phân loại:

$$\text{Accuracy} = \frac{\text{Số phán đoán chính xác}}{\text{Tổng số phán đoán}}$$

- Đối với bài toán hồi qui: MAE (mean absolute error):

$$MAE = \frac{1}{|D_{test}|} \sum_{x \in D_{test}} |o(x) - y(x)|$$

trong đó:

- $o(x)$: Giá trị đầu ra dự đoán bởi hệ thống đối với ví dụ x .
- $d(x)$: Giá trị đầu ra đúng đối với ví dụ x .

Trong các độ đo đánh giá mô hình phân loại thì độ chính xác khá được ưa chuộng vì nó có công thức tường minh và dễ diễn giải ý nghĩa. Tuy nhiên, hạn chế của nó là đo lường trên tất cả các lớp mà không quan tâm đến độ chính xác trên từng lớp.

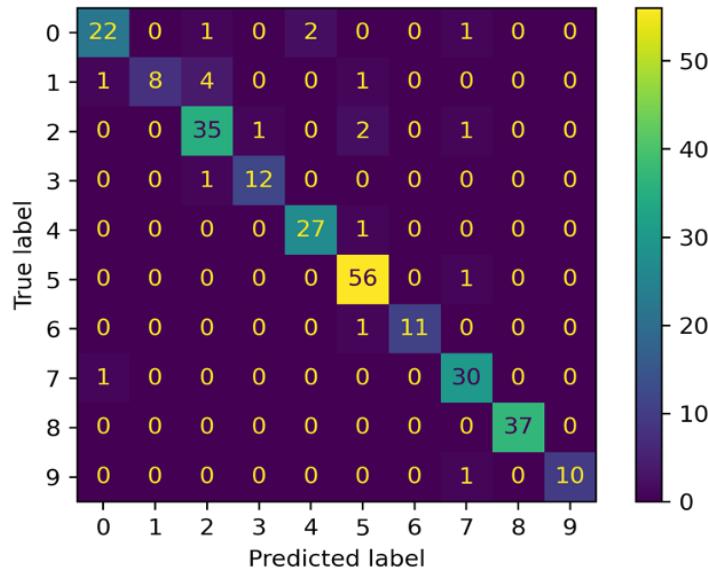
Ma trận nhầm lẫn (Confusion matrix)

Cách tính sử dụng accuracy như ở trên chỉ cho chúng ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi lớp được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là ma trận nhầm lẫn (confusion matrix). Tuy nhiên, ma trận nhầm lẫn chỉ được sử dụng đối với bài toán phân loại, không áp dụng được cho bài toán học máy hồi quy vì giá trị đầu ra của bài toán là một giá trị số, không phải là một nhãn lớp.

Ma trận nhầm lẫn là một ma trận vuông với kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ i , cột thứ j là số lượng mẫu lẻ ra thuộc vào lớp i nhưng lại được dự đoán là thuộc vào lớp j . Chúng ta có thể suy ra ngay rằng tổng các phần tử trong toàn ma trận này chính là số mẫu trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số mẫu được phân loại đúng của mỗi lớp dữ liệu. Từ đây có thể suy ra accuracy chính bằng tổng các phần tử trên đường chéo chia cho tổng các phần tử của toàn ma trận.

Ma trận nhầm lẫn có thể giúp chúng ta nhìn thấy các dự đoán đối với mỗi lớp rất chi tiết trong bài toán phân loại đa lớp. Cụ thể, đối với mỗi lớp c_i , sử dụng ma trận nhầm lẫn, chúng ta xác định được:

- TP_i (true positive): Số lượng các mẫu thuộc lớp c_i được phân loại chính xác vào lớp c_i
- FP_i (false positive): Số lượng các mẫu không thuộc lớp c_i bị phân loại nhầm vào lớp c_i
- TN_i (true negative): Số lượng các mẫu không thuộc lớp c_i được phân loại chính xác



Hình 41: Một ma trận nhầm lẫn đối với bài toán phân loại 10 lớp.

Lớp c_i		Được phân lớp bởi hệ thống	
		Thuộc	Ko thuộc
Phân lớp thực sự (đúng)	Thuộc	TP_i	FN_i
	Ko thuộc	FP_i	TN_i

Hình 42: Mối liên hệ giữa các đánh giá.

- FN_i (false negative): Số lượng các mẫu thuộc lớp c_i bị phân loại nhầm vào các lớp khác c_i

Từ các đại lượng trên, chúng ta có thể tính được độ chính xác (Precision) và độ tin cậy (Recall) đối với mỗi lớp. Các độ đo này rất hay được sử dụng để đánh giá các hệ thống phân loại văn bản. Precision đối với lớp c_i là tổng số các mẫu thuộc lớp c_i được phân loại chính xác chia cho tổng số các mẫu được phân loại vào lớp c_i

$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

Recall đối với lớp c_i là tổng số các mẫu thuộc lớp c_i được phân loại chính xác chia

cho tổng số các mẫu thuộc lớp c_i

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

Một câu hỏi đặt ra là làm thế nào để tính toán được giá trị Precision và Recall một cách tổng thể cho toàn bộ các lớp $C=c_i$? Ở đây, chúng ta sẽ đưa ra 2 cách tính trung bình vi mô và vĩ mô đối với các giá trị Precision và Recall của các lớp trong C.

Trung bình vi mô (Micro-averaging):

$$Precision = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FP_i)}$$

$$Recall = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FN_i)}$$

Trung bình vĩ mô (Macro-averaging):

$$Precision = \frac{\sum_{i=1}^{|C|} Precision(c_i)}{|C|}$$

$$Recall = \frac{\sum_{i=1}^{|C|} Recall(c_i)}{|C|}$$

Bằng một chút suy luận logic, ta có thể thấy thường khi Precision cao thì Recall thấp và ngược lại. Khi đó rất khó để lựa chọn đâu là một mô hình tốt vì không biết rằng đánh giá trên Precision hay Recall sẽ phù hợp hơn. Chính vì vậy chúng ta sẽ tìm cách kết hợp cả Precision và Recall trong một tiêu chí mới, đó chính là F_1 .

$$F_1 = \frac{2.Precision.Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

F_1 là một trung bình điều hòa (harmonic mean) của các tiêu chí Precision và Recall, trong đó F_1 có xu hướng lấy giá trị gần với giá trị nào nhỏ hơn giữa 2 giá trị Precision và Recall F_1 có giá trị lớn nếu cả 2 giá trị Precision và Recall đều lớn

3.5.6 Tóm tắt

- Accuracy là tỉ lệ giữa số điểm được phân loại đúng và tổng số điểm. Accuracy chỉ phù hợp với các bài toán mà kích thước các lớp dữ liệu là tương đối như nhau.
- Ma trận nhầm lẫn giúp có cái nhìn rõ hơn về việc các điểm dữ liệu được phân loại đúng/sai như thế nào.

3.6 Mạng nơ-ron nhân tạo

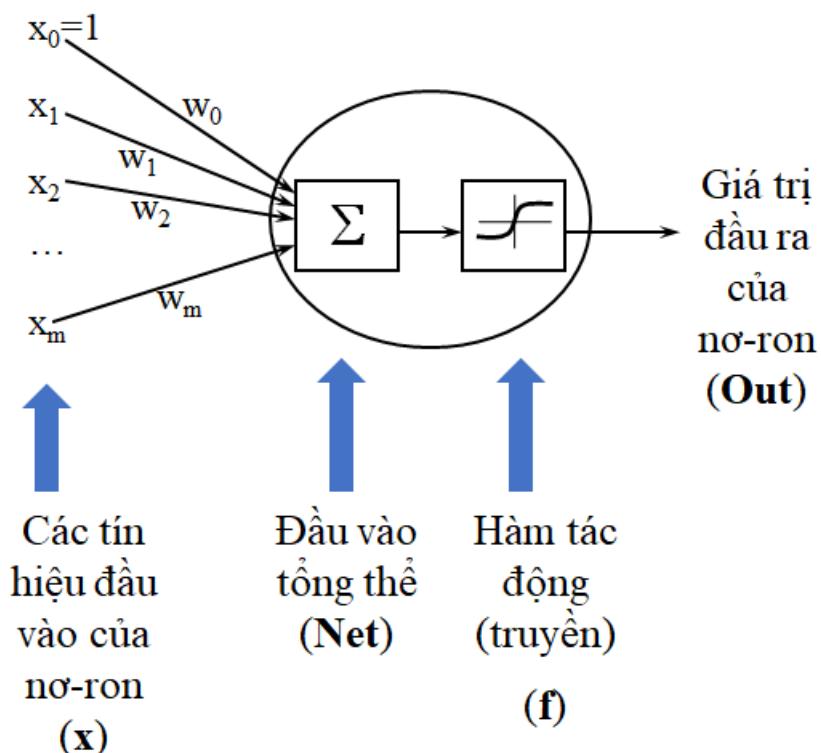
Mạng nơ-ron nhân tạo (Artificial Neural Network - ANN) là một mô hình tính toán lấy cảm hứng từ cách hệ thống nơ-ron trong não người hoạt động. Mục tiêu của mạng nơ-ron nhân tạo là mô phỏng khả năng học và xử lý thông tin giống như con người. Một mạng nơ-ron nhân tạo bao gồm các đơn vị xử lý cơ bản được gọi là nơ-ron, và các kết nối giữa chúng được trọng số hóa. Mỗi nơ-ron nhận đầu vào, xử lý thông tin bằng cách áp dụng một hàm kích thích, và sau đó tạo ra đầu ra. Quá trình này được lặp lại qua nhiều lớp nơ-ron để tạo ra một mô hình có khả năng học từ dữ liệu.

ANN có thể được xem như một cấu trúc xử lý thông tin một cách phân tán và song song ở mức cao. ANN có khả năng học (learn), nhớ lại (recall), và khái quát hóa (generalize) từ các dữ liệu học. Khả năng của một ANN phụ thuộc vào: Kiến trúc (topology) của mạng nơ-ron, đặc tính đầu vào/ra của mỗi nơ-ron, thuật toán học (huấn luyện) và dữ liệu học.

3.6.1 Kiến trúc của một nơ-ron

Cấu trúc và hoạt động của một nơ-ron bao gồm những thành phần như sau (Hình 43):

- Các tín hiệu đầu vào (input signals) của nơ-ron ($x_i, i = 1..m$)
- Mỗi tín hiệu đầu vào xi gắn với một trọng số w_i
- Trọng số điều chỉnh (bias) w_0 (với $x_0 = 1$)
- Đầu vào tổng thể (Net input) là một hàm tích hợp của các tín hiệu đầu vào – $Net(\mathbf{w}, \mathbf{x})$



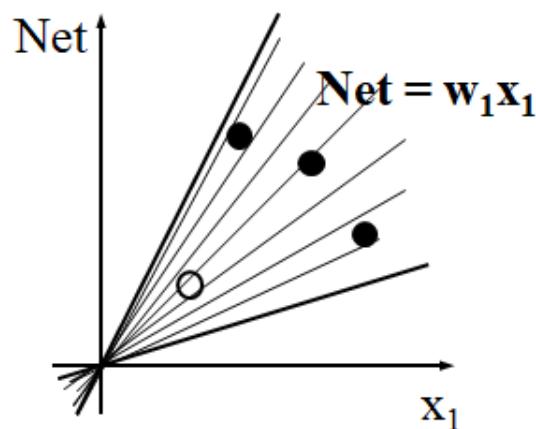
Hình 43: Kiến trúc của một nơ-ron

- Hàm tác động/truyền (Activation/transfer function) tính giá trị đầu ra của nơ-ron – $f(Net(w,x))$
- Giá trị đầu ra (Output) của nơ-ron: $Out = f(Net(w,x))$

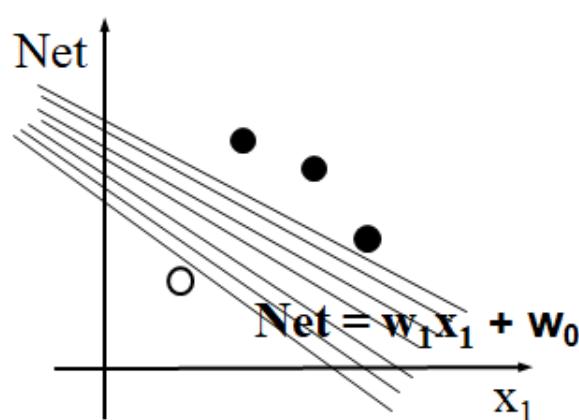
Đầu vào tổng thể (net input) thường được tính toán bởi một hàm tuyến tính

$$\begin{aligned}
 Net &= w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n \\
 &= w_0 \cdot 1 + \sum_{i=1}^m w_i x_i \\
 &= \sum_{i=0}^m w_i x_i
 \end{aligned}$$

Trọng số điều chỉnh (bias) w_0 giúp mở rộng hàm tuyến tính đầu vào tổng thể (net input) theo dữ liệu đầu vào thay vì chỉ là họ hàm tuyến tính đi qua gốc tọa độ. Ví dụ, chúng ta xét dữ liệu trên không gian một chiều như Hình 44, 45. Họ các hàm $Net=w_1x_1$ không thể phân tách được các ví dụ thành 2 lớp (two classes) 44 nhưng họ các hàm $Net=w_1x_1 + w_0$ có thể phân tách được 45.



Hình 44: Hàm Net không thể phân tách được hai lớp khi không sử dụng bias.



Hình 45: Hàm Net không thể phân tách được hai lớp khi sử dụng bias.

Hàm kích thích (activation function), còn được gọi là hàm truyền (transfer function), là một phần quan trọng trong mỗi nơ-ron nhân tạo (ANN). Nhiệm vụ của hàm kích thích là định rõ cách mà đầu vào được chuyển đổi thành đầu ra của nơ-ron trong mạng. Các hàm kích thích khác nhau có ảnh hưởng đến khả năng học tập và hiệu suất của mô hình. Lựa chọn hàm kích thích phụ thuộc vào đặc điểm của vấn đề cụ thể và cách mà mô hình được cấu hình.

Dưới đây là một số hàm kích thích phổ biến được sử dụng trong mạng nơ-ron nhân tạo:

Giới hạn cứng: Còn được gọi là hàm ngưỡng (threshold function). Giá trị đầu ra lấy một trong 2 giá trị 1 hoặc 0 phụ thuộc theo giá trị ngưỡng θ như sau:

$$\begin{aligned} Out(Net) &= HL(Net, \theta) \\ &= \begin{cases} 1 & \text{nếu } Net > \theta \\ 0 & \text{ngược lại} \end{cases} \end{aligned}$$

Đầu ra của hàm có thể thay đổi khi nhận giá trị 1 hoặc –1 theo hàm dấu như sau:

$$\begin{aligned} Out(Net) &= HL2(Net, \theta) \\ &= \text{sign}(Net, \theta) \end{aligned}$$

Tuy hàm giới hạn cứng đơn giản, dễ tính toán, nhưng chúng không liên tục, không có đạo hàm tại ngưỡng.

Hàm tác động logic ngưỡng: Còn được gọi là hàm tuyến tính bão hòa (saturating linear function). Hàm logic ngưỡng kết hợp của 2 hàm tác động: tuyến tính và giới hạn

chặt. Công thức chi tiết của hàm logic ngưỡng như sau:

$$\begin{aligned} Out(Net) &= tl(Net, \alpha, \theta) \\ &= \begin{cases} 0 & \text{nếu } Net < -\theta \\ \alpha(Net + \theta) & \text{nếu } -\theta \leq Net \leq \frac{1}{\alpha} - \theta \\ 1 & \text{nếu } Net > \frac{1}{\alpha} - \theta \end{cases} \\ &= \max(0, \min(1, \alpha(Net + \theta))) \end{aligned}$$

trong đó, tham số α xác định độ dốc của khoảng tuyênn tính. Hàm logic ngưỡng liên tục, nhưng không có đạo hàm.

Hàm tác động sigmoid: Hàm sigmoid giúp chuyển đổi đầu vào thành giá trị nằm trong khoảng $(0, 1)$. Nó thường được sử dụng trong các lớp đầu ra của mô hình để thực hiện phân loại nhị phân. Công thức chi tiết của hàm Sigmoid như sau:

$$\begin{aligned} Out(Net) &= sf(Net, \alpha, \theta) \\ &= \frac{1}{1 + e^{-Net}} \end{aligned}$$

Giá trị đầu ra của hàm sigmoid trong khoảng $(0, 1)$. Hàm sigmoid liên tục và đạo hàm liên tục. Đạo hàm của một hàm sigmoid được biểu diễn bằng một hàm của chính nó. Bởi vì có nhiều khoảng đầu vào làm cho đầu ra của hàm sigmoid gần 0 hoặc 1, nên đạo hàm của hàm sigmoid dễ thu được giá trị nhỏ, gần 0 (vanishing gradient). Vì vậy, sử dụng hàm sigmoid có thể dẫn đến hiện tượng mạng không được cập nhật tham số trong quá trình học.

Trong nhiều trường hợp, chúng ta có thể lựa chọn hàm Logistic (hàm tổng quát hơn của hàm sigmoid):

$$\begin{aligned} Out(Net) &= sf(Net, \alpha, \theta) \\ &= \frac{1}{1 + e^{-\alpha(Net + \theta)}} \end{aligned}$$

với θ và α là các tham số phải lựa chọn trong quá trình thử nghiệm. Hàm logistic như công thức trên cũng có các tính chất như hàm sigmoid.

Hàm tác động Hyperbolic tangent: Tương tự như sigmoid, hàm tanh giúp chuyển

đổi đầu vào thành giá trị nằm trong khoảng $(-1, 1)$. Hàm này thường được sử dụng trong các lớp ẩn của mô hình. Công thức chi tiết của hàm như sau:

$$\text{Out(Net)} = \tanh(\text{Net}) = \frac{1 - e^{-2\text{Net}}}{1 + e^{-2\text{Net}}} = \frac{2}{1 + e^{-2\text{Net}}} - 1$$

Tính chất của hàm này tương tự như hàm sigmoid: Liên tục, có đạo hàm liên tục và đạo hàm của một hàm tanh có thể được biểu diễn bằng một hàm của chính nó. Tổng quát hơn, chúng ta có thể sử dụng hàm sau:

$$\text{Out(Net)} = \tanh(\text{Net}, \alpha, \theta) = \frac{1 - e^{-\alpha(\text{Net}+\theta)}}{1 + e^{-\alpha(\text{Net}+\theta)}} = \frac{2}{1 + e^{-\alpha(\text{Net}+\theta)}} - 1$$

với θ và α là các tham số phải lựa chọn trong quá trình thử nghiệm.

Hàm ReLU (Rectified Linear Unit):

$$f(x) = \max(0, x)$$

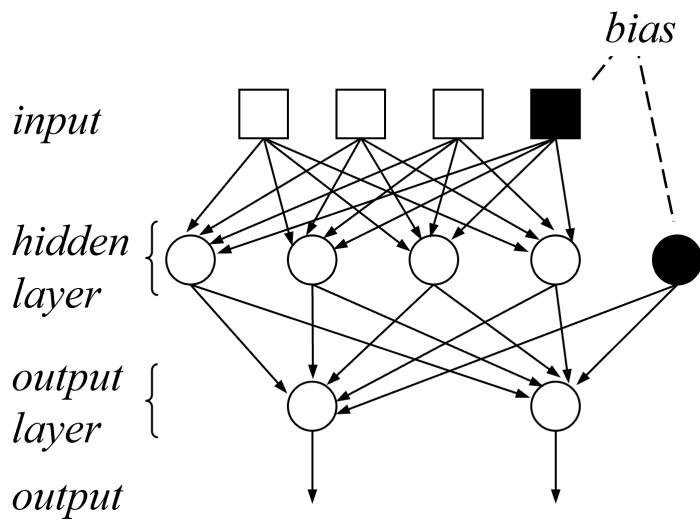
ReLU giữ nguyên giá trị dương của đầu vào và đưa về 0 nếu giá trị âm. Nó đã trở thành một lựa chọn phổ biến cho các lớp ẩn, vì nó giúp giảm vấn đề biến mất đạo hàm và có tính chất kích thích sự học tập của mạng.

3.6.2 Kiến trúc mạng ANN

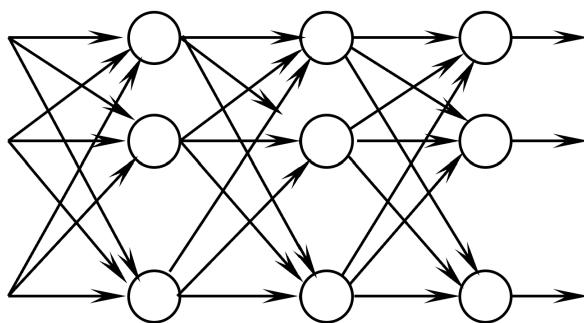
Kiến trúc của một ANN được xác định bởi các thành phần sau:

- Số lượng các tín hiệu đầu vào và đầu ra
- Số lượng các tầng
- Số lượng các nơ-ron trong mỗi tầng
- Số lượng các liên kết đối với mỗi nơ-ron
- Cách thức các nơ-ron (trong một tầng, hoặc giữa các tầng) liên kết với nhau

Một ANN phải có: Một tầng đầu vào (input layer), một tầng đầu ra (output layer) và có thể có hoặc không nhiều tầng ẩn (hidden layer(s)). Trong đó, một tầng (layer) chứa một nhóm các nơ-ron. Tầng ẩn (hidden layer) là một tầng nằm ở giữa tầng đầu



Hình 46: Kiến trúc của mạng nơ-ron. Một ANN với một tầng ẩn gồm 3 tín hiệu vào, 2 giá trị đầu ra. Tổng cộng, có 6 neurons (4 ở tầng ẩn và 2 ở tầng đầu ra).

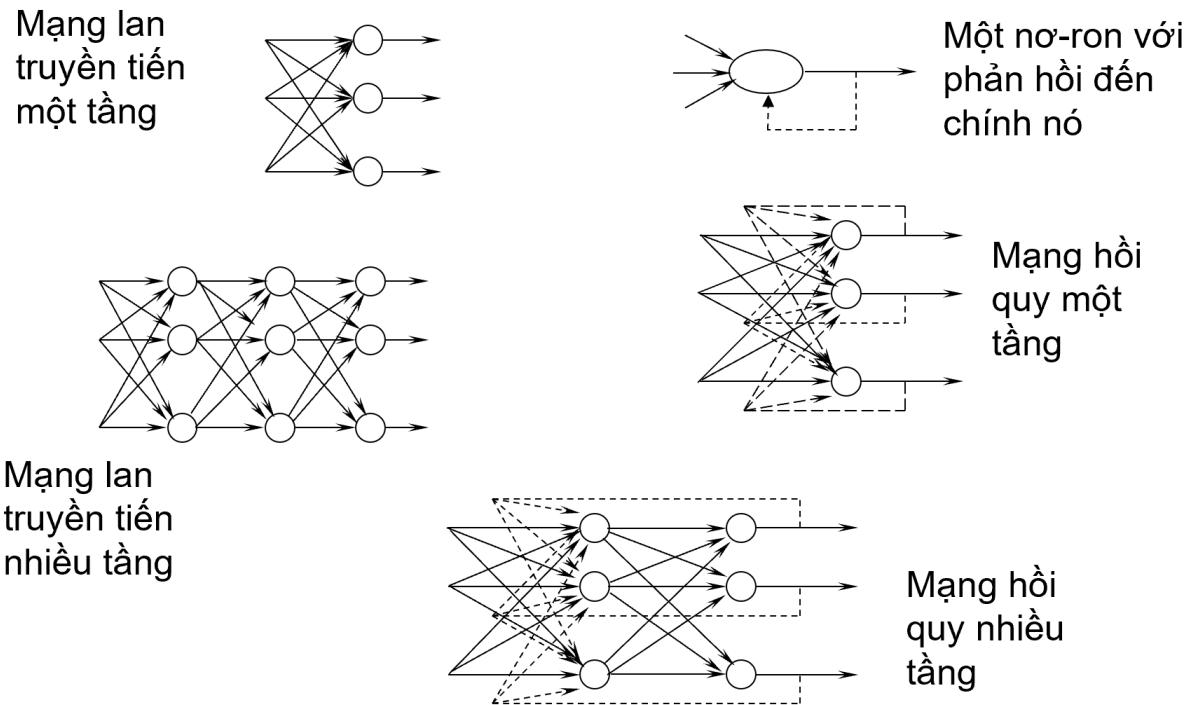


Hình 47: Kiến trúc của mạng nơ-ron đầy đủ

vào (input layer) và tầng đầu ra (output layer). Các nút ở tầng ẩn (hidden nodes) không tương tác trực tiếp với môi trường bên ngoài (của mạng nơ-ron). Hình 46 minh họa kiến trúc mạng gồm một tầng ẩn.

Một số khái niệm kiến trúc mạng phổ biến được trình bày như sau (Hình 48):

- Một ANN được gọi là liên kết đầy đủ (fully connected) (Hình 47) nếu mọi đầu ra từ một tầng liên kết với mọi nơ-ron của tầng kế tiếp.
- Một ANN được gọi là mạng lan truyền tiến (feed-forward network) nếu không có bất kỳ đầu ra của một nút là đầu vào của một nút khác thuộc cùng tầng (hoặc thuộc một tầng phía trước).
- Khi các đầu ra của một nút liên kết ngược lại làm các đầu vào của một nút thuộc cùng tầng (hoặc thuộc một tầng phía trước), thì đó là một mạng phản hồi (feedback network)



Hình 48: Một số kiến trúc của mạng nơ-ron

- Nếu phản hồi là liên kết đầu vào đối với các nút thuộc cùng tầng, thì đó là phản hồi bên (lateral feedback).
- Các mạng phản hồi có các vòng lặp kín (closed loops) được gọi là các mạng hồi quy (recurrent networks).

3.6.3 Mạng perceptron

Mạng perceptron là một mô hình nơ-ron nhân tạo đơn giản nhất được đề xuất bởi Frank Rosenblatt vào những năm 1950. Nó là một dạng cơ bản của mạng nơ-ron chỉ gồm duy nhất một nơ-ron. Mô hình này được sử dụng chủ yếu để thực hiện tác vụ phân loại nhị phân. Biểu diễn của mạng perceptron

$$Out = \text{sign}(Net(w, x)) = \text{sign}\left(\sum_{j=0}^m w_j x_j\right)$$

Đối với một ví dụ x , giá trị đầu ra của perceptron là 1, nếu $Net(w, x) > 0$ và -1 trong trường hợp còn lại.

Với một tập các ví dụ học $D = \{(x, d)\}$, với x là đầu vào, d là giá trị đầu ra mong

Algorithm 1 Perceptron_batch(D, η)

```

1: Initialize  $w$  ( $w_i \leftarrow$  an initial (small) random value)
2: repeat
3:    $\Delta w \leftarrow 0$ 
4:   for each training instance  $(x, d) \in D$  do
5:     Compute the real output value  $Out$ 
6:     if  $(Out \neq d)$  then
7:        $\Delta w \leftarrow \Delta w + \eta(d - Out)x$ 
8:     end if
9:   end for
10:   $w \leftarrow w + \Delta w$ 
11: until all the training instances in  $D$  are correctly classified
12: return  $w$ 

```

muốn (-1 hoặc 1), quá trình học của perceptron nhằm xác định một vectơ trọng số cho phép perceptron sinh ra giá trị đầu ra chính xác (-1 hoặc 1) cho mỗi ví dụ học. Với một ví dụ học x được perceptron phân lớp chính xác, thì vectơ trọng số w không thay đổi. Trong trường hợp khác thì:

- Nếu $d = 1$ nhưng perceptron lại sinh ra -1 ($Out = -1$), thì w cần được thay đổi sao cho giá trị $Net(w, x)$ tăng lên.
- Nếu $d = -1$ nhưng perceptron lại sinh ra 1 ($Out = 1$), thì w cần được thay đổi sao cho giá trị $Net(w, x)$ giảm đi.

Thuật toán 1 mô tả quá trình học của mạng perceptron. Giải thuật học cho perceptron được chứng minh là hội tụ (converge) nếu các ví dụ học là có thể phân tách tuyến tính (linearly separable) và sử dụng một tốc độ học η đủ nhỏ. Tuy nhiên, giải thuật học perceptron có thể không hội tụ nếu như các ví dụ học không thể phân tách tuyến tính (not linearly separable). Trong khi một perceptron chỉ có thể biểu diễn một hàm phân tách tuyến tính (linear separation function), một mạng nơ-ron nhiều tầng (multi-layer NN) được học bởi giải thuật lan truyền ngược (Back Propagation - BP) có thể biểu diễn một hàm phân tách phi tuyến phức tạp (highly non-linear separation function). Chúng ta sẽ xem xét cách học mạng nơ-ron tổng quát trong phần tiếp theo.

3.6.4 Huấn luyện mạng nơ-ron tổng quát

Có hai kiểu học phổ biến trong các mạng nơ-ron nhân tạo: Học tham số (Parameter learning) và học cấu trúc (Structure learning). Mục tiêu của học tham số (Parameter learning) là thay đổi thích nghi các trọng số (weights) của các liên kết trong mạng nơ-

ron. Trong khi đó, mục tiêu của học cấu trúc là thay đổi thích nghi cấu trúc mạng, bao gồm số lượng các nơ-ron và các kiểu liên kết giữa chúng. Hai kiểu học này có thể được thực hiện đồng thời hoặc riêng rẽ. Trong nội dung mô học này, chúng ta sẽ chỉ xét việc học tham số.

Huấn luyện một mạng nơron (khi cố định kiến trúc) chính là việc học các trọng số w của mạng từ tập học D . Thông thường, chúng ta đưa việc học tham số mạng về bài toán cực tiểu hoá một hàm lỗi thực nghiệm:

$$L(\mathbf{w}) = \frac{1}{|\mathbf{D}|} \sum_{x \in \mathbf{D}} loss(d_x, out(x))$$

trong đó $out(x)$ là đầu ra của mạng, với đầu vào x có nhãn tương ứng là d_x ; $loss$ là một hàm đo lỗi phán đoán. Với hồi quy hàm lỗi bình phương thường được sử dụng. Với phân loại, hàm cross-entropy thường được sử dụng để đo sự sai khác của kết quả đầu ra của mạng và giá trị mong muốn thực tế. Để cập nhật tham số, nhiều phương pháp lặp dựa trên Gradient: Backpropagation, SGD, Adam, AdaGrad, etc.

Xét một ANN có n nơ-ron đầu ra, chúng ta sử dụng lỗi bình phương cho bài toán hồi quy. Đối với một ví dụ học (x, d) , giá trị lỗi học (training error) gây ra bởi vectơ trọng số (hiện tại) w :

$$E_x(w) = \frac{1}{2} \sum_{i=1}^n (d_i - out_i)^2$$

Hàm lỗi gây ra bởi vectơ trọng số (hiện tại) w đối với toàn bộ tập học D :

$$E_D(w) = \frac{1}{|D|} \sum_{x \in D} E_x(w)$$

Gradient của E (ký hiệu là ∇E) là một vectơ:

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_N} \right)$$

trong đó N là tổng số các trọng số (các liên kết) trong mạng. Gradient ∇E xác định hướng gây ra việc tăng nhanh nhất (steepest increase) đối với giá trị lỗi E . Trong khi đó, hướng gây ra việc giảm nhanh nhất (steepest decrease) là hướng ngược với gradient

Algorithm 2 Gradient_descent_incremental(D, η)

```

1: Initialize  $w$  ( $w_i \leftarrow$  an initial (small) random value)
2: repeat
3:   for each training instance  $(x, d) \in D$  do
4:     Compute the network output
5:     for each weight component  $w_i$  do
6:        $w_i \leftarrow w_i - \eta \left( \frac{\partial E_x}{\partial w_i} \right)$ 
7:     end for
8:   end for
9: until stopping criterion satisfied
10: return  $w$ 

```

của E :

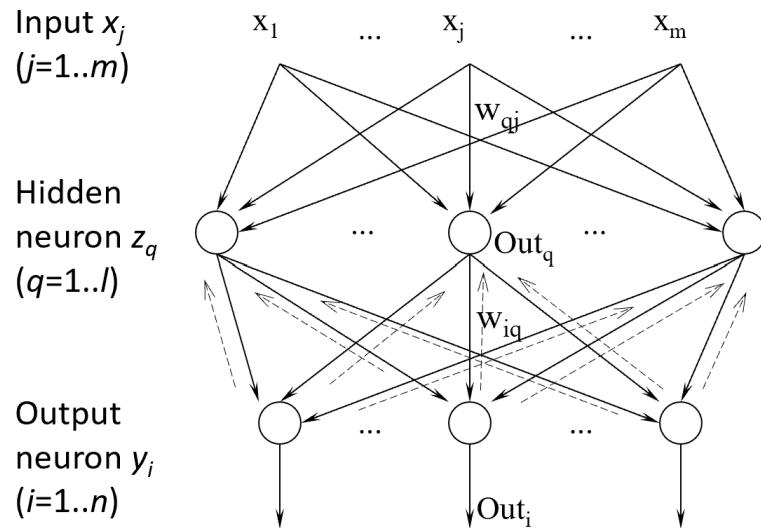
$$\delta W = -\eta \cdot \nabla E(w)$$

$$\Delta W_i = -\eta \frac{\partial E}{\partial w_i}, \forall i = 1..N$$

Giải thuật học lan truyền ngược BP (backpropagation) được sử dụng để học các trọng số của một mạng nơ-ron nhiều tầng với cấu trúc mạng cố định (các nơ-ron và các liên kết giữa chúng là cố định) và đối với mỗi nơ-ron, hàm tác động phải có đạo hàm liên tục (hoặc có chiến lược giải quyết tại một vài điểm không tồn tại đạo hàm). Giải thuật BP áp dụng chiến lược gradient descent (Alg.2) trong quy tắc cập nhật các trọng số để cực tiểu hóa lỗi (khác biệt) giữa các giá trị đầu ra thực tế và các giá trị đầu ra mong muốn, đối với các ví dụ học.

Giải thuật học lan truyền ngược tìm kiếm một vectơ các trọng số (weights vector) giúp cực tiểu hóa lỗi tổng thể của hệ thống đối với tập học. Giải thuật BP bao gồm 2 giai đoạn (bước):

- Giai đoạn lan truyền tín hiệu (Signal forward). Các tín hiệu đầu vào (vectơ các giá trị đầu vào) được lan truyền từ tầng đầu vào đến tầng đầu ra (đi qua các tầng ẩn).
- Giai đoạn lan truyền ngược lỗi (Error backward). Căn cứ vào giá trị đầu ra mong muốn của vectơ đầu vào, hệ thống tính toán giá trị lỗi. Bắt đầu từ tầng đầu ra, giá trị lỗi được lan truyền ngược qua mạng, từ tầng này qua tầng khác (phía trước), cho đến tầng đầu vào. Việc lan truyền ngược lỗi (error back-propagation) được thực hiện thông qua việc tính toán (một cách truy hồi) giá trị gradient cục bộ của mỗi nơ-ron.



Hình 49: Quá trình lan truyền tiến (nét liền) và lan truyền ngược (nét đứt) trong học mạng ANN.

Chúng ta sử dụng mạng nơ-ron 3 tầng (Hình 49) để minh họa giải thuật học BP như sau:

- m tín hiệu đầu vào x_j ($j = 1..m$), l nơ-ron tầng ẩn z_q ($q = 1..l$), n nơ-ron đầu ra y_i ($i = 1..n$)
- w_{qj} là trọng số của liên kết từ tín hiệu đầu vào x_j tới nơ-ron tầng ẩn z_q
- w_{iq} là trọng số của liên kết từ nơ-ron tầng ẩn z_q tới nơ-ron đầu ra y_i
- Out_q là giá trị đầu ra (cục bộ) của nơ-ron tầng ẩn z_q
- Out_i là giá trị đầu ra của mạng tương ứng với nơ-ron đầu ra y_i

Bước lan truyền tiến: Đối với mỗi ví dụ học x , vectơ đầu vào x được lan truyền từ tầng đầu vào đến tầng đầu ra. Mạng sẽ sinh ra một giá trị đầu ra thực tế (actual output) Out (là một vectơ của các giá trị Out_i , $i = 1..n$). Nơ-ron z_q ở tầng ẩn sẽ nhận được giá trị đầu vào tổng thể (*netinput*) bằng:

$$Net_q = \sum_{j=1}^m w_{qj}x_j$$

và sinh ra một giá trị đầu ra (cục bộ) bằng:

$$Out_q = f \left(\sum_{j=1}^m w_{qj} x_j \right)$$

trong đó $f(\cdot)$ là hàm tác động (activation function) của nơ-ron z_q . Tiếp theo, giá trị đầu vào tổng thể (net input) của nơ-ron y_i ở tầng đầu ra:

$$Net_i = \sum_{q=1}^l w_{iq} Out_q = \sum_{q=1}^l w_{iq} f \left(\sum_{j=1}^m w_{qj} x_j \right)$$

Nơ-ron y_i sinh ra giá trị đầu ra (là một giá trị đầu ra của mạng):

$$Out_i = f(Net_i) = f \left(\sum_{q=1}^l w_{iq} Out_q \right) = f \left(\sum_{q=1}^l w_{iq} f \left(\sum_{j=1}^m w_{qj} x_j \right) \right)$$

Vectơ các giá trị đầu ra Out_i ($i=1..n$) chính là giá trị đầu ra thực tế của mạng, đối với vectơ đầu vào x . Từ đó, đối với mỗi ví dụ học x , các tín hiệu lỗi (error signals) do sự khác biệt giữa giá trị đầu ra mong muốn d và giá trị đầu ra thực tế Out được tính toán. Các tín hiệu lỗi này được lan truyền ngược (back-propagated) từ tầng đầu ra tới các tầng phía trước, để cập nhật các trọng số (weights).

Quá trình lan truyền ngược: Xét các tín hiệu lỗi và việc lan truyền ngược của chúng với một hàm lỗi sau:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^n (d_i - Out_i)^2 = \frac{1}{2} \sum_{i=1}^n [d_i - f(Net_i)]^2 \\ &= \frac{1}{2} \sum_{i=1}^n \left[d_i - f \left(\sum_{q=1}^l w_{iq} Out_q \right) \right]^2 \end{aligned}$$

Theo phương pháp gradient-descent, các trọng số của các liên kết từ tầng ẩn tới tầng đầu ra được cập nhật bởi

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

Sử dụng quy tắc chuỗi đạo hàm đối với $\partial E / \partial w_{iq}$, ta có

$$\begin{aligned}\Delta w_{iq} &= -\eta \frac{\partial E}{\partial \text{Out}_i} \frac{\partial \text{Out}_i}{\partial \text{Net}_i} \frac{\partial \text{Net}_i}{\partial w_{iq}} \\ &= \eta (d_i - \text{Out}_i) f'(\text{Net}_i) \text{Out}_q \\ &= \eta \delta_i \text{Out}_q\end{aligned}$$

Chú ý rằng dấu “–” đã được kết hợp với giá trị $\partial E / \partial \text{Out}_i$. Gọi δ_i là tín hiệu lỗi (error signal) của nơ-ron y_i ở tầng đầu ra, chúng ta có:

$$\begin{aligned}\delta_i &= -\frac{\partial E}{\partial \text{Net}_i} = -\frac{\partial E}{\partial \text{Out}_i} \frac{\partial \text{Out}_i}{\partial \text{Net}_i} \\ &= (d_i - \text{Out}_i) f'(\text{Net}_i)\end{aligned}$$

trong đó Net_i là đầu vào tổng thể (net input) của nơ-ron y_i ở tầng đầu ra, và $f'(\text{Net}_i) = \partial f(\text{Net}_i) / \partial \text{Net}_i$. Để cập nhật các trọng số của các liên kết từ tầng đầu vào tới tầng ẩn, chúng ta cũng áp dụng phương pháp gradient-descent và quy tắc chuỗi đạo hàm

$$\Delta w_{qj} = -\eta \frac{\partial E}{\partial w_{qj}} = -\eta \frac{\partial E}{\partial \text{Out}_q} \frac{\partial \text{Out}_q}{\partial \text{Net}_q} \frac{\partial \text{Net}_q}{\partial w_{qj}}$$

Từ công thức tính hàm lỗi $E(w)$, ta thấy rằng mỗi thành phần lỗi $(d_i - y_i)$ ($i = 1..n$) là một hàm của Out_q :

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \left[d_i - f \left(\sum_{q=1}^l w_{iq} \text{Out}_q \right) \right]^2$$

Áp dụng quy tắc chuỗi đạo hàm, ta có:

$$\begin{aligned}\Delta w_{qj} &= \eta \sum_{i=1}^n [(d_i - \text{Out}_i) f'(\text{Net}_i) w_{iq}] f'(\text{Net}_q) x_j \\ &= \eta \sum_{i=1}^n [\delta_i w_{iq}] f'(\text{Net}_q) x_j = \eta \delta_q x_j\end{aligned}$$

Gọi δ_q là tín hiệu lỗi (error signal) của nơ-ron z_q ở tầng ẩn, chúng ta có

$$\begin{aligned}\delta_q &= -\frac{\partial E}{\partial \text{Net}_q} = -\frac{\partial E}{\partial \text{Out}_q} \frac{\partial \text{Out}_q}{\partial \text{Net}_q} \\ &= f'(\text{Net}_q) \sum_{i=1}^n (\delta_i) w_{iq}\end{aligned}$$

trong đó Net_q là đầu vào tổng thể (net input) của nơ-ron z_q ở tầng ẩn, và $f'(\text{Net}_q) = \partial f(\text{Net}_q) / \partial \text{Net}_q$.

Theo các công thức tính các tín hiệu lỗi δ_i và δ_q đã nêu, thì tín hiệu lỗi của một nơ-ron ở tầng ẩn khác với tín hiệu lỗi của một nơ-ron ở tầng đầu ra. Do sự khác biệt này, thủ tục cập nhật trọng số trong giải thuật BP còn được gọi là quy tắc học delta tổng quát. Tín hiệu lỗi δ_q của nơ-ron z_q ở tầng ẩn được xác định bởi: Các tín hiệu lỗi δ_i của các nơ-ron y_i ở tầng đầu ra (mà nơ-ron z_q liên kết tới) và các hệ số chính là các trọng số w_{iq} .

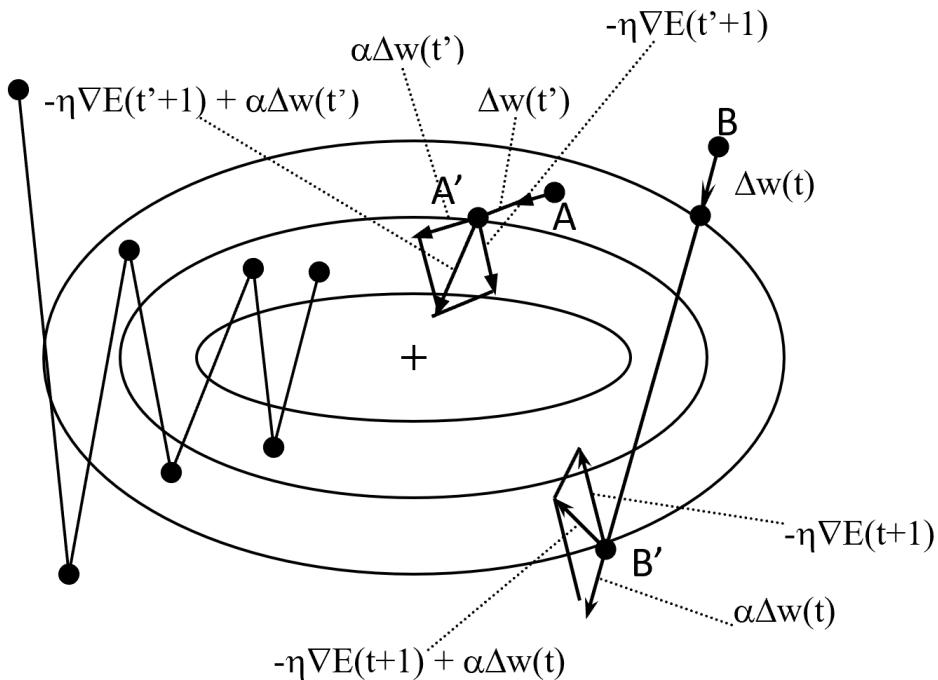
Quá trình tính toán tín hiệu lỗi (error signals) như trên có thể được mở rộng (khái quát) dễ dàng đối với mạng nơ-ron có nhiều hơn 1 tầng ẩn. Dạng tổng quát của quy tắc cập nhật trọng số trong giải thuật BP là: $\Delta W_{ab} = \eta \delta_a x_b$ với b và a là 2 chỉ số tương ứng với 2 đầu của liên kết ($b \rightarrow a$) (từ một nơ-ron (hoặc tín hiệu đầu vào) b đến nơ-ron a), x_b là giá trị đầu ra của nơ-ron ở tầng ẩn (hoặc tín hiệu đầu vào) b và δ_a là tín hiệu lỗi của nơ-ron a .

3.6.5 Một số vấn đề với mạng ANN

Khởi tạo giá trị của các trọng số cho mạng: Vấn đề khởi tạo trọng số là một phần quan trọng của quá trình huấn luyện mạng nơ-ron nhân tạo (ANN). Việc chọn phương pháp khởi tạo trọng số có thể ảnh hưởng đến khả năng hội tụ của mô hình và thậm chí đến chất lượng của mô hình sau khi huấn luyện. Nếu các trọng số có các giá trị ban đầu lớn:

- Các hàm sigmoid sẽ đạt trạng thái bão hòa sớm
- Hệ thống sẽ tắc ở một điểm yên ngựa (saddle stationary points)

Thông thường, các trọng số được khởi tạo với các giá trị nhỏ ngẫu nhiên với các phương pháp khởi tạo ngẫu nhiên như Gaussian (phân phối Gaussian), Uniform (phân phối đều).



Hình 50: Hình minh họa quá trình cập nhật tham số dựa trên momentum. Quỹ đạo bên trái không sử dụng momentum. Quỹ đạo bên phải có sử dụng momentum.

Tốc độ học (learning rate) η : Tốc độ học (learning rate) là một tham số quan trọng trong quá trình huấn luyện mô hình học máy. Nó định rõ mức độ ảnh hưởng của mỗi bước cập nhật trọng số đến mô hình trong quá trình tối ưu hóa hàm mất mát. Tốc độ học ảnh hưởng đến tốc độ hội tụ của mô hình, và việc chọn giá trị phù hợp cho tốc độ học là một phần quan trọng của điều chỉnh mô hình. Một số quan sát ảnh hưởng của quá trình học theo tốc độ học như sau:

- Một giá trị η lớn có thể đẩy nhanh sự hội tụ của quá trình học, nhưng có thể làm cho hệ thống bỏ qua điểm tối ưu toàn cục hoặc hội tụ vào điểm không tốt (saddle points)
- Một giá trị η nhỏ có thể làm cho quá trình học kéo dài rất lâu

Tốc độ học thường được chọn theo thực nghiệm (experimentally) đối với mỗi bài toán. Các giá trị tốt của tốc độ học ở lúc bắt đầu (quá trình học) có thể không tốt ở một thời điểm sau đấy. Sử dụng một tốc độ học thích nghi (động) hoặc lập lịch, giảm tốc độ học theo thời gian.

Momentum: Phương pháp Gradient descent có thể rất chậm nếu η nhỏ, và có thể dao động mạnh nếu η quá lớn. Để giảm mức độ dao động, cần đưa vào một thành phần

momentum (Hình 50):

$$\Delta w(t) = -\eta \nabla E^{(t)} + \alpha \Delta w^{(t-1)}$$

trong đó $\alpha (\in [0, 1])$ là một tham số momentum (thường lấy $=0.9$). Dựa trên các kinh nghiệm, ta nên chọn các giá trị hợp lý cho tốc độ học và momentum thỏa mãn $(\eta + \alpha) \gtrsim 1$ trong đó $\alpha > \eta$ để tránh dao động.

Số lượng các nơ-ron ở tầng ẩn: Kích thước (số nơ-ron) của tầng ẩn là một câu hỏi quan trọng đối với việc áp dụng các mạng nơ-ron lan truyền tiến nhiều tầng để giải quyết các bài toán thực tế. Trong thực tế, rất khó để xác định chính xác số lượng các nơ-ron cần thiết để đạt được một độ chính xác mong muốn của hệ thống. Kích thước của tầng ẩn thường được xác định qua thí nghiệm (experiment/trial and test).

CHƯƠNG 4. KHAI PHÁ DỮ LIỆU

4.1 Khai phá dữ liệu

4.1.1 Tại sao cần Khai phá dữ liệu?

Tập đoàn Dữ liệu quốc tế (IDC), một đơn vị nghiên cứu thông tin thị trường, đã dự đoán rằng sau mỗi năm, “vũ trụ số” (digital universe) – dung lượng dữ liệu được tạo ra và sao chép sẽ đạt 180 zettabyte (180 ngàn tỷ tỷ byte) vào năm 2025 và dung lượng dữ liệu sẽ tăng gấp đôi sau mỗi 2 năm trong thập kỷ tới. Theo tờ báo Wall Street Journal, các tập đoàn lớn như Amazon, Alphabet và Microsoft đã đầu tư 32 tỷ USD cho các “nhà máy lọc dữ liệu” với dự đoán rằng, dữ liệu sẽ ngày một tăng mạnh và nhanh hơn trong tương lai.

Thế hệ dữ liệu hiện nay cũng có chất lượng cao hơn do chúng nằm trong một thế giới kết nối. Theo nghiên cứu của PwC (PricewaterhouseCoopers), hiện có khoảng 8,4 tỷ thiết bị di động thông minh, cảm biến, thiết bị truyền động, xe cộ, máy ghi hình và máy bay không người lái được kết nối với nhau trên toàn cầu. Trung bình mỗi ngày các doanh nghiệp có thể tạo ra hàng triệu mảnh dữ liệu không cấu trúc (unstructured). Các dữ liệu này có thể ở dưới dạng thư điện tử, bài viết trên mạng xã hội hay là những cuộc trò chuyện về khách hàng, hành vi của công chúng và những xu hướng phát triển mới. Lượng dữ liệu đang thực sự bùng nổ hơn bao giờ hết và dữ liệu đang ngày càng dễ tiếp cận. Việc ứng dụng của dữ liệu cũng đa dạng hơn rất nhiều so với những thế kỷ trước nhưng chỉ có 0,5% toàn bộ dữ liệu trên thế giới được xử lý và sử dụng mỗi năm, theo Technology Review. Có thể nói chúng ta bị tràn ngập trong dữ liệu nhưng lại thiếu (cần) tri thức. Khai phá dữ liệu liên quan đến việc trích xuất thông tin từ một lượng lớn dữ liệu. Khai phá dữ liệu là một kỹ thuật khám phá các loại mẫu khác nhau được ẩn chứa trong tập dữ liệu và là các tri thức mới và hữu ích. Vì vậy, ứng dụng của khai phá dữ liệu để thấu hiểu thị trường, khách hàng và bản thân doanh nghiệp đóng một vai trò rất quan trọng, đặc biệt là trong khả năng quản trị doanh nghiệp, dự báo và lên kế hoạch kinh doanh. Nó giúp quản lý chi phí, giảm thiểu rủi ro và là chìa khóa cho sự tăng trưởng của doanh nghiệp và tạo ra những điểm khác biệt và lợi thế cạnh tranh nổi bật cho doanh nghiệp trong tương lai.

4.1.2 Dữ liệu, Thông tin, và Tri thức

Dữ liệu, Thông tin và Tri thức là ba khái niệm cơ bản của thời đại chúng ta - thời đại số. Nói một cách căn bản, dữ liệu là những tín hiệu thô con người quan sát, thu thập hoặc đo đạc được về các sự kiện, sự vật đã và đang diễn ra; thông tin là ý nghĩa của dữ liệu có được nhờ xử lý dữ liệu; tri thức là những hiểu biết ở mức cao hơn, có từ tổng hợp kinh nghiệm và liên kết thông tin liên quan dẫn đến quyết định hành động.

Định nghĩa của dữ liệu

Dữ liệu là tín hiệu (signals) thu được do quan sát, đo đạc, thu thập... từ các đối tượng. Cụ thể, dữ liệu là giá trị (values) của các thuộc tính (features) của các đối tượng, được biểu diễn bằng dãy các bits, các con số hay ký hiệu...

Một số các định nghĩa khác về dữ liệu:

- Dữ liệu chỉ là dữ kiện thô [Long and Long, 2002].
- Dữ liệu... là các luồng dữ kiện thô biểu diễn các sự kiện... trước khi chúng được sắp xếp thành một dạng mà mọi người có thể hiểu và sử dụng [Laudon and Laudon, 1997].
- Dữ liệu bao gồm các dữ kiện [Hayes, 1993], các ký hiệu được ghi lại [McNurlin and Sprague, 2005].

Định nghĩa của Thông tin

Thông tin là dữ liệu có ý nghĩa (data equipped with meaning), thu được khi xử lý dữ liệu để lọc bỏ đi các phần dư thừa, tìm ra phần cốt lõi đặc trưng cho dữ liệu.

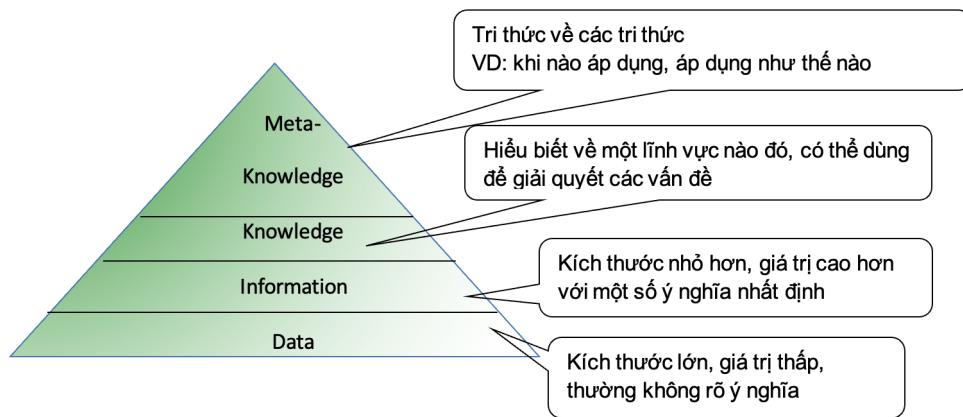
Một số các định nghĩa khác về thông tin:

- Dữ liệu đã được đưa về một dạng có ý nghĩa và hữu ích đối với con người (Laudon and Laudon, 1998)
- Dữ liệu đã được thu thập và xử lý thành một dạng có ý nghĩa. Đơn giản, thông tin là ý nghĩa mà chúng ta cung cấp cho các dữ kiện tích lũy (Long and Long, 1998)

Định nghĩa của Tri thức

Tri thức là thông tin tích hợp, như quan hệ giữa các sự kiện, giữa các thông tin... thu được qua quá trình nhận thức, phát hiện hoặc học tập.

Một số các định nghĩa khác về tri thức:



Hình 51: Minh họa kim tự tháp tri thức.

- Kết quả của sự hiểu biết thông tin [Hayes, 1993].
- Kết quả của việc ngầm thông tin [Hayes, 1993].
- Thông tin có định hướng hoặc ý định, nó giúp hỗ trợ cho một quyết định hoặc một hành động [Zachman, 1987].

Kim tự tháp tri thức

Con người thu dữ liệu bằng nhiều cách và dữ liệu (tín hiệu thô) thường có nhiều dạng, như chữ số, văn bản, hình ảnh, âm thanh, đồ thị, video... Dữ liệu có thể có được qua khảo sát và điều tra; qua quan sát, ghi chép và thu thập. Dữ liệu cũng có thể có được qua đo đạc Điều cốt lõi là thông tin và tri thức-đặc biệt là các tri thức không tường minh (tacit knowledge) - cần cho các quyết định và hoạt động hằng ngày của con người đều ẩn chứa trong dữ liệu thu thập được. Trong kỷ nguyên số, khi tri thức là sức mạnh và trong cuộc cách mạng công nghiệp lần thứ tư, khi "thông minh hóa" là xu thế, chính dữ liệu là nguồn tài nguyên vô giá để giúp ta thấu hiểu về các hoạt động, để tìm ra thông tin, tìm ra tri thức, để mỗi hoạt động đều có thể là hoạt động thông minh.

4.1.3 Các bài toán trong khai phá dữ liệu

Các bài toán trong Khai phá dữ liệu được có thể chia làm 2 nhóm chính:

- Tác vụ mô tả: có nhiệm vụ mô tả về các tính chất hoặc phân tích các đặc tính chung của dữ liệu để thu được thông tin mới hoặc cho một mục đích hữu ích nào đó. Nhóm bài toán thuộc tác vụ này gồm: phân cụm(Clustering), tổng hợp hóa (Summerization), khai phá luật kết hợp (Association Rules), ...

- Tác vụ tiên đoán: có nhiệm vụ đưa ra các dự đoán về những sự kiện chưa biết trong tương lai và tìm ra lý do đằng sau những sự kiện đó. Nhóm bài toán thuộc tác vụ này gồm: phân lớp (Classification), hồi quy (Regression), phát hiện ngoại lai (Outlier Detection), ...

Phân lớp

Quá trình phân lớp dữ liệu thường gồm 2 bước: xây dựng mô hình và sử dụng mô hình để phân lớp dữ liệu.

- Bước 1: một mô hình sẽ được xây dựng dựa trên việc phân tích các mẫu dữ liệu sẵn có với mỗi mẫu tương ứng với một lớp. Tập các mẫu dữ liệu này còn được gọi là tập dữ liệu huấn luyện (training data set). Các nhãn lớp của tập dữ liệu huấn luyện đều phải được xác định trước khi xây dựng mô hình. Vì vậy, phương pháp này còn được gọi là học có giám sát (supervised learning) khác với phân nhóm dữ liệu là học không có giám sát (unsupervised learning).
- Bước 2: sử dụng mô hình để phân lớp dữ liệu. Trước hết chúng ta phải tính độ chính xác của mô hình. Nếu độ chính xác là chấp nhận được, mô hình sẽ được sử dụng để dự đoán nhãn lớp cho các mẫu dữ liệu khác trong tương lai.

Hồi quy (Regression)

Hồi quy là việc học một hàm ánh xạ từ một mẫu dữ liệu thành một biến dự đoán có giá trị thực. Nhiệm vụ của hồi quy tương tự như phân lớp, điểm khác nhau là thuộc tính để dự báo là liên tục chứ không rời rạc. Việc dự báo các giá trị số thường được làm bởi các phương pháp thống kê cổ điển chẳng hạn như hồi quy tuyến tính. Tuy nhiên, phương pháp mô hình hóa cũng có thể được sử dụng như cây quyết định.

Phát hiện ngoại lai

Ngoại lai là một đối tượng mà có khác biệt rất lớn với các đối tượng thông thường, tưởng chừng như nó được sinh ra bởi một cơ chế hoàn toàn khác. Ví dụ:

- Một thanh toán tín dụng bất thường
- Một tấn công mạng
- Một giá cổ phiếu bất thường

Các điểm ngoại lai thường thú vị. Nó vi phạm các cơ chế sinh dữ liệu thông thường và khác với nhiễu. Nhiệm vụ của chúng ta là phát hiện các ngoại lai này (outlier detection, anomaly detection).

Luật kết hợp

Luật kết hợp là dạng luật biểu diễn tri thức ở dạng tương đối đơn giản. Mục tiêu của phương pháp này là phát hiện và đưa ra các mối liên hệ giữa các giá trị dữ liệu trong cơ sở dữ liệu (CSDL) giao dịch. Mẫu đầu ra của giải thuật khai phá này là tập luật kết hợp tìm được. Tuy luật kết hợp là một dạng luật khá đơn giản nhưng lại mang rất nhiều ý nghĩa. Thông tin mà dạng luật này đem lại rất có lợi trong các hệ hỗ trợ ra quyết định. Tìm kiếm được những luật kết hợp đặc trưng và mang nhiều thông tin từ CSDL tác nghiệp là một trong những hướng tiếp cận chính của lĩnh vực khai phá dữ liệu.

Phân nhóm/phân đoạn (Clustering / Segmentation)

Mục tiêu chính của việc phân nhóm dữ liệu là nhóm các đối tượng tương tự nhau trong tập dữ liệu vào các nhóm sao cho mức độ tương tự giữa các đối tượng trong cùng một nhóm là lớn nhất và mức độ tương tự giữa các đối tượng nằm trong các nhóm khác nhau là nhỏ nhất. Các nhóm có thể tách nhau hoặc phân cấp gối lên nhau và số lượng các nhóm là chưa biết trước. Một đối tượng có thể vừa thuộc nhóm này, nhưng cũng có thể vừa thuộc nhóm khác. Không giống như phân lớp dữ liệu, phân nhóm dữ liệu không đòi hỏi phải định nghĩa trước các mẫu dữ liệu huấn luyện. Vì thế, có thể coi phân nhóm dữ liệu là một cách học bồi quan sát (learning by observation), trong khi phân lớp dữ liệu là học bồi ví dụ (learning by example). Trong phương pháp này bạn sẽ không thể biết kết quả các nhóm thu được sẽ như thế nào khi bắt đầu quá trình. Vì vậy, thông thường cần có một chuyên gia về lĩnh vực đó để đánh giá các nhóm thu được. Phân nhóm còn được gọi là học không có giám sát (unsupervised learning). Phân nhóm dữ liệu được sử dụng nhiều trong các ứng dụng về phân đoạn thị trường, phân đoạn khách hàng, nhận dạng mẫu, phân cụm trang Web, ... Ngoài ra phân nhóm dữ liệu còn có thể được sử dụng như một bước tiền xử lý cho các thuật toán Khai phá dữ liệu khác.

Tổng hợp hóa (Summarization)

Tổng hợp hay tóm tắt dữ liệu liên quan đến các phương pháp tìm kiếm một mô tả ngắn gọn cho tập con dữ liệu. Kỹ thuật mô tả khái niệm và tổng hợp hóa thường áp dụng trong việc phân tích dữ liệu có tính thăm dò và báo cáo tự động. Nhiệm vụ chính

là sản sinh ra các mô tả đặc trưng cho một lớp. Mô tả loại này là một kiểu tổng hợp, tóm tắt các đặc tính chung của tất cả hay hầu hết các mẫu của một lớp. Các mô tả đặc trưng thể hiện theo luật có dạng sau: “Nếu một mẫu thuộc về lớp đã chỉ trong tiền đề thì mẫu đó có tất cả các thuộc tính đã nêu trong kết luận”.

Mô hình hóa sự phụ thuộc (dependency modeling)

Mô hình sự phụ thuộc trong dữ liệu là việc tìm kiếm một mô hình mà nó mô tả những phụ thuộc có ý nghĩa giữa các biến hay các thuộc tính theo hai mức:

- Mức cấu trúc của mô hình mô tả các biến cục bộ phụ thuộc vào nhau thường dưới dạng đồ thị.
- Mức định lượng của mô hình mô tả độ mạnh của các phụ thuộc dưới dạng một số.

Những phụ thuộc này thường được biểu thị dưới dạng luật “nếu - thì”, có nghĩa là nếu tiền đề đúng thì kết luận đúng. Về nguyên tắc, cả tiền đề và kết luận đều có thể là sự kết hợp logic của các giá trị thuộc tính. Trên thực tế, tiền đề thường là nhóm các giá trị thuộc tính và kết luận chỉ là một thuộc tính. Hơn nữa, hệ thống có thể phát hiện các luật phân lớp trong đó tất cả các luật cần phải có cùng một thuộc tính phân lớp do người dùng chỉ ra trong kết luận. Quan hệ phụ thuộc cũng có thể biểu diễn dưới dạng mạng tin cậy Bayes. Đó là đồ thị có hướng, không chu trình, trong đó các nút biểu diễn các thuộc tính và trọng số của liên kết thể hiện độ mạnh của phụ thuộc giữa các thuộc tính đó.

4.1.4 Quá trình khám phá tri thức

Quá trình khám phá tri thức từ dữ liệu lớn là một quá trình có sử dụng nhiều phương pháp và công cụ phân tích dữ liệu nhưng vẫn là một quá trình mà trong đó con người là trung tâm. Người sử dụng hệ thống ở đây phải là người có kiến thức cơ bản về lĩnh vực cần phát hiện tri thức để có thể chọn được đúng các tập con dữ liệu, các lớp mẫu phù hợp và đạt tiêu chuẩn quan tâm so với mục đích. Tri thức mà ta nói ở đây là các tri thức rút ra từ các CSDL, thường để phục vụ cho việc giải quyết một loạt nhiệm vụ nhất định trong một lĩnh vực nhất định. Do đó, quá trình phát hiện tri thức cũng mang tính chất hướng nhiệm vụ, không phải là phát hiện mọi tri thức bất kỳ mà là phát hiện tri thức nhằm giải quyết tốt nhiệm vụ đề ra.

Tích hợp dữ liệu (Integration)

Tập hợp dữ liệu là bước đầu tiên trong quá trình khám phá tri thức. Đây là bước tìm kiếm hay khai thác dữ liệu trong một cơ sở dữ liệu, một kho dữ liệu, một cơ sở dữ liệu giao dịch và thậm chí các dữ liệu từ các nguồn ứng dụng Web.

Trích lọc dữ liệu (Selection)

Bước này lựa chọn những dữ liệu phù hợp với nhiệm vụ phân tích mà được trích rút từ các nguồn dữ liệu gom được.

Làm sạch, tiền xử lý và chuẩn bị trước dữ liệu (Data Cleaning, Pre-processing and Preparation)

Bước thứ ba này là một bước rất quan trọng trong quá trình khám phá tri thức. Một số vấn đề thường gặp trong khi gom dữ liệu từ nhiều nguồn không thuần nhất là dữ liệu thường không đầy đủ, không chính xác và không nhất quán. Giai đoạn này sẽ tiến hành điền các dữ liệu thiếu, xử lý những dữ liệu không đúng đắn hay dữ liệu có lỗi và nhiều nói trên để đảm bảo tập dữ liệu thu được có chất lượng. Bởi vậy, đây là một giai đoạn rất quan trọng vì dữ liệu này nếu không được “làm sạch” sẽ gây nên những kết quả khai phá trả về sai lệch nghiêm trọng.

Chuyển đổi dữ liệu (Transformation)

Tiếp theo là giai đoạn chuyển đổi dữ liệu, dữ liệu được chuyển đổi hay được hợp nhất về dạng thích hợp cho việc khai phá.

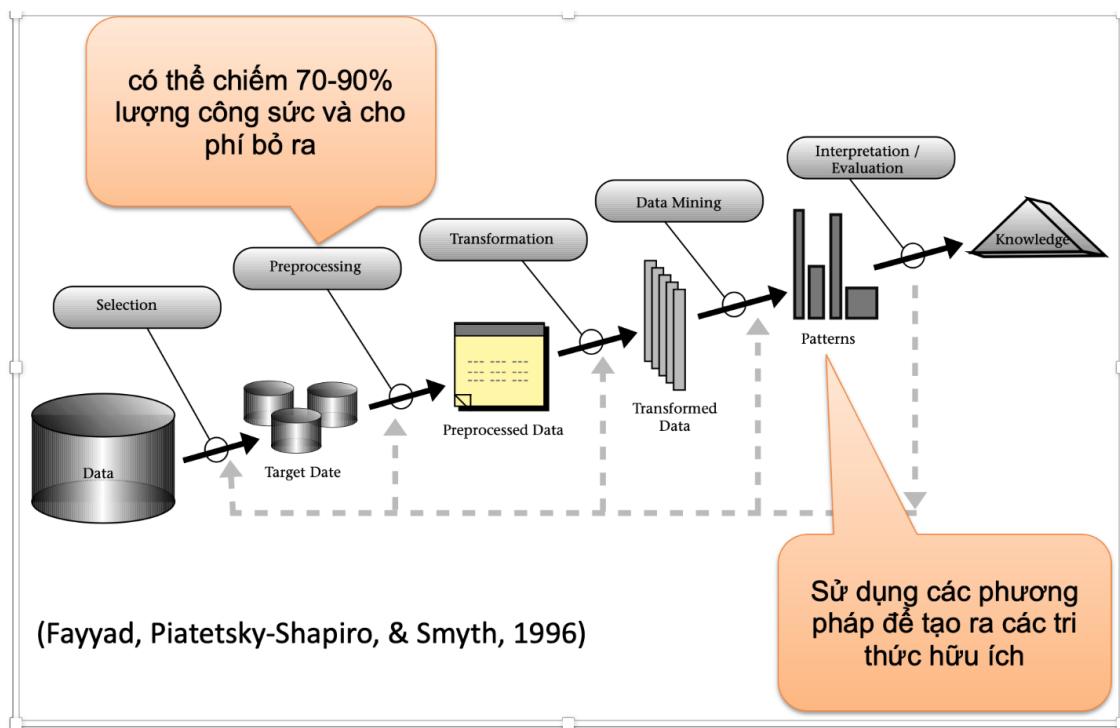
Khai phá dữ liệu (Data Mining) Đây là một giai đoạn quan trọng và cốt yếu. Ở giai đoạn này, nhiều thuật toán khác nhau được sử dụng một cách phù hợp để trích xuất thông tin có ích hoặc các mẫu điển hình trong dữ liệu.

Đánh giá kết quả mẫu (Result evaluation)

Đây là giai đoạn cuối trong quá trình khám phá tri thức. Thông thường, trong số các mẫu dữ liệu được chiết xuất, không phải bất cứ mẫu dữ liệu nào cũng đều hữu ích, đôi khi nó còn bị sai lệch. Vì vậy, cần phải xác định những tiêu chuẩn đánh giá để chiết xuất ra các tri thức cần thiết.

Từ quá trình khám phá tri thức trên, chúng ta thấy được sự khác biệt giữa khám phá tri thức và khai phá dữ liệu. Trong khi khám phá tri thức là nói đến quá trình tổng thể phát hiện tri thức hữu ích từ dữ liệu. Còn khai phá dữ liệu chỉ là một bước trong quá trình khám phá tri thức, các công việc chủ yếu là xác định được bài toán khai phá, tiến

hành lựa chọn phương pháp khai phá dữ liệu phù hợp với dữ liệu có được và tách ra các tri thức cần thiết. Tuy nhiên, giai đoạn tiền xử lý dữ liệu có thể chiếm 70-90% lượng công sức và tiền của bỏ ra trong toàn bộ quá trình khám phá tri thức.



4.1.5 Các loại dữ liệu có thể khai phá

Các loại dữ liệu có thể được khai phá như sau:

- Cơ sở dữ liệu quan hệ (relational database): là những cơ sở dữ liệu (CSDL) được tổ chức theo mô hình quan hệ. Hiện nay, các hệ quản trị CSDL đều hỗ trợ mô hình này như: MS Access, MS SQL Server, Oracle, IBM DB2,...
- Cơ sở dữ liệu đa chiều (multidimensional structures, data warehouse, data mart): còn được gọi là kho dữ liệu, trong đó dữ liệu được hợp nhất từ nhiều nguồn khác nhau và chứa những đặc tính lịch sử thông qua thuộc tính thời gian tưởng minh hoặc ngầm định.
- Cơ sở dữ liệu giao dịch (transaction database): là loại dữ liệu được sử dụng nhiều trong siêu thị, thương mại, ngân hàng,...
- Cơ sở dữ liệu cấp cao (Multimedia/Graph database): là loại dữ liệu có nhiều trên mạng, bao gồm các loại như âm thanh, hình ảnh, video, văn bản và nhiều kiểu dữ liệu định dạng khác.

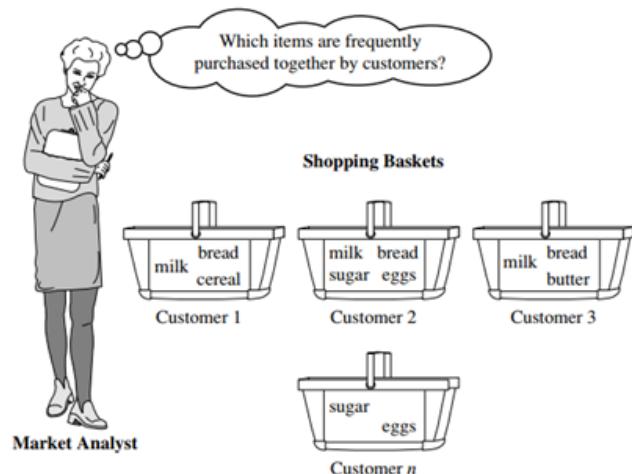
4.1.6 Nhũng thách thức trong khai phá dữ liệu

- Kích thước lớn của các tập dữ liệu cần xử lý: Các tập dữ liệu trong Khai phá dữ liệu thường có kích thước cực kỳ lớn. Trong thực tế, kích thước của các tập dữ liệu trong Khai phá dữ liệu thường ở mức tera-byte. Với kích thước như thế, thời gian xử lý thường cực kỳ dài.Thêm vào đó, các giải thuật học truyền thống thường yêu cầu tập dữ liệu được tải toàn bộ lên bộ nhớ để xử lý. Mặc dù kích thước bộ nhớ trong của máy tính đã gia tăng đáng kể trong thời gian gần đây, việc gia tăng này cũng không thể đáp ứng kịp với việc tăng kích thước dữ liệu. Vì vậy, việc vận dụng các kỹ thuật xác suất, lấy mẫu, song song và tăng dần vào các giải thuật để tạo ra các phiên bản phù hợp với yêu cầu của Khai phá dữ liệu trở nên ngày càng quan trọng.
- Các kỹ thuật trong Khai phá dữ liệu là hướng tác vụ và hướng dữ liệu. Thay vì tập trung vào xử lý tri thức dạng kí hiệu và khái niệm như trong máy học, mọi phát triển trong Khai phá dữ liệu thì kết chặt vào các ứng dụng thực tế và đặc tính dữ liệu cụ thể trong các ứng dụng đó. Ví dụ, luật kết hợp là kỹ thuật Khai phá dữ liệu nhằm tìm kiếm những mối liên kết giữa các món hàng trong các hóa đơn ở siêu thị. Giải thuật học trong kỹ thuật này được phát triển dựa trên đặc tính về dữ liệu rất đặc thù là ở dạng nhị phân.
- Mức độ nhiễu cao trong dữ liệu của Khai phá dữ liệu cùng với những mối tương tác ẩn chứa bên trong dữ liệu có thể rất lớn: Tiêu chuẩn mạnh mẽ của giải thuật đối với nhiễu và tương tác ẩn trở nên quan trọng hơn trong khi các tiêu chuẩn khác phần nào có thể giảm bớt.
- Các ứng dụng thực tế thường có số chiều rất cao. Vì vậy, dữ liệu dù thu thập được lớn đến đâu thì cũng là quá nhỏ so với không gian của chúng.

4.2 Khai phá luật kết hợp

4.2.1 Bài toán phân tích giỏ hàng

Bài toán phân tích giỏ mua hàng (Market basket analysis) thực hiện phân tích thói quen mua hàng của khách hàng bằng cách tìm ra mối liên hệ giữa các mặt hàng khác



Hình 52: Bài toán phân tích giỏ hàng (Liu [2011])

nhau mà khách hàng đặt vào “giỏ mua hàng” của họ (Hình 52). Việc phát hiện ra những mối liên hệ này có thể giúp các nhà bán lẻ phát triển các chiến lược tiếp thị bằng cách hiểu rõ hơn những mặt hàng nào được khách hàng thường xuyên mua cùng nhau.

Ví dụ: nếu khách hàng đang mua sữa thì khả năng họ cũng mua bánh mì trong cùng một lần đi siêu thị. Ở đây bài toán giỏ hàng tìm kiếm thông tin các mặt hàng được mua cùng nhau (“Which items are bought together frequently”). Thông tin này có thể làm tăng doanh số bán hàng bằng cách giúp các nhà bán lẻ có kế hoạch hợp lý cho không gian trưng bày các mặt hàng.

Trong một ngữ cảnh khác, ví dụ bán hàng trực tuyến, việc gợi ý các mặt hàng một cách phù hợp là đặc biệt quan trọng do khách hàng không có cái nhìn toàn cảnh tại cửa hàng thực tế. Ví dụ: sau khi quyết định chọn một chiếc máy tính, khách hàng có thể quan tâm đến các thiết bị ngoại vi hoặc phần mềm tiện ích. Bài toán phân tích giỏ hàng tìm kiếm các mặt hàng khách hàng có thể quan tâm theo luật “nếu khách hàng mua mặt hàng A thì có khả năng sẽ mua mặt hàng B” (Which items would customers also be interested in purchasing along with this one?).

Phân tích giỏ hàng thị trường cũng có thể giúp các nhà bán lẻ lên kế hoạch bán những mặt hàng nào với giá giảm. Nếu khách hàng có xu hướng mua máy tính và máy in cùng nhau thì việc giảm giá máy in có thể khuyến khích việc bán máy in cũng như máy tính.

4.2.2 Tập thường xuyên và luật kết hợp

Các mẫu thường xuyên (Frequent patterns) là các mẫu xuất hiện thường xuyên trong một tập dữ liệu. Việc tìm kiếm các mẫu thường xuyên đóng một vai trò quan trọng trong việc khai thác các liên kết, mối tương quan và nhiều mối quan hệ thú vị khác giữa các dữ liệu. Hơn nữa, nó giúp phân loại dữ liệu, phân cụm và các tác vụ khai phá dữ liệu khác. Do đó, tìm kiếm mẫu thường xuyên đã trở thành một nhiệm vụ khai phá dữ liệu quan trọng và là một trong các chủ đề trọng tâm trong nghiên cứu khai phá dữ liệu.

Ký hiệu:

- $I = \{i_1, i_2, \dots, i_m\}$ bao gồm m phần tử riêng biệt được gọi là tập các mục (ví dụ, tương ứng với các mặt hàng trong siêu thị).
- Cơ sở dữ liệu giỏ hàng (basket market database) D bao gồm các giỏ hàng hay các giao dịch của từng khách hàng trong một lần mua hàng.
- Mỗi giỏ hàng (basket) B là một tập hợp các phần tử (các mặt hàng trong một giao dịch). Tức là $B \subseteq I$.

Định nghĩa Tập mục:

- Tập mục (itemset) X là tập các mục trong I (tức là $X \subseteq I$). Một tập mục X được chứa trong giỏ hàng B khi và chỉ khi B chứa mọi mục của X (tức là $X \subseteq T$).
- Một tập k mục (k – itemset) là một tập mục gồm k mục.

Độ đo đối với tập mục:

- Độ đo support của một tập mục X là tỷ lệ phần trăm, hay xác suất một giỏ hàng trong cơ sở dữ liệu chứa tập mục X : $supp(X) = P(X)$

Định nghĩa Tập mục thường xuyên:

- Một tập mục X được gọi là tập mục thường xuyên nếu độ đo support của X thỏa mãn một ngưỡng cho trước: $supp(X) \leq minSupp$

Định nghĩa Luật kết hợp:

- Luật kết hợp là một mệnh đề kéo theo có dạng $R : X \Rightarrow Y$, trong đó X và Y là các tập mục không giao nhau: $X \cup Y = \Phi$.

ID	Basket
1	Bread, Coke, Milk
2	Beer, Coke
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Bread, Diaper, Milk

Hình 53: Ví dụ cơ sở dữ liệu giỏ hàng

Độ đo đối với luật kết hợp:

- Confidence của luật kết hợp R $conf(R)$ là tỷ lệ phần trăm (hay xác suất) các giỏ hàng trong D chứa Y trong số các giỏ hàng chứa X :

$$conf(R) = \frac{P[X \wedge Y]}{P[X]} \quad (102)$$

- Support của luật kết hợp R $supp(R)$ là tỷ lệ phần trăm (hay xác suất) các giỏ hàng trong D chứa cả X và Y so với tổng số các giỏ hàng trong D .

$$supp(R) = P[X \wedge Y] \quad (103)$$

- Tập mục X được gọi là tiền đề (antecedence) của luật kết hợp R và tập mục Y được gọi là hệ quả (consequence) của luật kết hợp R .

Ví dụ tập mục và tập mục thường xuyên:

- Cơ sở dữ liệu D gồm 5 giỏ hàng (Hình vẽ 53)
- Một số tập mục và support của chúng liệt kê ở Hình vẽ 54.
- Trường hợp ngưỡng tối thiểu là $minSupp = 50\%$, ta có các tập mục thường xuyên là $\{Milk\}$, $\{Bread\}$ và $\{Milk, Bread\}$

Ví dụ luật kết hợp và độ đo:

- Một số luật kết hợp (từ cơ sở dữ liệu Hình 53)
 - $R1 : Milk \Rightarrow Bread$
 - $R2 : Bread \Rightarrow Milk$

Itemsets	Count	Support
{Milk}	4	80%
{Bread}	3	60%
{Milk, Bread}	3	60%
{Milk, Bread, Coke}	1	20%

Hình 54: Ví dụ tập mục, tập mục thường xuyên

- $R3 : Milk, Bread \Rightarrow Coke$
- Độ đo support
 - $Supp(R1) = Supp(R2) = 60\%$
 - $Supp(R3) = 20\%$
- Độ đo confidence
 - $Conf(R1) = 75\%$
 - $Conf(R2) = 100\%$
 - $Conf(R3) = 33\%$

4.2.3 Khai phá luật kết hợp

Các độ đo support và confidence là hai thước đo mức độ thú vị của luật kết hợp. Mức support 2% của luật kết hợp có nghĩa là 2% trong số tất cả các giao dịch được phân tích cho thấy máy tính và phần mềm chống vi-rút được mua cùng nhau. Confidence 60% có nghĩa là 60% khách hàng mua máy tính cũng mua phần mềm. Support thể hiện tần suất của luật kết hợp trong cơ sở dữ liệu. Luật kết hợp nếu có support đủ lớn, nó sẽ xuất hiện thường xuyên trong các giỏ hàng (trường hợp cơ sở dữ liệu giỏ hàng), và việc xem xét các luật như vậy sẽ có ý nghĩa hay hữu ích đối với việc kinh doanh. Mặt khác, nếu luật kết hợp có confidence đủ lớn phản ánh tính chắc chắn của mệnh đề “X kéo theo Y” trong các giỏ hàng. Người bán hàng có thể bố trí sản phẩm phần mềm gần với máy tính trong không gian trưng bày của siêu thị hay trên trang Web và thu được tỷ lệ thành công là 60% đối với các khách hàng mua máy tính.

Về mặt toán học, khai phá luật kết hợp là tìm ra tất cả các luật kết hợp R sao cho support và confidence của R lớn hơn hoặc bằng các ngưỡng được xác định trước tương

ứng minSupp và minConf như sau:

$$supp(R) \geq minSupp \quad (104)$$

$$conf(R) \geq minConf \quad (105)$$

Các luật kết hợp thỏa mãn điều kiện 104 và 105 được gọi là các luật mạnh [HK01]. Nói cách khác, khai thác luật kết hợp có nghĩa là tìm ra tất cả các luật mạnh từ một cơ sở dữ liệu nhất định.

Đặt $Z = X \cup Y$. Ta có $supp(R) = P[X \wedge Y] = P[Z]$. Do đó, luật kết hợp R thỏa mãn điều kiện 104 có nghĩa là độ hỗ trợ của tập mục Z lớn hơn hoặc bằng $minSupp$, hay Z là một tập mục thường xuyên.

Một trong những cách có thể khai thác luật kết hợp là trước tiên tìm kiếm tất cả các tập mục thường xuyên Z , sau đó kiểm tra các luật kết hợp hình thành từ các cặp mục X và Y sao cho $Z = X \wedge Y$ và $X \setminus Y \neq \emptyset$. Vì Z là tập mục thường xuyên nên điều kiện (refeq:support-threshold) về support của R được thỏa mãn. Do đó, quy tắc R là mạnh nếu điều kiện (105) cũng được thỏa mãn. Hầu hết các thuật toán khai thác luật kết hợp đều tuân theo khuôn khổ này để tìm kiếm luật kết hợp. Quy trình gồm hai bước trên được tóm tắt dưới đây:

- Bước 1. Tìm tất cả các tập mục phổ biến Z từ cơ sở dữ liệu D , sao cho: $Supp(Z) \leq minSupp$
- Bước 2. Với mỗi tập mục Z được tìm thấy ở Bước 1, tạo các luật kết hợp $R : X \Rightarrow Y$ sao cho:
 - $X \subseteq Z$
 - $Y = Z \setminus X$
 - $conf(R) \geq minConf$

Bước thứ hai khá đơn giản. Đối với mỗi tập mục Z , X có thể là bất kỳ tập mục con nào của Z . Đối với mỗi X , giá trị hợp lý $conf(R)$ của quy tắc R có thể được tính dựa trên $P[X]$ và $P[X \cap Y] = P[Z]$ như thể hiện trong công thức (104). Nếu $conf(R) \geq minConf$ (điều kiện (105) đúng), thì R là một luật kết hợp mạnh với hệ quả Y được tính là $Y = Z \setminus X$. Lưu ý rằng $(X \subseteq Z) \cap (Y = Z \setminus X)$ tương đương với $(Z = X \cup Y) \wedge (X \setminus Y = \emptyset)$.

Vì vậy, nhiệm vụ chính của khai thác luật kết hợp là khám phá các tập phổ biến. Nói chung đây là một quá trình rất tốn kém. Cho một tập hợp các mục có kích thước m , số lượng các tập hợp con riêng biệt là 2^m . Đôi với số lượng mặt hàng trung bình của một giỏ thị trường điển hình, chẳng hạn lên tới 100, có $2^{100} \approx 10^{30}$ tập hợp con. Rõ ràng, việc truy cập tất cả các tập con có thể có để tìm ra những tập con thường xuyên là rất tốn thời gian.

Một số ứng dụng của khai phá luật kết hợp:

- Market basket analysis. Một trong những ứng dụng nổi tiếng nhất của khai phá luật kết hợp là phân tích giỏ hàng thị trường. Điều này liên quan đến việc phân tích các mặt hàng mà khách hàng mua cùng nhau để hiểu thói quen và sở thích mua hàng của họ.
- Medical diagnosis. Sử dụng trong chẩn đoán y khoa có thể được sử dụng để hỗ trợ bác sĩ chữa bệnh cho bệnh nhân. Việc áp dụng khai phá luật kết hợp giúp xác định xác suất mắc bệnh của một bệnh nào đó.
- Customer Segmentation. Sử dụng để phân khúc khách hàng dựa trên thói quen mua hàng của họ. Ví dụ, nhà bán hàng sử dụng khai phá luật kết hợp để phát hiện ra rằng khách hàng mua một số loại sản phẩm nhất định có nhiều khả năng trẻ hơn. Tương tự, họ có thể biết rằng những khách hàng mua một số kết hợp sản phẩm nhất định có nhiều khả năng sống ở các khu vực địa lý cụ thể hơn.
- Fraud Detection. Sử dụng khai phá luật kết hợp để phát hiện hoạt động gian lận. Ví dụ, một công ty thẻ tín dụng có thể sử dụng khai thác quy tắc kết hợp để xác định các mẫu giao dịch gian lận, chẳng hạn như mua nhiều lần từ cùng một người bán trong một khoảng thời gian ngắn.
- Social network analysis. Sử dụng khai phá luật kết hợp để phân tích mạng xã hội. Ví dụ, phân tích dữ liệu Twitter có thể tiết lộ rằng người dùng tweet về một chủ đề cụ thể cũng có khả năng tweet về các chủ đề liên quan khác, điều này có thể cho biết việc xác định các nhóm hoặc cộng đồng trong mạng.
- Recommendation systems. Sử dụng để đề xuất các mặt hàng mà khách hàng có thể quan tâm dựa trên lịch sử mua hàng hoặc duyệt web trước đây của họ. Ví dụ, dịch

vụ phát nhạc trực tuyến có thể sử dụng khai phá luật kết hợp để giới thiệu nghệ sĩ hoặc album mới cho người dùng dựa trên lịch sử nghe của họ.

4.2.4 Độ đo luật kết hợp

Độ đo support và cắt tỉa dựa trên support

Support count:

- Cho một cơ sở dữ liệu D gồm n giỏ hàng. Support count của X là số lượng các giỏ hàng chứa X:

$$\text{count}(X) = |t|X \subseteq t, t \subseteq D|$$

- Nhắc lại công thức tính support của tập mục: $\text{supp}(X) = P[X]$.

Suy ra quan hệ giữa support count và support: $\text{supp}(X) = P(X) = \text{count}(X)/n$

Tính chất phản đơn điệu (Anti-monotone property) của support:

$$\forall X \subseteq Y : \text{supp}(Y) \leq \text{supp}(X)$$

Chứng minh (sử dụng support count):

- Từ $X \subseteq Y$ suy ra: $\forall t : Y \subseteq t \rightarrow X \subseteq t$

Từ định nghĩa support count, suy ra: $(Y) \leq \text{count}(X)$

Từ công thức support count, ta có: $\text{supp}(Y) \leq \text{supp}(X)$

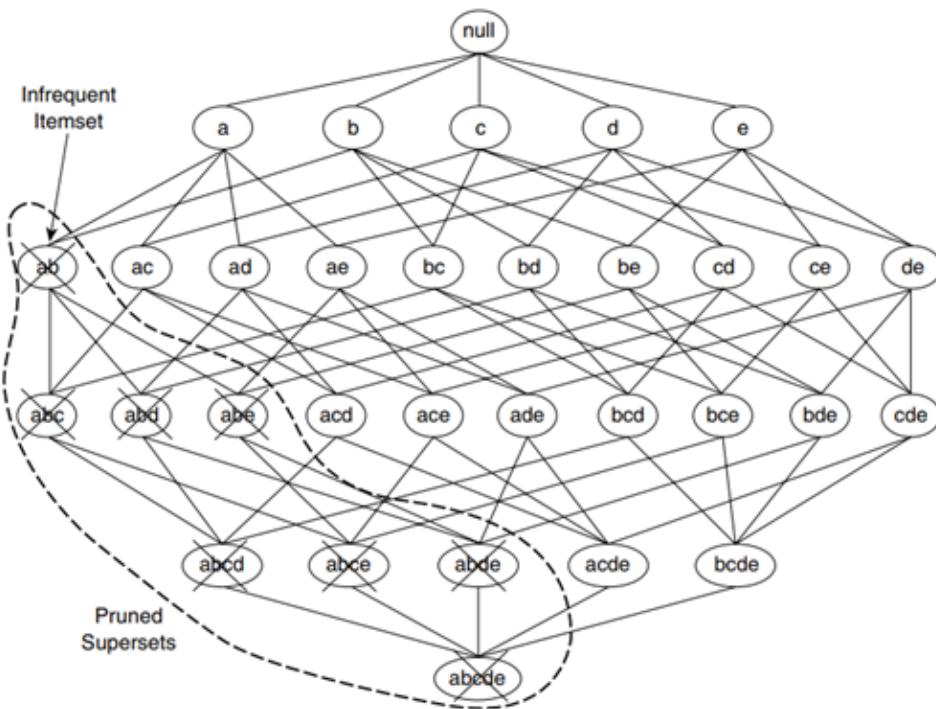
Hệ quả của tính phi đơn điệu của support:

- Nếu X không phải làm tập mục thường xuyên (infrequent itemsets) thì tất cả các tập chứa nó (supersets) đều không phải tập mục thường xuyên!
- Nếu X không phải làm tập mục thường xuyên \rightarrow có thể bỏ qua, không cần xem xét (cắt tỉa - pruning) tất cả các tập mục chứa X (Hình 55).

Độ đo confidence và sinh luật kết hợp dựa trên confidence

Quá trình sinh luật kết hợp đối với mỗi tập mục thường xuyên I:

- Mọi tập mục con A của I sinh ra luật kết hợp $A \Rightarrow I \setminus A$
- Vì I là tập mục thường xuyên nên A cũng là tập mục thường xuyên
- Do đó, chỉ cần kiểm tra $\text{supp}(A \Rightarrow I \setminus A) \geq \text{minSupp}$?



Hình 55: Tính phản đơn điệu để cắt tỉa các tập mục

Vấn đề ở đây là một tập mục thường xuyên lớn (gồm nhiều phần tử) sẽ dẫn đến việc kiểm tra rất nhiều luật kết hợp được sinh ra. Ví dụ, 10-itemset có thể sinh ra 1022 luật kết hợp từ các phần tử của nó.

Tương tự như tính toán với độ đo support ở phần trê, ta có thể tính confidence theo support count: $conf(X \Rightarrow Y) = count(X \cup Y) / count(X)$. Nhắc lại công thức confidence: $conf(X \Rightarrow Y) = P(X \cup Y) / P(X)$.

Tính chất của độ đo confidence đối với luật kết hợp:

Nếu $A \subseteq A'$, thì $conf(A \Rightarrow I \setminus A) \geq conf(A \Rightarrow I \setminus A')$

Chứng minh

- Nếu $A \subseteq A'$, dẫn đến mọi giỏ hàng t , $\forall t : A' \subseteq t \rightarrow A \subseteq t$

Theo công thức support count, ta có: $(A') \leq count(A)$

Theo công thức confidence ở trên:

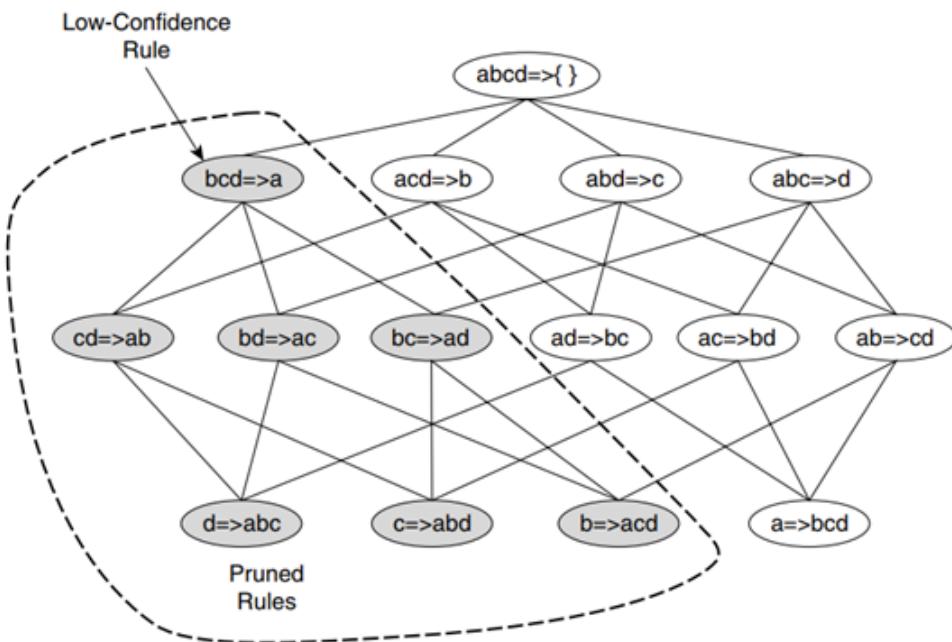
$$conf(A \Rightarrow I \setminus A) = count(I) / count(A)$$

$$conf(A' \Rightarrow I \setminus A') = count(I) / count(A')$$

Suy ra: $conf(A \Rightarrow I \setminus A) \geq conf(A \Rightarrow I \setminus A')$

Ví dụ:

- Do $conf(\{A, B, C\} \Rightarrow \{D\}) \geq conf(\{A, B\} \Rightarrow \{C, D\})$



Hình 56: Sinh rule sử dụng tinh chất độ đo confidence

Do đó, nếu $\{A, B, C\} \Rightarrow \{D\}$ không thỏa mãn ngưỡng $minConf$ thì $\{A, B\} \Rightarrow \{C, D\}$ cũng không thỏa mãn.

Hệ quả:

- Nếu $A \subseteq A'$, và $(A \Rightarrow I \setminus A)$ không thỏa mãn ngưỡng $minConf$ thì $(A' \Rightarrow I \setminus A')$ cũng không thỏa mãn.

Như vậy, nếu $(A \Rightarrow I \setminus A)$ không thỏa mãn ngưỡng $minConf$, thì chúng ta có thể bỏ qua, không cần xem xét (cắt tỉa) các luật kết hợp có dạng $(A' \Rightarrow I \setminus A')$ trong đó $A \subseteq A'$.

Các độ đo khác của luật kết hợp

Khai phá luật kết hợp tìm kiếm các mối quan hệ thú vị giữa các tập mục trong một tập dữ liệu nhất định. Luật kết hợp $X \Rightarrow Y$ có nghĩa là bất cứ khi nào một giao dịch hay giỏ hàng chứa X thì nó có thể chứa Y. Xác suất hay confidence của luật $X \Rightarrow Y$ thể hiện mối quan hệ kéo theo từ X đến Y. Tuy nhiên, một trong những nhược điểm chính của thước đo độ confidence là nó bị ảnh hưởng bởi tần suất của các tập mục. Do đó, tập hợp kết quả thiên về luật kết hợp của các tập mục “rất” thường xuyên. Ví dụ: nếu hai tập mục X và Y có tần số rất cao thì chúng có thể sẽ hình thành một luật kết hợp có giá trị confidence cao ngay cả khi không có mối quan hệ nào giữa chúng.

Độ đo **Interest** được đề xuất trong [Brin et al., 1997b] như một thước đo mối tương quan giữa các mục. Giá trị *interest* của luật kết hợp $R : X \Rightarrow Y$ là tỷ lệ giữa *confidence*

Measure	Formula
Interest	$intr(R) = \frac{P[X \wedge Y]}{P[X] \times P[Y]} = \frac{conf(R)}{P[Y]}$
Conviction	$conv(R) = \frac{P[X] \times P[\neg Y]}{P[X \wedge \neg Y]}$
Reliability	$relb(R) = \frac{P[X \wedge Y]}{P[X]} - P[Y] = conf(R) - P[Y]$
Certainty factor	$CF(R) = \frac{conf(X) - P[Y]}{1 - P[Y]}$

Hình 57: Một số độ đo khác của luật kết hợp

của R đối với *confidence* của hệ quả của R (Y). Giá trị *interest* lớn hơn 1 thể hiện mối tương quan dương giữa hai tập mục trong khi giá trị dưới 1 thể hiện mối tương quan âm. Giá trị *interest* bằng 1 cho thấy các tập mục thực sự không có mối tương quan (có nghĩa X và Y độc lập đối với nhau).

Độ đo ***Conviction*** của luật kết hợp được giới thiệu trong [Brin et al., 1997a] như một phiên bản bất đối xứng của *interest*. Công thức của *conviction* có thể viết lại thành $conv(X \Rightarrow Y) = 1/intr(X \Rightarrow \neg Y)$. Do đó, conviction của luật kết hợp R phản ánh nghịch đảo sự tương quan giữa X và $\neg Y$. Điều đó có nghĩa là conviction của R phản ánh mối tương quan của X và $\neg(\neg Y) = Y$. Nói cách khác, thước đo conviction cũng thể hiện mối tương quan giữa các tập mục, tương tự thước đo *interest*.

Trong [Ahmed et al., 2000], độ đo ***Reliability*** của luật kết hợp R được định nghĩa là sự khác biệt giữa giá trị của thước đo *interest* và giá trị *support* của hệ quả của R . Nó đo lường hiệu quả của thông tin có sẵn về tiền đề đối với xác suất xảy ra hậu quả. Độ tin cậy cao hơn ngụ ý sự liên kết mạnh mẽ hơn “if X then Y ”. Tương quan dương được biểu thị bằng giá trị dương của độ tin cậy trong khi tương quan âm được biểu thị ngược lại.

Hệ số chắc chắn ***Certainty Factor (CF)*** [Shortliffe and Buchanan, 1990] được giới thiệu trong MICYN - một trong những mô hình tốt nhất trong việc phát triển hệ thống chuyên gia dựa trên luật. Hệ số chắc chắn $CF(h|e)$ đo lường niềm tin vào giả thuyết h dựa trên bằng chứng e . Trong cơ sở dữ giờ hàng, hàm ý của luật $X \Rightarrow Y$ thể hiện niềm tin vào giả thuyết h về việc Y xảy ra trong một giờ hàng dựa trên bằng chứng e về điều này giao dịch chứa X . Do đó, với h và e được quan sát trong cơ sở dữ liệu, chúng ta có $P(h|e) = conf(X \Rightarrow Y)$, $P(h) = P[Y]$ và $P[e] = P[X]$.

Theo định nghĩa trên, giá trị của thước đo CF là dương khi $P(h|e) > P(h)$ trong khi nó âm khi $P(h|e) < P(h)$. Trong khai phá luật kết hợp, các mối quan hệ tích cực thường được xem xét. Ví dụ, trong bài toán về giỏ hàng trên thị trường, người ta có thể mong muốn tìm ra mối quan hệ dương “nếu bánh-mì thì sữa” để việc bán bánh mì có thể giúp thúc đẩy doanh số bán sữa. Niềm tin vào giả thuyết “bán sữa” phải tăng dựa trên bằng chứng “bán bánh mì”. Nói cách khác, $P(milk|bread) > P(milk)$ hoặc $CF(milk|bread) > 0$.

4.2.5 Giải thuật Apriori

Vấn đề

Hai bước khai phá luật kết hợp:

- Bước 1. Tìm các tập mục thường xuyên Z từ cơ sở dữ liệu D , sao cho:

$$supp(Z) \geq minSupp$$

- Bước 2. Với mỗi tập Z tìm ở Bước 1, sinh luật $R : X \Rightarrow Y$ sao cho:

$$(X \subseteq Z) \wedge (Y = Z \setminus X) \wedge (conf(R) \geq minConf)$$

Vấn đề tìm tập mục thường xuyên không gian tìm kiếm lớn, tương đương với số lượng các khả năng kết hợp (hay tổ hợp) các phần tử thành tập mục. Khi số lượng phần tử tăng, số lượng này tăng theo hàm số mũ. Với d phần tử, số các khả năng của tập mục bao gồm đến d phần tử là 2^d . Ví dụ đối với kích cỡ trung bình của một cơ sở dữ liệu giỏ hàng từ hàng trăm đến hàng nghìn mặt hàng, việc xem xét tất cả tổ hợp các mặt hàng để xác định tập mục thường xuyên là không khả thi.

Hình 58 minh họa các khả năng kết hợp tập mục từ 5 phần tử (mặt hàng).

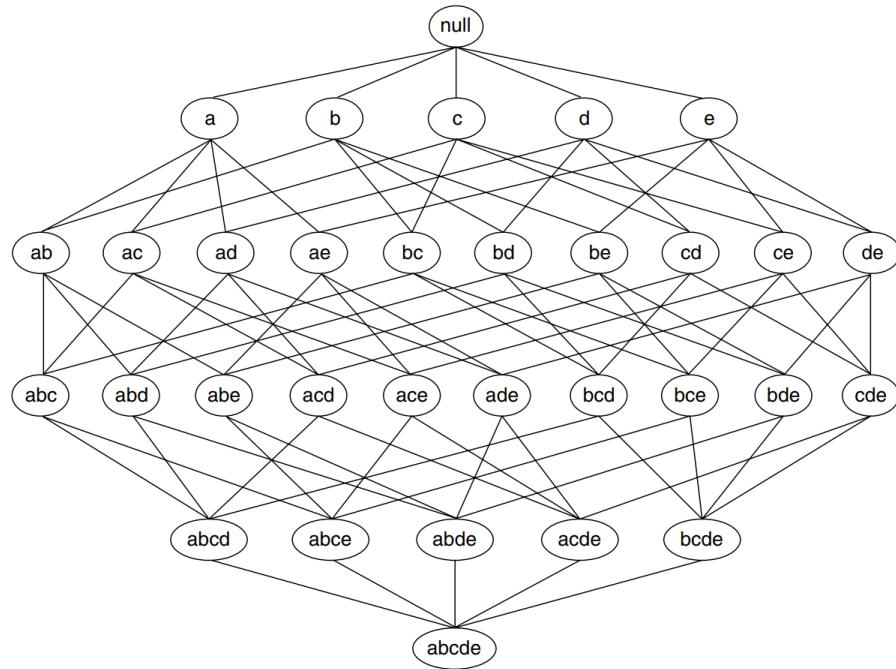
Giải thuật Apriori

Giải thuật Apriori được trình bày ở Hình 59. Giải thuật sử dụng tính chất phản đơn điệu của *support* nêu ở Phần 4.2.4.

Lưu ý:

- C_k chứa các tập mục ứng viên (candidate itemsets) kích thước k .
- L_k chứa các tập mục thường xuyên kích thước k .

Hình 60 trình bày giả mã cho thuật toán Apriori.

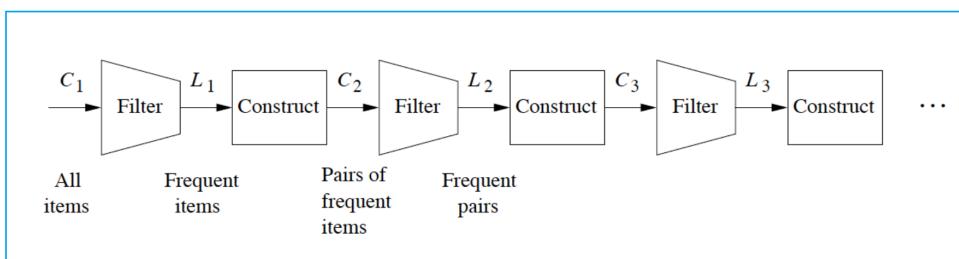


Hình 58: Minh họa các khả năng kết hợp tập mục từ 5 phần tử

Apriori for frequent itemsets

1. $k = 1$, C_1 = all items
 2. While C_k not empty:
 3. Count support of itemsets in C_k and put frequent itemsets into L_k
 4. Use L_k to generate candidates C_{k+1} of size $k+1$
 5. $k = k+1$
 6. $L = \cup L_k$
- C_k = candidate itemsets of size k
 L_k = frequent itemsets of size k

Hình 59: Giải thuật Apriori



Hình 60: Mô tả thuật toán Apriori.

- Tại Bước 1, Apriori đưa tất cả các tập mục một phần tử vào C_1 - tập mục ứng viên 1 phần tử ($1-itemsets$).
- Trong bước thứ k , các tập mục của C_k được "đếm support" (support counting) để loại các tập mục không thường xuyên, thu được các tập mục thường xuyên k phần tử trong L_k . L_k sau đó sử dụng để sinh ra các tập mục ứng viên $k+1$ phần tử C_{k+1} .
- Quá trình kết thúc khi không có ứng viên mới hoặc tập mục thường xuyên mới (sau quá trình sinh ứng viên và đếm support để xác định tập mục thường xuyên).

Các quá trình của giải thuật Apriori cũng được mô tả trực quan ở Hình 63.

Ở mỗi bước của giải thuật Apriori, các tập mục thường xuyên với k phần tử ($k-itemsets$) được xác định thông qua thủ tục đếm support:

- Tại mỗi lần lặp, Apriori quét cơ sở dữ liệu một lần để xác định support của các tập mục ứng viên trong C_k (Hay còn gọi là support counting). Các tập mục trong C_k có support thỏa mãn ngưỡng ($minSupp$) cho trước được đưa các tập mục thường xuyên vào L_k .
- Cụ thể, thuật toán thiết lập bộ đếm bằng 0 với mỗi c trong C_k . Đối với mỗi giỏ trong cơ sở dữ liệu, một tập mục c trong C_k sẽ được tăng bộ đếm lên 1 nếu c xuất hiện trong giỏ hàng. Cuối cùng, tất cả các ứng viên thỏa mãn ngưỡng support tối thiểu tạo thành tập các tập mục thường xuyên, L_k .
- Thủ tục đếm support của C_k và xây dựng tập mục thường xuyên k phần tử L_k được mô tả chi tiết ở Hình 61.

Cũng tại mỗi bước, sau khi tập mục thường xuyên k phần tử L_k được xác định, nó sẽ được sử dụng để sinh các tập mục ứng viên $k+1$ phần tử (còn gọi candidate generation). Chi tiết được mô tả ở Hình 62

2.2. Support counting

```
# Count support of itemsets in Ck and put frequent itemsets into Lk
▪ Ck = all candidates size k
▪ Establish c.count = 0 for each c in Ck, Lk = ∅
▪ For each basket in the database:
    For each c ∈ Ck:
        If c ∈ basket then c.count += 1
    ▪ For each c in Ck :
        If c.count ≥ minSuppCount then Lk ← Lk + c
```

Hình 61: Đếm support (Support counting)

2.3. Candidate generation

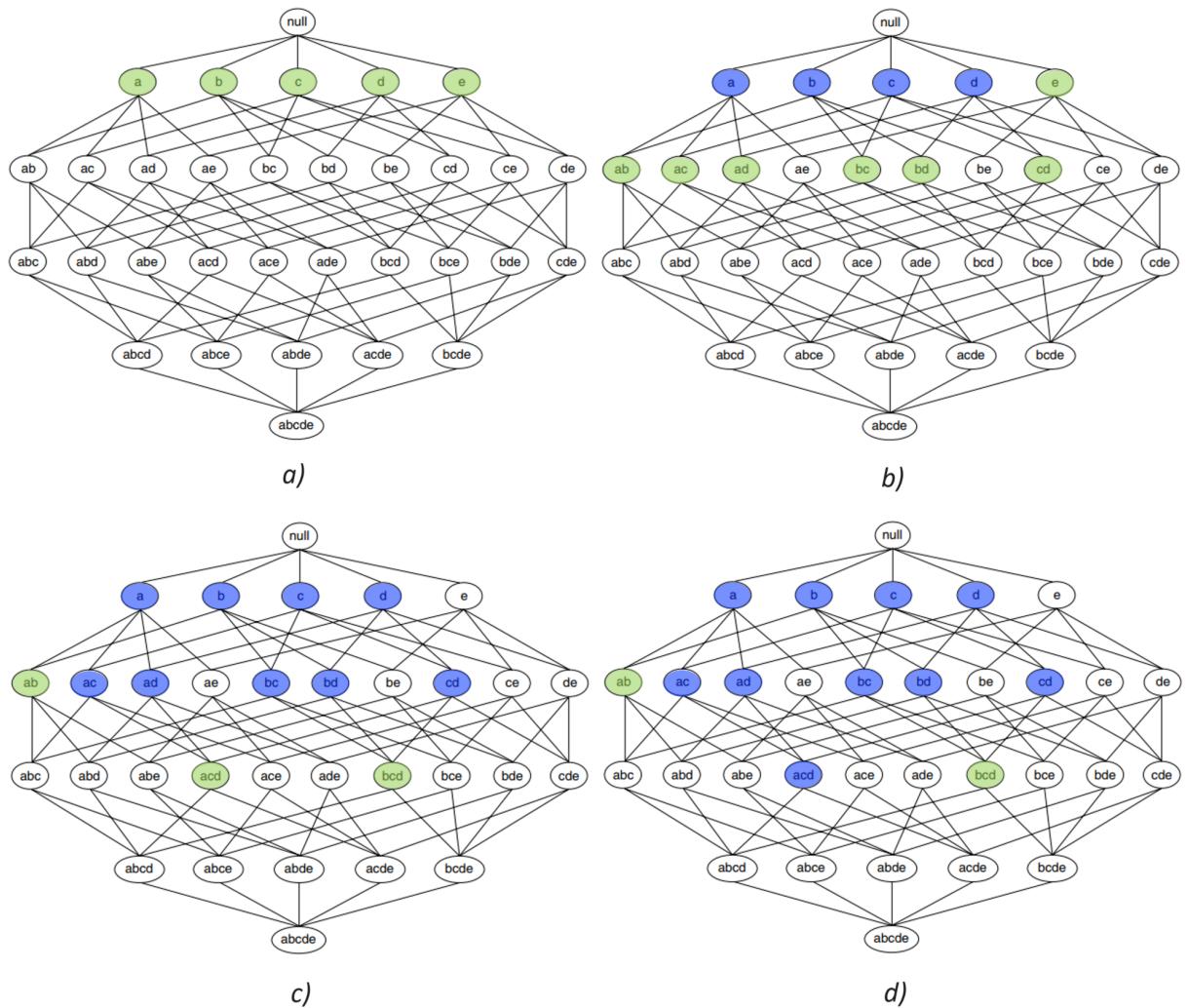
```
# Use Lk to generate candidates Ck+1 of size k+1
▪ If k = 1
    # Take all pairs of frequent items (ci, cj are items)
    • C2 ← {ci, cj} forall ci, cj: (ci ∈ C1) ∧ (cj ∈ C1)
▪ If k > 1
    # Join pairs of itemsets that differ by just one item (C is an (k-1)-itemset)
    • Ck+1 ← C ∪ {ci, cj} forall C, ci, cj: (C ∪ {ci} ∈ Lk) ∧ (C ∪ {cj} ∈ Lk)
    # For candidate, ensure all subsets of size k are frequent (C is an k-itemset)
    • Ck+1.remove(C) forall C: (C ∪ {ci} ∈ Ck+1) ∧ (C ∉ Lk)
```

Hình 62: Sinh ứng viên tập mục thường xuyên (Candidate generation)

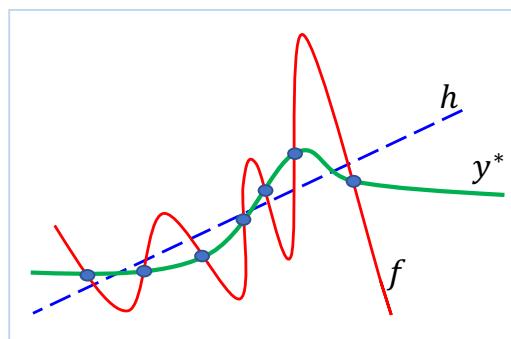
Thủ tục sinh tập mục thực hiện hai loại hành động, đó là nối (join) và cắt tỉa (prune). Trong thành phần nối, L_{k-1} được nối với L_{k-1} để tạo ra các ứng viên tiềm năng. Thành phần Prune sử dụng thuộc tính phản ứng đơn điệu của support để loại bỏ các ứng cử viên có tập con không thường xuyên. Nếu một tập mục k + 1 phần tử trong C_{k+1} mà có tập con k phần tử không thường xuyên (nghĩa là tập con không thuộc L_k), thì nó sẽ bị loại bỏ khỏi C_{k+1}.

Ví dụ về thuật toán Apriori được trình bày ở Hình 63.

- Hình 63(a): k = 1, C₁ bao gồm các tập mục một phần tử.
- Hình 63(b):
 - Support counting: Quét CSDL, đếm support các tập mục trong C₁, đưa các tập mục thường xuyên 1 phần tử vào L₁.
 - Candidate generation: Sử dụng L₁ để sinh tập mục ứng viên 2 phần tử C₂.
- Hình 63(c) k = 2:



Hình 63: Ví dụ Giải thuật Apriori



Hình 64: Độ phức tạp của hàm và khả năng xấp xỉ y^* . Hàm f quá phức tạp và xấp xỉ hoàn hảo tập học (các dấu chấm). Trong khi đó hàm h quá đơn giản.

- Support counting: Quét CSDL, đếm support các tập mục trong C_2 , đưa các tập mục thường xuyên 1 phần tử vào L_2 .
- Candidate generation: Sử dụng L_2 để sinh tập mục ứng viên 2 phần tử C_3 .
- Hình 63(d) $k = 3$:
 - Support counting: Quét CSDL, đếm support các tập mục trong C_3 , đưa các tập mục thường xuyên 1 phần tử vào L_3 .
 - Candidate generation: Sử dụng L_3 để sinh tập mục ứng viên 2 phần tử C_4 (tập rỗng).

4.3 Mở rộng: Hiệu chỉnh

Trong phần này, chúng ta sẽ tìm hiểu kỹ hơn về Hiệu chỉnh (regularization) và ý nghĩa của nó đối với các thuật toán học và các mô hình học máy.

4.3.1 Độ phức tạp mô hình và Quá khớp

Chúng ta nhớ lại rằng việc học sẽ cần tìm một hàm y^* nào đó không biết trước, dựa vào tập dữ liệu huấn luyện \mathcal{D} . Cách làm thường được dùng là chọn một kiểu mô hình \mathcal{H} , rồi thực hiện quá trình học từ tập \mathcal{D} để tìm ra một hàm $f \in \mathcal{H}$ mà có thể xấp xỉ tốt y^* . Khả năng xấp xỉ của hàm f phụ thuộc vào ba nguồn quan trọng là: *kiểu mô hình, thuật toán học, và tập học*.

Hãy xét lớp mô hình \mathcal{H} mà ta đã chọn. Một lớp mô hình đơn giản (ví dụ mô hình tuyến tính) thì sẽ chứa các hàm đơn giản. Khi đó khả năng xấp xỉ của mỗi hàm bên trong có thể rất hạn chế. Hình 64 minh họa hàm h đơn giản đến nỗi không thể xấp xỉ

tốt y^* . Như vậy bất kỳ hàm nào trong \mathcal{H} đều có khả năng xấp xỉ yếu, nghĩa là khả năng tổng quát hoá tệ. Tuy vậy, chúng có độ phức tạp bé. Ngược lại, hàm f trong Hình 64 rất phức tạp và khớp hoàn hảo với tập học. Khi đó \mathcal{H} chứa những hàm có khả năng cao (high capacity). Chúng đều có độ phức tạp cao. Tuy nhiên, nếu việc huấn luyện không tốt thì có thể tìm ra một hàm bị quá khớp.

Hãy xét một thuật toán A. Khi cho trước một tập học D thì thuật toán A sẽ thực hiện quá trình huấn luyện để tìm ra một hàm f nào đó thuộc lớp mô hình đã chọn \mathcal{H} . Nghĩa là ta có thể viết $f = A(\mathcal{H}, D)$. Nếu f là một hàm phức tạp, chẳng hạn như trong Hình 64, thì nó có khả năng xấp xỉ hoàn hảo với nhiều mẫu dữ liệu. Nghĩa là hàm đó có khả năng mạnh. Ngược lại, nếu thuật toán A tìm ra một hàm rất đơn giản, chẳng hạn hàm h trong Hình 64, thì nó có thể xấp xỉ ít mẫu dữ liệu hơn. Nghĩa là đó là hàm có khả năng thấp hơn.

Như vậy độ phức tạp có mối liên hệ chặt chẽ với vấn đề quá khớp và khả năng tổng quát hoá. Mặc dù những lớp mô hình phức tạp thường có khả năng cao, nhưng dễ gặp vấn đề quá khớp nếu thuật toán A thực hiện học không tốt. Đôi khi A có thể tìm ra một hàm quá đơn giản từ lớp mô hình khá mạnh. Như vậy khả năng của một mô hình sau huấn luyện phụ thuộc lớn vào thuật toán học A.

4.3.2 Phân tích lỗi và mâu thuẫn Bias-Variance

Để thấy được rõ hơn về khả năng của một mô hình đã huấn luyện, chúng ta có thể phân tích lỗi phán đoán. Việc phân tích này cũng mang lại một số hiểu biết và có thể giúp ta tìm được cách cải thiện việc huấn luyện. Tiếp theo chúng ta sẽ phân tích lỗi chi tiết hơn.

Xét một bài toán hồi qui mà trong đó hàm cần tìm có dạng $y(\mathbf{x}) = y^*(\mathbf{x}) + \varepsilon$:

- ε đại diện cho nhiễu hoặc lỗi. Chú ý rằng chúng ta không thể tránh sự xuất hiện của nhiễu hoặc lỗi trong tập dữ liệu thu thập được. Nguyên nhân có thể xuất phát từ thiết bị thu thập, từ nguồn tạo ra dữ liệu, từ cách lưu trữ, ...
- Giả sử ε là một nhiễu trắng, tức là nó tuân theo phân phối chuẩn với kỳ vọng 0 và phương sai σ^2 . Trong thực tế, nhiễu này có thể phức tạp hơn. Nhưng ta tạm dùng giả thuyết đó để việc phân tích lỗi đơn giản và dễ hình dung hơn.

Gọi $f_A(\mathbf{x}; \mathbf{D})$ là một hàm thu được sau quá trình huấn luyện, bởi thuật toán A, từ một tập học \mathbf{D} . Tức là $f_A = A(\mathcal{H}, \mathbf{D})$. Ta thường mong muốn hàm này có khả năng tổng quát hoá cao, nghĩa là f_A có thể xấp xỉ tốt hàm y^* tại mọi điểm chứ không chỉ trong tập học. Chú ý rằng hàm này phụ thuộc vào tập học \mathbf{D} . Nếu \mathbf{D} thay đổi thì hàm học được cũng sẽ thay đổi. Các lần thu thập khác nhau có thể tạo ra các tập học khác nhau. Cho nên ta có thể coi $f_A(\mathbf{x}; \mathbf{D})$ là một đại lượng ngẫu nhiên, vì bản thân \mathbf{D} có tính ngẫu nhiên.

Đối với mỗi mẫu dữ liệu \mathbf{x} mới, lỗi phán đoán của hàm f_A là $[y(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D})]^2$. Đại lượng này có thể coi là lỗi phán đoán của thuật toán A đối với mẫu \mathbf{x} , nếu chỉ được học từ \mathbf{D} . Nếu ta dùng tập học \mathbf{D}' khác thì lỗi của A sẽ khác. Do đó ta cần xét đại lượng $err_A(\mathbf{x}) = \mathbb{E}_{\mathbf{D}, \varepsilon} [(y(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D}))^2]$ để thấy lỗi trung bình của A.

Tiếp theo ta hãy phân tích lỗi:

$$err_A(\mathbf{x}) = \mathbb{E}_{\mathbf{D}, \varepsilon} [(y(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D}))^2] = \mathbb{E}_{\mathbf{D}, \varepsilon} [(y^*(\mathbf{x}) + \varepsilon - f_A(\mathbf{x}; \mathbf{D}))^2] \quad (106)$$

$$= \mathbb{E}_{\mathbf{D}, \varepsilon} [(y^*(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D}))^2 + 2(y^*(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D}))\varepsilon + \varepsilon^2] \quad (107)$$

(tách làm 3 kỳ vọng riêng và bỏ vị trí ở giữa vì $\mathbb{E}_\varepsilon[\varepsilon] = 0$)

$$= \mathbb{E}_{\mathbf{D}} [(y^*(\mathbf{x}) - f_A(\mathbf{x}; \mathbf{D}))^2] + \mathbb{E}_\varepsilon [\varepsilon^2] \quad (108)$$

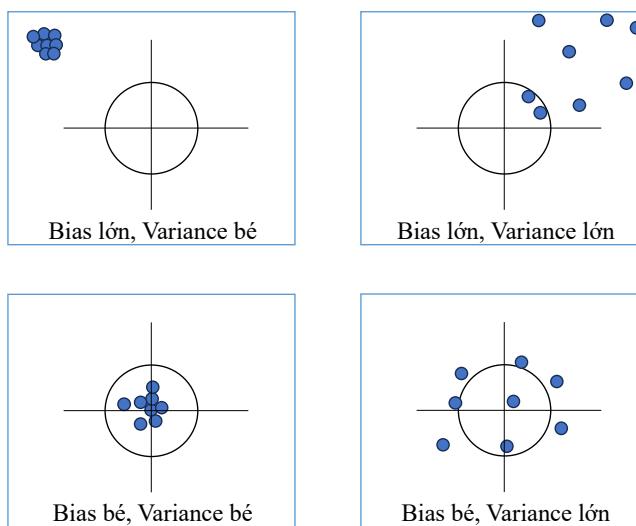
Nhớ lại rằng đối với một biến ngẫu nhiên z thì phương sai $\text{Var}(z) = \mathbb{E}(z^2) - (\mathbb{E}z)^2$, nghĩa là $\mathbb{E}(z^2) = \text{Var}(z) + (\mathbb{E}z)^2$. Áp dụng tính chất này cho biểu thức (108) ta thu được:

$$\begin{aligned} err_A(\mathbf{x}) &= [y^*(\mathbf{x}) - \mathbb{E}_{\mathbf{D}} f_A(\mathbf{x}; \mathbf{D})]^2 + \mathbb{E}_{\mathbf{D}} [f_A(\mathbf{x}; \mathbf{D}) - \mathbb{E}_{\mathbf{D}'} f_A(\mathbf{x}; \mathbf{D}')]^2 + \sigma^2 \\ &= \text{Bias}^2 + \text{Variance} + \sigma^2 \end{aligned} \quad (109)$$

Đây được gọi là **Phân tích Bias-Variance**. Trong đó $\text{Bias} = y^*(\mathbf{x}) - \mathbb{E}_{\mathbf{D}} f_A(\mathbf{x}; \mathbf{D})$ mô tả lỗi trung bình của các phán đoán của thuật toán A, còn đại lượng $\text{Variance} = \mathbb{E}_{\mathbf{D}} [f_A(\mathbf{x}; \mathbf{D}) - \mathbb{E}_{\mathbf{D}'} f_A(\mathbf{x}; \mathbf{D}')]^2$ mô tả mức độ dao động của các phán đoán của A quanh giá trị phán đoán trung bình.

Phân tích Bias-Variance cho ta khá nhiều tính chất và ý nghĩa khác nhau về thuật toán học:

- Do nhiều/lỗi tiềm ẩn trong dữ liệu nên σ^2 không thể tránh được. Điều này gợi ý rằng để A có lỗi phán đoán bé thì ta cần đảm bảo Bias và Variance bé.



Hình 65: Ví dụ về Bias và Variance. Các dấu chấm minh họa các phán đoán của thuật toán A về một mẫu dữ liệu. Hình tròn minh họa vùng phán đoán chấp nhận được.

- Nếu Bias hoặc Variance lớn thì lỗi của A cũng lớn. Để ý rằng Bias lớn nghĩa là thuật toán A phán đoán kém chính xác. Còn Variance lớn gợi ý rằng các phán đoán (cho mỗi mẫu dữ liệu) của A thường rất phân tán. Hình 65 minh họa một số ví dụ về các tính chất này.
- Bias lớn sẽ xảy ra nếu các hàm học được bởi A đều rất đơn giản, nhưng hàm y^* lại phức tạp. Ví dụ thuật toán kNN trong trường hợp dùng số lượng hàng xóm lớn.
- Variance lớn có thể xuất hiện nếu A thường tìm ra các hàm phức tạp. Các hàm đó có thể tạo ra các phán đoán rất khác nhau cho cùng một mẫu dữ liệu. Ví dụ thuật toán kNN trong trường hợp dùng ít hàng xóm.

Bias-Variance trade-off: Phân tích Bias-Variance gợi ý rằng thuật toán A cần đảm bảo Bias bé và Variance bé. Tuy nhiên, để làm được điều này đôi khi rất khó. Hai đại lượng này có thể tạo ra một vấn đề nan giải. Thực vậy nếu thuật toán A cố gắng đi tối ưu Bias thì thường cần tìm ra những hàm đủ phức tạp để xấp xỉ tốt các tập học khác nhau. Lúc đó cần chọn lớp mô hình \mathcal{H} đủ mạnh. Tuy nhiên, Variance lại có xu hướng tăng và quá khớp có thể diễn ra. Tương tự, nếu thuật toán A đi tối ưu Variance thì có thể thường tạo ra những hàm rất đơn giản. Những hàm này lại có thể tạo ra Bias cao, và do đó kém khớp xảy ra.

4.3.3 Nguyên lý cơ bản của Hiệu chỉnh

Ở trên chúng ta đã thấy một vấn đề khó khi tối ưu lỗi phán đoán. Để cân bằng tốt giữa Bias và Variance, một con đường hiệu quả thường được sử dụng trong ML là *Hiệu chỉnh* (Regularization). Tiếp theo chúng ta sẽ tìm hiểu nguyên lý cơ bản của nó.

Giả sử ta cần học hàm có dạng $f(\mathbf{x}; \mathbf{w})$. Khi đó việc học sẽ động đến các không gian:

- *Không gian đầu vào* (Input space) \mathcal{X} chứa các mẫu dữ liệu \mathbf{x} .
- *Không gian tham số* (Parameter space) \mathcal{W} chứa các tham số \mathbf{w} của mô hình.
- *Không gian hàm* (Function space) $\mathcal{H} = \{f(\mathbf{x}; \mathbf{w}) : \mathbf{x} \in \mathcal{X}, \mathbf{w} \in \mathcal{W}\}$. Đôi khi còn được gọi là không gian mô hình (model space), hoặc không gian giả thuyết (hypothesis space). Mỗi hàm bên trong được đánh chỉ số bởi tham số \mathbf{w} . Ta có thể hiểu rằng mỗi \mathbf{w} xác định duy nhất một hàm trong \mathcal{H} .

Đối với nhiều phương pháp ML, việc huấn luyện được đưa về việc giải bài toán

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{x}; \mathbf{w}), \mathbf{D}) \quad (110)$$

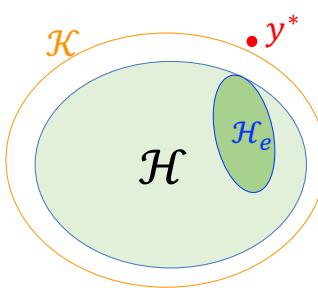
Trong đó $L(f(\mathbf{x}; \mathbf{w}), \mathbf{D})$ là một hàm lỗi thực nghiệm. Hàm này mô tả chất lượng của hàm f trên tập học \mathbf{D} . Vì mỗi tham số \mathbf{w} xác định một hàm trong \mathcal{H} , nên về bản chất ta có thể viết lại bài toán trên như sau:

$$f^* = \arg \min_{f \in \mathcal{H}} L(f, \mathbf{D}) \quad (111)$$

Nếu ta chỉ tập trung tối ưu lỗi trên tập học thì nguy cơ quá khớp có thể diễn ra. Lý do là trong lớp hàm \mathcal{H} có thể tồn tại nhiều hàm phức tạp mà có thể xấp xỉ tốt tập học. Khả năng này xảy ra càng cao khi ta chọn kiểu mô hình càng phức tạp, chẳng hạn mạng nơron nhân tạo hoặc cây quyết định. Kiểu mô hình càng phức tạp thì không gian hàm có thể càng lớn. Hình 66 minh họa một số không gian hàm khác nhau.

Thay vì cho thuật toán học khám phá toàn bộ không gian \mathcal{H} , ta hãy thu hẹp vùng tìm kiếm để đảm bảo rằng hàm tìm được vừa khớp tốt trên tập học và vừa tổng quát hoá tốt. Để làm được điều đó, ta hãy tìm:

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{x}; \mathbf{w}), \mathbf{D}) + \lambda g(\mathbf{w}) \quad (\text{Bài toán hiệu chỉnh}) \quad (112)$$



Hình 66: Ví dụ một số không gian mô hình. Không gian \mathcal{K} lớn hơn \mathcal{H} và có khả năng xấp xỉ y^* tốt hơn. \mathcal{H}_e mặc dù bé hơn nhưng khả năng xấp xỉ ngang ngửa với \mathcal{H} .

Nghĩa là ta đi tìm \mathbf{w}^* mà làm cho hàm $L_{reg} = L(f(\mathbf{x}; \mathbf{w}), \mathbf{D}) + \lambda g(\mathbf{w})$ đạt cực tiểu. Trong đó λ là hệ số phạt và thường là hằng số không âm, còn $g(\mathbf{w})$ là một hàm mô tả độ phức tạp của \mathbf{w} hoặc hàm f . Thông thường ta cần chọn hàm g để đảm bảo $g(\mathbf{w}) \geq 0, \forall \mathbf{w}$.

Hàm mục tiêu của (112) chia hai thành phần là lỗi L trên tập học và độ phức tạp g của mô hình. Do đó:

- Việc tối ưu hàm này gợi ý rằng chúng ta đi tìm \mathbf{w}^* mà vừa làm cho lỗi thực nghiệm bé và vừa đảm bảo g nhỏ. Nghĩa là hàm tìm được vừa khớp tập học tốt và vừa có độ phức tạp bé. Cách làm này tuân theo nguyên lý kéo Occam (Occam's razor).
- Chú ý rằng việc đảm bảo g bé nghĩa là ta đã thu hẹp không gian tìm kiếm từ \mathcal{W} về không gian con \mathcal{W}' nào đó, với $\mathcal{W}' \subset \mathcal{W}$.
- Khi thu hẹp vùng tìm kiếm, chúng ta đã giảm bớt khả năng gặp vấn đề quá khớp. Lý do là các hàm với $\mathbf{w} \in \mathcal{W}'$ không quá phức tạp.
- Hệ số phạt λ cho phép ta cân bằng giữa lỗi trên tập học và khả năng tổng quát hoá cho các mẫu dữ liệu trong tương lai. Nếu λ lớn quá thì ta đang phạt quá mạnh vào độ phức tạp và hàm học được có xu hướng rất đơn giản. Khi đó kém khớp có thể xảy ra. Ngược lại, nếu λ bé quá thì thuật toán học sẽ tập trung vào tối ưu lỗi trên tập học, và khả năng quá khớp có thể diễn ra. Do đó trong thực tế, chúng ta cần chọn hệ số λ cho phù hợp với bài toán đang giải quyết.

Trong thực tế, có một số hàm g thường được chọn là chuẩn của vectơ. Ví dụ nếu dùng dạng chuẩn ℓ_1 thì $g(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$, nếu dùng dạng chuẩn ℓ_2 thì $g(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^n w_i^2$, trong đó giả sử $\mathbf{w} = (w_1, \dots, w_n)$. Ngoài ra có thể dùng dạng chuẩn ℓ_0 hoặc ℓ_p nào đó.

4.3.4 Vài ví dụ

Trong các chương trước đây, chúng ta đã tìm hiểu một số phương pháp đã dùng hiệu chỉnh, bao gồm Ridge, Lasso, và SVM.

Thực vậy phương pháp Ridge học ra hàm hồi qui $f(\mathbf{x}; \mathbf{w}^*)$ bằng cách tìm vectơ

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} RSS(f) + \lambda \|\mathbf{w}\|_2^2 \quad (113)$$

trong đó $RSS(f)$ mô tả lỗi trên tập học $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_M, y_M)\}$ cho trước. Rõ ràng Ridge đã sử dụng hiệu chỉnh với chuẩn ℓ_2 để tìm ra hàm mà có vectơ trọng số nhỏ. Tương tự như thế, phương pháp Lasso dùng hiệu chỉnh với chuẩn ℓ_1 . Các phương pháp này có thể hiệu quả hơn phương pháp bình phương tối thiểu, nhờ khả năng cân bằng lỗi trên tập học và độ phức tạp của mô hình học được.

Xét phương pháp SVM tuyển tính đã trình bày trong mục 3.3.4. SVM học một siêu phẳng mà có mức lề lớn nhất, nghĩa là tìm hệ số \mathbf{w}^* và b^* mà là nghiệm của bài toán

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2 \quad \text{sao cho } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \forall i \in \{1, \dots, M\} \quad (114)$$

Để đương đầu với những bài toán mà có các lớp chồng lấn nhau, SVM sẽ đưa về

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^M \xi_i \quad \text{sao cho } \begin{cases} y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \\ \xi_i \geq 0, \forall i \in \{1, \dots, M\} \end{cases} \quad (115)$$

Ở đây $\sum_{i=1}^M \xi_i$ mô tả lỗi phân loại cho các mẫu dữ liệu trong \mathcal{D} . Ta có thể coi $\frac{1}{2} \|\mathbf{w}\|_2^2$ là đại lượng phạt. Như vậy SVM cũng dùng hiệu chỉnh để cân bằng giữa việc lỗi trên tập \mathcal{D} và khả năng tổng quát hoá.

Ngoài các cách hiệu chỉnh khá đơn giản ở trên, có nhiều phương pháp hiệu chỉnh phổ biến khác trong ML. Dưới đây là một vài ví dụ:

- *Dropout*: là một phương pháp vừa đơn giản và hiệu quả [Srivastava et al., 2014, Krizhevsky et al., 2012]. Ý tưởng cơ bản là tại mỗi bước lặp trong quá trình huấn luyện, ta chỉ lấy một phần của mô hình một cách ngẫu nhiên để cập nhật. Yêu tố ngẫu nhiên ở đây đóng vai trò quan trọng để mang lại vai trò hiệu chỉnh. Ban đầu Dropout được đề xuất cho mạng nơron, nhưng về sau được dùng rộng rãi ở các lớp

mô hình khác.

- *Chuẩn hoá (Normalization)*: là một phương pháp thường được sử dụng cho các mô hình phức tạp, như mạng nơron. Một số phương pháp chuẩn hoá gồm Batch normalization (BN) [Ioffe and Szegedy, 2015], Group normalization [Wu and He, 2020],... Chúng có thể giúp ta đảm bảo hành vi không quá khác biệt trong mỗi vùng nhỏ (ví dụ nhóm tham số, nhóm tín hiệu đầu vào,...). Ví dụ BN đảm bảo rằng tín hiệu đầu vào tại mỗi nơron tuân theo phân bố chuẩn.
- *Tăng cường dữ liệu (data augmentation)*: là một phương tiện để giúp tăng hiệu quả của một mô hình ML [Shorten and Khoshgoftaar, 2019]. Ý tưởng cơ bản là ta sẽ tạo ra nhiều phiên bản mới của một mẫu dữ liệu bằng cách sử dụng một (vài) phép biến đổi đơn giản (chẳng hạn xê dịch, xoay ảnh, đổi màu ảnh, cắt bớt ảnh, thêm nhiễu trắng, xoá bớt từ, ...), rồi dùng chúng để huấn luyện. Các phép biến đổi đó không nên quá mạnh để những ngữ nghĩa chính trong một mẫu dữ liệu vẫn có trong các phiên bản mới. Về cơ bản, cách làm này sẽ giúp mô hình học được giảm ảnh hưởng của nhiễu hay sự thay đổi nhỏ ở đầu vào.
- *Dừng học sớm (early stopping)*: Đối với một số mô hình phức tạp (ví dụ cây quyết định và mạng nơron), nếu ta huấn luyện quá nhiều thì có thể dẫn tới việc học quá mức và quá khớp có thể diễn ra. Do đó, một cách để giảm vấn đề này là cắt bớt số lần lặp khi huấn luyện. Cách làm này cũng tương tự như việc sử dụng hiệu chỉnh [Caruana et al., 2000], bởi thuật toán học chưa kịp khám phá toàn bộ không gian tham số.

4.3.5 Vai trò của hiệu chỉnh

Tiếp theo chúng ta sẽ phân tích một vài vai trò của hiệu chỉnh đối với các mô hình học máy. Có hai vai trò quan trọng, xuất phát từ hai góc nhìn khác nhau: vai trò MAP và thu hẹp không gian.

a. Vai trò MAP: Hiệu chỉnh có thể được coi là cách vận dụng *tri thức bên ngoài hoặc mong muốn của chúng ta* vào quá trình huấn luyện để tìm ra một hàm mà có xác suất hậu nghiệm lớn nhất. Nghĩa là ta đã sử dụng phương pháp MAP để học.

Để thấy được điều đó, ta xét bài toán (112). Giả sử các mẫu dữ liệu trong \mathcal{D} được

sinh ra bởi một phân bố xác suất nào đó mà hàm log likelihood có thể viết dưới dạng $-L(f, \mathbf{D})$, và \mathbf{w} là một biến ngẫu nhiên tuân theo phân bố có hàm mật độ là $p(\mathbf{w}) \propto \exp(-\lambda g(\mathbf{w}))$. Khi đó ta có thể viết lại bài toán (112) như sau:

$$\begin{aligned}\mathbf{w}^* &= \arg \max_{\mathbf{w} \in \mathcal{W}} \{-L(f(\mathbf{x}; \mathbf{w}), \mathbf{D}) - \lambda g(\mathbf{w})\} = \arg \max_{\mathbf{w} \in \mathcal{W}} \log \Pr(\mathbf{D}|\mathbf{w}) + \log \Pr(\mathbf{w}) \\ &= \arg \max_{\mathbf{w} \in \mathcal{W}} \log \Pr(\mathbf{w}|\mathbf{D}) = \arg \max_{\mathbf{w} \in \mathcal{W}} \Pr(\mathbf{w}|\mathbf{D})\end{aligned}\quad (116)$$

Như vậy ta tìm \mathbf{w}^* mà có xác xuất hậu nghiệm lớn nhất. Đây chính là phương pháp MAP.

Ta hãy xét mô hình xác suất đơn giản dưới đây cho bài toán hồi qui. Giả sử các mẫu dữ liệu (\mathbf{x}, y) được sinh bởi quá trình sinh như sau:

- Sinh vector $\mathbf{w} \sim Normal(0, \sigma^2 \mathbf{I})$
- Sinh mẫu dữ liệu $\mathbf{x} \sim Normal(0, \frac{1}{n} \mathbf{I})$
- Tính $y = \mathbf{w}^T \mathbf{x}$

trong đó \mathbf{I} là ma trận đơn vị cỡ $n \times n$, *Normal* là phân bố chuẩn.

Có thể thấy mô hình này mô tả một hàm hồi qui tuyến tính. Tuy nhiên do \mathbf{w} và \mathbf{x} đều là biến ngẫu nhiên nên hàm hồi qui này cũng là hàm ngẫu nhiên. Để tìm hàm này từ tập học \mathbf{D} theo phương pháp MAP, chúng ta đi tìm

$$\mathbf{w}^* = \arg \max_{\mathbf{w} \in \mathcal{W}} \Pr(\mathbf{w}|\mathbf{D}) = \arg \max_{\mathbf{w} \in \mathcal{W}} \log [\Pr(\mathbf{D}|\mathbf{w}) \log \Pr(\mathbf{w})] \quad (117)$$

$$= \arg \max_{\mathbf{w} \in \mathcal{W}} \log \Pr(\mathbf{D}|\mathbf{w}) + \log \Pr(\mathbf{w}) \quad (118)$$

$$= \arg \max_{\mathbf{w} \in \mathcal{W}} \sum_{(\mathbf{x}, y) \in \mathbf{D}} \log \Pr(\mathbf{x}, y|\mathbf{w}) + \log \Pr(\mathbf{w}) \quad (119)$$

Vì mỗi thành phần của \mathbf{x} tuân theo phân bố chuẩn với kỳ vọng 0 và phương sai $\frac{1}{n}$, nên $y = w_1 x_1 + \dots + w_n x_n$ cũng là biến ngẫu nhiên. Với một \mathbf{w} cho trước, theo tính chất của tổng các biến ngẫu nhiên, ta dễ thấy y tuân theo phân bố chuẩn với kỳ vọng 0 và phương

sai 1. Do đó sử dụng hàm mật độ của phân bố chuẩn vào bài toán (119) ta thu được:

$$\mathbf{w}^* = \arg \max_{\mathbf{w} \in \mathcal{W}} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \frac{1}{2} (y - \mathbf{w}^T \mathbf{x})^2 + \frac{1}{2\sigma^2} \mathbf{w}^T \mathbf{w} + \text{constant} \quad (120)$$

$$= \arg \max_{\mathbf{w} \in \mathcal{W}} \sum_{(\mathbf{x}, y) \in \mathcal{D}} (y - \mathbf{w}^T \mathbf{x})^2 + \frac{1}{\sigma^2} \mathbf{w}^T \mathbf{w} \quad (121)$$

Biểu thức cuối cùng này cho ta thấy rằng mô hình hồi qui đã được huấn luyện bằng phương pháp Ridge, được trình bày trong Chương 2, với hệ số phạt $\lambda = \sigma^{-2}$. Nếu chọn σ lớn thì λ sẽ nhỏ và độ lớn của \mathbf{w}^* sẽ có xu hướng lớn. Điều này có thể được suy ra từ bản chất của phân bố chuẩn, hoặc từ biểu thức (121). Ngược lại, chọn σ nhỏ thì độ lớn của \mathbf{w}^* sẽ có xu hướng nhỏ. Hình 67 mô tả minh họa kết quả trong thực tế.

Như vậy trong ví dụ trên chúng ta đã thấy việc dùng hiệu chỉnh có bản chất tương tự như việc huấn luyện theo phương pháp MAP. Hàm hiệu chỉnh đã chứa tri thức hoặc mong muốn của chúng ta về hàm cần tìm được. Ví dụ, biểu thức (121) cho thấy việc chọn σ nhỏ đã chứa mong muốn của ta rằng hàm hồi qui cần có vectơ trọng số với kích cỡ nhỏ.

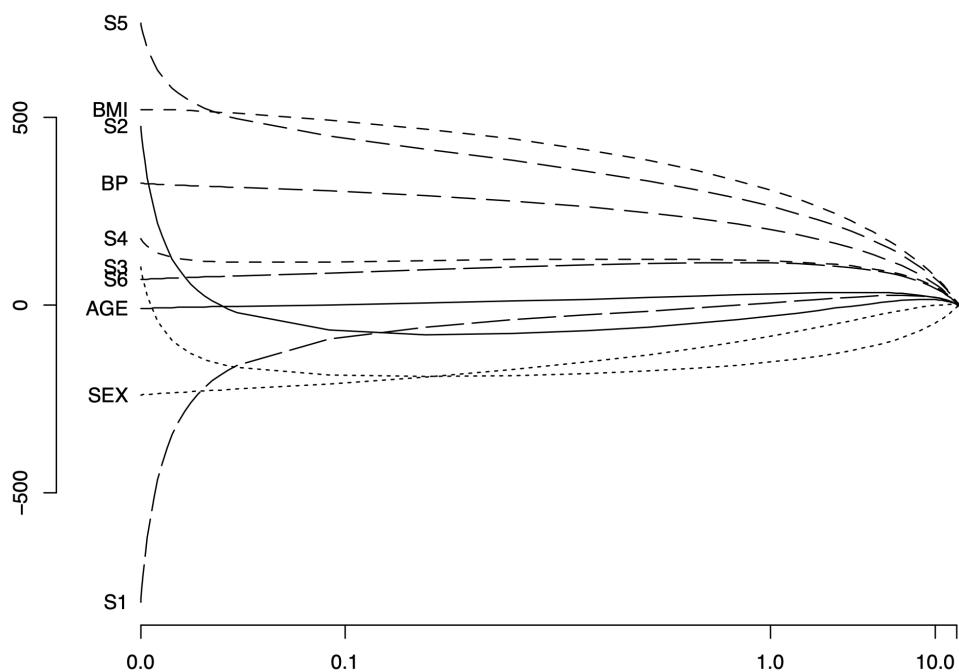
b. Vai trò Hạn chế không gian tìm kiếm: Nếu không dùng hiệu chỉnh thì giải thuật học có thể tìm kiếm trên toàn bộ không gian tham số \mathcal{W} để tìm ra một nghiệm \mathbf{w}^* cho bài toán (110). Tuy nhiên khi dùng hiệu chỉnh, giải thuật học chỉ cần tìm kiếm trong một vùng nhỏ của \mathcal{W} .

Để thấy được điều đó, hãy xét bài toán tối ưu sau với các hàm L và g khả vi:

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathcal{W}} L(f, \mathbf{D}) \text{ such that } g(\mathbf{w}) \leq s \quad (122)$$

trong đó s là một hằng số cho trước. Rõ ràng điều kiện $g(\mathbf{w}) \leq s$ đã thu hẹp không gian tìm kiếm từ \mathcal{W} về vùng nhỏ hơn. Theo Karush-Kuhn-Tucker, tồn tại một hằng số $\lambda^* \geq 0$ sao cho $(\mathbf{w}^*, \lambda^*)$ là điểm dừng (saddle point) của hàm $L_{la} = L(f, \mathbf{D}) + \lambda(g(\mathbf{w}) - s)$. Tức L_{la} có đạo hàm 0 tại điểm (\mathbf{w}^*, λ) .

Khi dùng hiệu chỉnh, ta cần tối ưu hàm $L_{reg} = L(f, \mathbf{D}) + \lambda g(\mathbf{w})$. Do ta thường cố định λ trước khi huấn luyện cho nên nghiệm \mathbf{w}^* tối ưu của hàm này cũng là nghiệm tối ưu của hàm $L_{la} = L(f, \mathbf{D}) + \lambda(g(\mathbf{w}) - s)$, với s là hằng số. Kết hợp điều này với KKT gọi ý rằng việc sử dụng hiệu chỉnh về bản chất là đi tìm một nghiệm nằm trong một



Hình 67: Xu hướng giảm độ lớn của trọng số khi tăng giá trị λ trong Ridge [Hesterberg, 2008]. Đây là kết quả của Ridge trên một tập dữ liệu, trong đó mỗi mẫu được biểu diễn bằng các thuộc tính $\{S1, S2, S3, S4, S5, S6, AGE, SEX, BMI, BP\}$.

vùng con của \mathcal{W} . Hay nói cách khác, hiệu chỉnh đã giúp thu hẹp không gian tìm kiếm cho giải thuật học.

Hình 67 cung cấp vài gợi ý về khả năng thu hẹp vùng tìm kiếm. Nếu chọn hệ số phạt λ bé thì không gian tìm kiếm còn rộng. Nếu chọn hệ số phạt lớn thì không gian tìm kiếm có xu hướng bé.

TÀI LIỆU THAM KHẢO

- K. M. Ahmed, N. M. El-Makky, and Y. Taha. A note on "beyond market baskets: generalizing association rules to correlations". *ACM SIGKDD Explorations Newsletter*, 1(2):46–48, 2000.
- E. Alpaydin. *Introduction to Machine Learning*. MIT press, 2020.
- D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. 2007.
- L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.
- D. M. Blei. Probabilistic topic models. *Tutorial at ICML*, 2012.
- D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003.
- D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.
- S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 265–276, 1997a.
- S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 255–264, 1997b.
- R. Caruana, S. Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in Neural Information Processing Systems*, 13, 2000.
- F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

- I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- J. Han, J. Pei, and H. Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2023.
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2017.
- R. M. Hayes. Measurement of information. *Information Processing & Management*, 29(1):1–11, 1993.
- T. Hesterberg. Least angle and l1 penalized regression: A review. *Statistics Surveys*, 2 (2):61–93, 2008.
- J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.
- J. D. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.
- K. C. Laudon and J. P. Laudon. *Management information systems: new approaches to organization and technology*. Prentice Hall PTR, 1997.
- B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer Publishing Company, Incorporated, 2nd edition, 2011. ISBN 3642194591.
- L. E. Long and N. Long. *Computers: Information Technology in Perspective*. Prentice Hall, USA, 10th edition, 2002.
- B. C. McNurlin and R. H. Sprague. *Information systems management in practice*. Prentice-Hall, Inc., 2005.

- T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.
- R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.
- C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.
- E. H. Shortliffe and B. G. Buchanan. A model of inexact reasoning in medicine. *Bellman Prize in Mathematical Biosciences*, 1990.
- H. A. Simon. Why should machines learn? In *Machine Learning*, pages 25–37. Morgan Kaufmann, 1983.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.
- D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- Y. Wu and K. He. Group normalization. *International Journal of Computer Vision*, 128(3):742–756, 2020.
- J. A. Zachman. A framework for information systems architecture. *IBM systems journal*, 26(3):276–292, 1987.