Chương 4

Phương pháp tính

Các thuật toán học máy thường đòi hỏi một lượng lớn tính toán số. Điều này thường ám chỉ các thuật toán giải quyết các bài toán toán học bằng các phương pháp cập nhật các ước lượng của nghiệm thông qua một quá trình lặp, thay vì suy ra một công thức giải tích cho lời giải đúng. Các thao tác phổ biến bao gồm tối ưu hóa (tìm giá trị của biến số làm cực tiểu hoặc cực đại một hàm số) và giải các hệ phương trình tuyến tính. Ngay cả việc đánh giá một hàm số trên máy tính kỹ thuật số cũng có thể khó khăn khi hàm số đó bao gồm các số thực, vốn không thể được biểu diễn chính xác bằng một lượng bộ nhớ hữu hạn.

4.1 Tràn số và hạ số

Khó khăn cơ bản trong việc thực hiện đại lượng liên tục trong toán học trên máy tính số là ta cần biểu diễn vô số số thực bằng một số lượng mẫu bit hữu hạn. Điều này có nghĩa là đối với hầu hết các số thực, ta sẽ gặp phải một số sai số xấp xỉ khi biểu diễn số đó trên máy tính. Trong nhiều trường hợp, đây chỉ là lỗi làm tròn. Lỗi làm tròn có thể gây ra vấn đề, đặc biệt khi nó tích lũy qua nhiều phép toán và có

thể khiến các thuật toán hoạt động tốt trong lý thuyết nhưng thất bại trong thực tế nếu chúng không được thiết kế để giảm thiểu sự tích lũy của lỗi làm tròn.

Một dạng lỗi làm tròn đặc biệt nghiêm trọng là **hạ số** (underflow). Hạ số xảy ra khi các số gần bằng 0 bị làm tròn về 0. Nhiều hàm số có hành vi khác nhau đáng kể khi đối số của chúng bằng 0 so với khi là một số dương nhỏ. Ví dụ, ta thường muốn tránh chia cho 0 (một số môi trường phần mềm sẽ đưa ra ngoại lệ khi điều này xảy ra, trong khi những môi trường khác sẽ trả về một kết quả với giá trị giữ chỗ "không phải là số" — not-a-number) hoặc lấy logarit của 0 (điều này thường được coi là $-\infty$, và nếu sử dụng cho nhiều phép toán số học tiếp theo, nó sẽ trở thành "không phải là số").

Một dạng lỗi số học gây hại cao khác là **tràn số** (overflow). Tràn số xảy ra khi các số có giá trị lớn bị xấp xỉ thành ∞ hoặc $-\infty$. Các phép toán số học tiếp theo thường sẽ biến những giá trị vô hạn này thành các giá trị "không phải là số".

Một ví dụ về một hàm cần được ổn định để chống lại tràn số và hạ lưu số là hàm softmax. Hàm softmax thường được sử dụng để dự đoán các xác suất liên quan đến phân phối Bernoulli bội. Hàm softmax được định nghĩa là:

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}. \tag{4.1}$$

Xét trường hợp khi tất cả các giá trị x_i đều bằng một hằng số c. Về mặt lý thuyết, ta có thể thấy rằng tất cả các đầu ra của hàm softmax sẽ bằng $\frac{1}{n}$. Tuy nhiên, về mặt tính toán, điều này có thể không xảy ra khi c có giá trị lớn. Nếu c âm và rất nhỏ, thì e^c sẽ bị hạ số, dẫn đến mẫu số của hàm softmax trở thành 0, làm cho kết quả cuối cùng không xác định. Khi c dương và rất lớn, e^c sẽ gặp tràn số, lại khiến cho biểu thức không xác định. Cả hai vấn đề này có thể được giải quyết bằng cách tính softmax $(\boldsymbol{z})$ với $\boldsymbol{z} = \boldsymbol{x} - \max_i x_i$. Một biến đổi đại số đơn giản cho thấy rằng giá trị của hàm softmax không thay đổi khi cộng hoặc trừ một hằng số từ véctơ đầu vào. Việc trừ $\max_i x_i$ đảm bảo rằng đối số lớn nhất của hàm mũ là 0, loại trừ khả năng tràn số. Tương tự, ít nhất một thành phần trong mẫu số có giá trị là 1, loại trừ khả năng hạ số trong mẫu số dẫn đến phép chia cho 0.

Vẫn còn một vấn đề nhỏ. Hạ số trong tử số vẫn có thể khiến cho toàn bộ biểu thức đánh giá về 0. Điều này có nghĩa là nếu ta triển khai hàm log softmax (x) bằng cách chạy chương trình con softmax trước, sau đó truyền kết quả vào hàm log, ta có thể thu được giá trị sai lầm $-\infty$. Thay vào đó, ta phải triển khai một hàm riêng biệt để tính log softmax theo cách ổn định về mặt số học. Hàm log softmax có thể

được ổn định bằng cách sử dụng cùng thủ thuật mà ta đã sử dụng để ổn định hàm softmax.

Phần lớn cuốn sách không trình bày chi tiết tất cả các vấn đề số học liên quan đến việc triển khai các thuật toán được mô tả trong cuốn sách này. Những nhà phát triển các thư viện cấp thấp nên lưu ý đến các vấn đề số học khi triển khai các thuật toán học sâu. Hầu hết độc giả của cuốn sách này có thể chỉ cần dựa vào các thư viện cấp thấp cung cấp các triển khai ổn định. Trong một số trường hợp, có thể triển khai một thuật toán mới và để triển khai đó tự động được ổn định. Theano (*Theano: A CPU and GPU Math Compiler in Python*, Bergstra và các tác giả, 2010, [1]; *Theano: new features and speed improvements*, Bastien và các tác giả, 2012, [2]) là một ví dụ về phần mềm tự động phát hiện và ổn định nhiều biểu thức số học không ổn định thường gặp trong bối cảnh học sâu.

4.2 Điều kiện xấu

Thuật ngữ điều kiện xấu hay tốt đề cập đến mức độ nhanh chóng mà một hàm thay đổi khi các đầu vào của nó bị xáo trộn một chút. Các hàm thay đổi nhanh chóng khi đầu vào bị biến đổi nhẹ có thể gây ra vấn đề trong tính toán khoa học, vì lỗi làm tròn trong đầu vào có thể dẫn đến thay đổi lớn trong đầu ra.

Xét hàm $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}$, trong đó $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ có phân tích giá trị riêng. **Số điều kiện** của nó là

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right| . \tag{4.2}$$

Đó là tỷ số giữa độ lớn của giá trị riêng lớn nhất và nhỏ nhất của ma trận A. Khi số điều kiện này lớn, phép nghịch đảo ma trận trở nên đặc biệt nhạy cảm với lỗi trong đầu vào.

Độ nhạy này là thuộc tính nội tại của ma trận, không phải là kết quả của lỗi làm tròn trong quá trình nghịch đảo ma trận. Các ma trận có điều kiện xấu sẽ khuếch đại các lỗi đã tồn tại sẵn khi ta nhân với ma trận nghịch đảo thực sự. Trong thực tế, lỗi sẽ được khuếch đại thêm bởi các sai số số học trong chính quá trình nghịch đảo.

4.3 Tối ưu dựa trên gradient

Hầu hết các thuật toán học sâu đều liên quan đến việc tối ưu hóa ở một mức độ nào đó. Tối ưu hóa đề cập đến nhiệm vụ tìm cách cực tiểu hoặc cực đại hóa

một hàm số $f(\boldsymbol{x})$ bằng cách thay đổi giá trị của $\boldsymbol{x}$. Thông thường, các bài toán tối ưu hóa được diễn đạt dưới dạng cực tiểu hóa $f(\boldsymbol{x})$. Nếu cần cực đại hóa, ta có thể thực hiện bằng cách sử dụng thuật toán cực tiểu hóa cho hàm $-f(\boldsymbol{x})$.

Hàm mà chúng ta muốn cực tiểu hóa hoặc cực đại hóa được gọi là **hàm mục tiêu** hoặc **hàm tiêu chí**. Khi thực hiện cực tiểu hóa, hàm này còn được gọi là

- **hàm chi phí**: tổng chi phí mà mô hình tạo ra khi đưa ra dự đoán; hoặc
- **hàm mất mát**: đo lường sai số cho một dự đoán cụ thể; hoặc
- **hàm lỗi**: đôi khi được dùng như một cách gọi khác của hàm mất mát, đặc biệt là trong ngữ cảnh đánh giá sai số tổng thể của một mô hình.

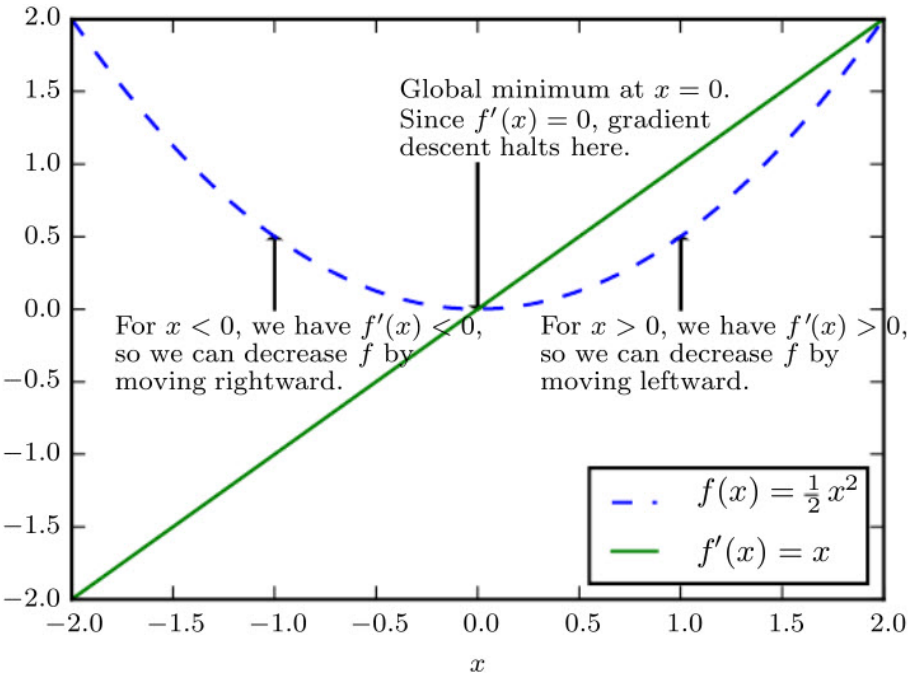
Các thuật ngữ này đều mô tả quá trình tìm cách làm giảm sự sai lệch giữa dự đoán của mô hình và giá trị thực tế trong quá trình học máy. Trong sách này, các thuật ngữ này được sử dụng thay thế lẫn nhau, mặc dù một số tài liệu trong lĩnh vực học máy có thể gán những ý nghĩa đặc biệt cho một số thuật ngữ này.

Ta thường ký hiệu giá trị cực tiểu hoặc cực đại của một hàm bằng một dấu chỉ số trên $*$. Ví dụ, ta có thể viết $\boldsymbol{x}^* = \arg\min\ f(\boldsymbol{x})$, để chỉ rằng $\boldsymbol{x}^*$ là giá trị của $\boldsymbol{x}$ tại đó hàm $f(x)$ đạt giá trị nhỏ nhất. Mặc dù giả định rằng người đọc đã quen thuộc với phép tính vi phân, dưới đây là một phần tóm tắt ngắn về cách các khái niệm trong phép tính vi phân liên quan đến bài toán tối ưu hóa.

Giả sử ta có một hàm số $y = f(x)$, trong đó cả x và y đều là các số thực. **Đạo hàm** của hàm này được ký hiệu là $f'(x)$ hoặc $\frac{dy}{dx}$. Đạo hàm $f'(x)$ cho biết độ dốc của hàm $f(x)$ tại điểm x. Nói cách khác, đạo hàm xác định cách thay đổi của đầu ra tương ứng với một thay đổi nhỏ trong đầu vào: $f(x + \varepsilon) \approx f(x) + \varepsilon \cdot f'(x)$.

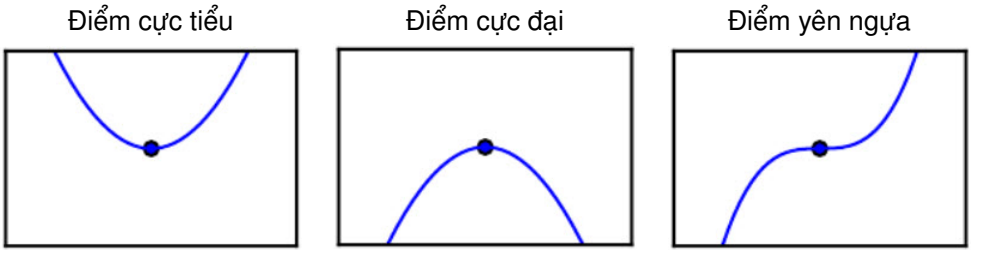
Đạo hàm rất hữu ích trong việc cực tiểu hóa một hàm vì nó cho ta biết cách thay đổi x để làm cải thiện nhỏ giá trị của y. Ví dụ, ta biết rằng $f\left(x - \varepsilon \text{sign}\left(f'(x)\right)\right)$ sẽ nhỏ hơn $f(x)$ với một giá trị ε đủ nhỏ. Do đó, ta có thể giảm giá trị của $f(x)$ bằng cách di chuyển x theo các bước nhỏ theo chiều ngược lại với dấu của đạo hàm. Kỹ thuật này được gọi là **hướng giảm** (*General Method for Solving Systems of Simultaneous Equations.* Cauchy, 1847, [3]). Xem Hình 4.1 để có ví dụ minh họa về kỹ thuật này.

Khi $f'(x) = 0$, đạo hàm không cung cấp thông tin về hướng di chuyển. Các điểm mà $f'(x) = 0$ được gọi là **điểm tới hạn** hoặc **điểm dừng**. **Cực tiểu địa phương** là một điểm tại đó $f(x)$ nhỏ hơn tại tất cả các điểm lân cận, vì vậy không thể giảm $f(x)$ bằng cách thực hiện các bước nhỏ. **Cực đại địa phương** là một điểm tại đó $f(x)$ lớn hơn tại tất cả các điểm lân cận, nên không thể tăng $f(x)$ bằng các bước



Hình 4.1: Một minh họa cho thấy cách thuật toán hướng giảm sử dụng đạo hàm của hàm số để di chuyển theo hướng giảm dần của hàm số nhằm đạt đến điểm cực tiểu.

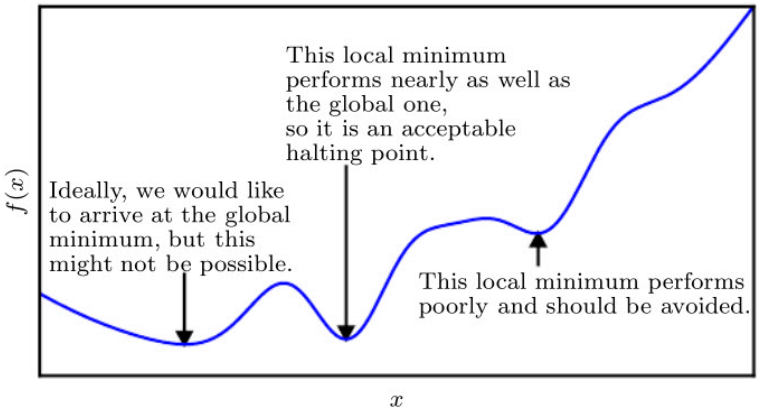
nhỏ. Một số điểm tới hạn không phải là cực đại hay cực tiểu, được gọi là **điểm yên ngựa**. Xem Hình 4.2 để có ví dụ về từng loại điểm tới hạn.



Hình 4.2: Ví dụ về ba loại điểm tới hạn trong không gian 1 chiều. Một điểm tới hạn là điểm có độ dốc bằng không. Điểm này có thể là cực tiểu địa phương, nơi mà giá trị thấp hơn các điểm lân cận; cực đại địa phương, nơi mà giá trị cao hơn các điểm lân cận; hoặc điểm yên ngựa, nơi mà các điểm lân cận có giá trị vừa cao hơn vừa thấp hơn điểm đó.

Một điểm mà $f(x)$ đạt giá trị thấp nhất tuyệt đối được gọi là điểm **cực tiểu toàn cục**. Có thể tồn tại một hoặc nhiều điểm cực tiểu toàn cục của hàm số. Cũng có thể

có các điểm cực tiểu địa phương không phải là tối ưu toàn cục. Trong bối cảnh học sâu, ta tối ưu hóa các hàm số có thể có nhiều điểm cực tiểu địa phương không phải nghiệm tối ưu và nhiều điểm yên ngựa được bao quanh bởi các vùng rất phẳng. Tất cả những điều này khiến việc tối ưu hóa trở nên rất khó khăn, đặc biệt trong trường hợp hàm số nhiều biến. Do đó, ta thường chấp nhận việc tìm một giá trị của f đủ thấp, nhưng không nhất thiết phải là giá trị tối thiểu theo định nghĩa. Xem Hình 4.3 để có ví dụ minh họa.



Hình 4.3: Các thuật toán tối ưu có thể không tìm được điểm cực tiểu toàn cục khi có nhiều điểm cực tiểu địa phương hoặc các bề mặt phẳng. Trong bối cảnh học sâu, ta thường chấp nhận những nghiệm này mặc dù chúng không thực sự là điểm cực tiểu, miễn là chúng tương ứng với các giá trị đủ thấp của hàm chi phí.

Ta thường cực tiểu hóa các hàm nhiều biến: $f : \mathbb{R}^n \to \mathbb{R}$. Để khái niệm "cực tiểu hóa" có ý nghĩa, hàm này phải nhận giá trị chỉ có một thành phần (dạng vô hướng).

Đối với các hàm số nhiều biến, ta phải sử dụng khái niệm **đạo hàm riêng**. Đạo hàm riêng $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ đo sự thay đổi của f khi chỉ có biến x_i tăng tại điểm x. **Gradient** tổng quát hóa khái niệm đạo hàm thành trường hợp mà đạo hàm được xem như một véctơ: gradient của f là véctơ chứa tất cả các đạo hàm riêng, được ký hiệu là $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Phần tử thứ i của gradient là đạo hàm riêng của f theo x_i. Trong không gian nhiều chiều, các điểm tới hạn là những điểm mà mọi phần tử của gradient đều bằng không.

Đạo hàm theo hướng $\boldsymbol{u}$ (một véctơ đơn vị) là độ dốc của hàm f theo hướng $\boldsymbol{u}$. Nói cách khác, đạo hàm theo hướng là đạo hàm của hàm $f(\boldsymbol{x} + \alpha\boldsymbol{u})$ theo α, được tính tại $\alpha = 0$. Sử dụng quy tắc đạo hàm của hàm hợp, ta có thể thấy rằng $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha\boldsymbol{u})$ khi $\alpha = 0$ được tính bằng $\boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.

Để cực tiểu hóa f, ta muốn tìm hướng mà f giảm nhanh nhất. Ta có thể làm điều này bằng cách sử dụng đạo hàm theo hướng:

$$\min_{\boldsymbol{u}:\, \boldsymbol{u}^T\boldsymbol{u}=1} \boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.3}$$

$$= \min_{\boldsymbol{u}:\, \boldsymbol{u}^T\boldsymbol{u}=1} \|\boldsymbol{u}\|_2 \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\|_2 \cos\theta \tag{4.4}$$

trong đó θ là góc giữa $\boldsymbol{u}$ và gradient. Thay $\|\boldsymbol{u}\|_2 = 1$ và bỏ qua các yếu tố không phụ thuộc vào $\boldsymbol{u}$, điều này rút gọn thành $\min_{\boldsymbol{u}} \cos\theta$. Giá trị này nhỏ nhất khi $\boldsymbol{u}$ hướng ngược lại với gradient. Nói cách khác, gradient chỉ trực tiếp lên phía trên, và gradient âm chỉ xuống phía dưới. Ta có thể giảm f bằng cách di chuyển theo hướng của gradient âm. Phương pháp này được gọi là **phương pháp dốc nhất** hoặc **hướng giảm**.

Phương pháp dốc nhất đề xuất một điểm mới

$$\boldsymbol{x}' = \boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.5}$$

trong đó ε là **tốc độ học**, một hệ số dương xác định kích thước bước đi. Ta có thể chọn ε theo nhiều cách khác nhau. Một cách phổ biến là đặt ε nhận một hằng số nhỏ. Đôi khi, ta có thể giải cỡ bước này vì nó làm cho đạo hàm theo hướng bằng 0. Một cách tiếp cận khác là đánh giá $f(\boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$ với một số giá trị của ε và chọn giá trị cho kết quả nhỏ nhất của hàm mục tiêu. Chiến lược này được gọi là **tìm kiếm theo đường thẳng**.

Phương pháp dốc nhất hội tụ khi mọi phần tử của gradient bằng 0 (hoặc, trong thực tế, rất gần 0). Trong một số trường hợp, ta có thể tránh việc chạy thuật toán lặp này và tìm trực tiếp điểm tới hạn bằng cách giải phương trình $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{0}$ theo $\boldsymbol{x}$.

Mặc dù phương pháp hướng giảm chỉ giới hạn khi tối ưu hóa trong không gian liên tục, khái niệm tổng quát về việc lặp lại các bước di chuyển nhỏ (xấp xỉ bước tốt nhất) hướng đến các cấu hình tốt hơn có thể được mở rộng cho không gian rời rạc. Việc leo lên một hàm mục tiêu với các tham số rời rạc được gọi là **thuật toán leo đồi** (*Artificial Intelligence: A Modern Approach*, Russel và Norvig, 2021, [4]).

4.3.1 Mở rộng khái niệm gradient: ma trận Jacobi và ma trận Hess

Đôi khi ta cần tìm tất cả các đạo hàm riêng của một hàm mà đầu vào và đầu ra đều là các véctơ, hay hàm véctơ nhiều biến. Ma trận chứa tất cả các đạo hàm

riêng đó được gọi là **ma trận Jacobi**. Cụ thể, nếu ta có một hàm $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$, thì ma trận Jacobi $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ của $\boldsymbol{f}$ được định nghĩa bởi $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$.

Đôi khi, ta cũng quan tâm đến **đạo hàm cấp hai**, tức là đạo hàm của một đạo hàm. Ví dụ, đối với hàm $f : \mathbb{R}^n \to \mathbb{R}$, đạo hàm theo biến x_i của đạo hàm của f theo biến x_j được ký hiệu là $\frac{\partial^2}{\partial x_i \partial x_j} f$. Trong trường hợp hàm số một biến, ta có thể ký hiệu $\frac{d^2}{dx^2} f$ bằng $f''(x)$. Đạo hàm cấp hai cho biết đạo hàm cấp một sẽ thay đổi như thế nào khi ta thay đổi biến đầu vào. Điều này quan trọng vì nó cho ta biết liệu một bước gradient có mang lại cải thiện nhiều như mong đợi dựa trên gradient hay không. Ta có thể coi đạo hàm bậc hai là đo **độ cong**. Giả sử ta có một hàm bậc hai (nhiều hàm xuất hiện trong thực tế không phải là bậc hai nhưng có thể được xấp xỉ bởi hàm bậc hai trong một phạm vi nhỏ). Nếu hàm này có đạo hàm cấp hai bằng 0, thì không có độ cong nào cả. Nó là một đường hoàn toàn phẳng, và giá trị của nó có thể dự đoán chỉ dựa trên gradient. Nếu gradient là 1, ta có thể thực hiện một bước có kích thước ε theo hướng ngược với gradient, và hàm mục tiêu sẽ giảm đi ε. Nếu đạo hàm cấp hai âm, hàm số cong xuống dưới, do đó hàm mục tiêu sẽ thực tế giảm nhiều hơn ε. Cuối cùng, nếu đạo hàm bậc hai dương, hàm số cong lên trên, do đó hàm mục tiêu có thể giảm ít hơn ε. Hãy xem Hình 4.4 để thấy cách các dạng độ cong khác nhau ảnh hưởng đến mối quan hệ giữa giá trị của hàm mục tiêu được dự đoán bởi gradient và giá trị thực tế.

Khi hàm số có nhiều biến đầu vào, sẽ có nhiều đạo hàm cấp hai. Các đạo hàm này có thể được tập hợp lại thành một ma trận gọi là **ma trận Hess**. Ma trận Hessian $\boldsymbol{H} = \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ được định nghĩa bởi

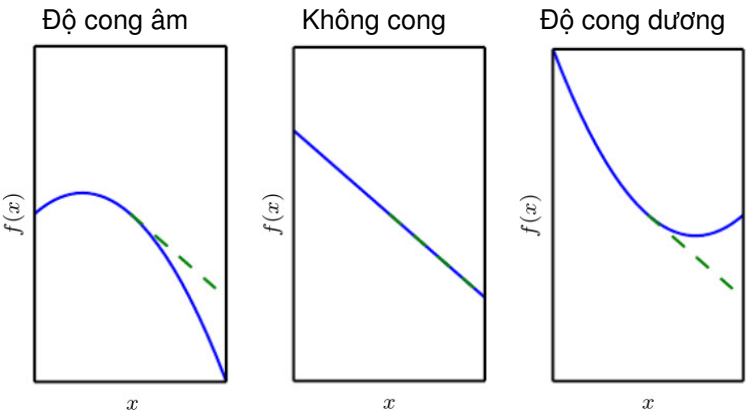
$$\boldsymbol{H}_{i,j} = \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) . \tag{4.6}$$

Nói cách khác, ma trận Hess là ma trận Jacobi của gradient.

Bất kỳ điểm nào mà các đạo hàm riêng cấp hai liên tục, các toán tử vi phân này là giao hoán, nghĩa là thứ tự của biến lấy đạo hàm có thể hoán đổi:

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}) . \tag{4.7}$$

Điều này ngụ ý $H_{i,j} = H_{j,i}$, do đó ma trận Hess đối xứng tại những điểm đó. Hầu hết các hàm mà ta gặp trong bối cảnh học sâu đều có ma trận Hess đối xứng hầu khắp nơi. Vì ma trận Hess là ma trận thực và đối xứng, ta có thể phân tích nó thành một tập các trị riêng thực và một cơ sở trực giao của các véctơ riêng. Đạo hàm cấp



Hình 4.4: Đạo hàm bậc hai xác định độ cong của một hàm. Ở đây, ta minh họa các hàm bậc hai với các độ cong khác nhau. Đường nét đứt biểu thị giá trị của hàm mục tiêu mà ta mong đợi dựa trên thông tin từ gradient khi thực hiện một bước giảm theo hướng gradient. Trong trường hợp độ cong âm, hàm mục tiêu thực tế giảm nhanh hơn dự đoán của gradient. Trong trường hợp không có độ cong, gradient dự đoán chính xác mức giảm. Trong trường hợp độ cong dương, hàm số giảm chậm hơn dự kiến và cuối cùng bắt đầu tăng lên, do đó các bước quá lớn có thể thực sự làm tăng hàm số một cách không mong muốn

hai theo một hướng cụ thể được biểu diễn bởi một véctơ đơn vị $\boldsymbol{d}$ được cho bởi $\boldsymbol{d}^T\boldsymbol{H}\boldsymbol{d}$. Khi $\boldsymbol{d}$ là một véctơ riêng của $\boldsymbol{H}$, đạo hàm cấp hai theo hướng đó được cho bởi trị riêng tương ứng. Với các hướng khác của $\boldsymbol{d}$, đạo hàm cấp hai theo hướng đó là trung bình trọng số của tất cả các trị riêng, với các trọng số từ 0 đến 1, và các véctơ riêng có góc với $\boldsymbol{d}$ nhỏ hơn sẽ nhận được trọng số lớn hơn. Giá trị riêng lớn nhất xác định đạo hàm cấp hai lớn nhất và giá trị riêng nhỏ nhất xác định đạo hàm cấp hai nhỏ nhất.

Đạo hàm cấp hai (theo hướng) cho ta biết mức độ hiệu quả của một bước trong phương pháp hướng giảm có thể đạt được. Ta có thể thực hiện xấp xỉ Taylor bậc hai cho hàm $f(x)$ quanh điểm hiện tại $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f\left(\boldsymbol{x}^{(0)}\right) + \left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \boldsymbol{g} + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \boldsymbol{H}\left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right) \tag{4.8}$$

trong đó $\boldsymbol{g}$ là gradient và $\boldsymbol{H}$ là ma trận Hess tại $\boldsymbol{x}^{(0)}$. Nếu ta sử dụng một tốc độ học ε, thì điểm mới $\boldsymbol{x}$ sẽ được cho bởi $\boldsymbol{x}^{(0)} - \varepsilon\boldsymbol{g}$. Thay vào xấp xỉ trên, ta được

$$f\left(\boldsymbol{x}^{(0)} - \varepsilon\boldsymbol{g}\right) \approx f\left(\boldsymbol{x}^{(0)}\right) + \varepsilon\boldsymbol{g}^T\boldsymbol{g} + \frac{1}{2}\varepsilon^2\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}. \tag{4.9}$$

Có ba thành phần ở đây: giá trị ban đầu của hàm, cải thiện kỳ vọng do độ dốc

của hàm, và sự điều chỉnh mà ta phải áp dụng để tính đến độ cong của hàm. Khi thành phần cuối cùng này quá lớn, bước của hướng giảm thực tế có thể di chuyển ngược lên phía trên. Khi $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ bằng 0 hoặc âm, xấp xỉ Taylor dự đoán rằng việc tăng ε mãi mãi sẽ làm giảm f mãi mãi. Trong thực tế, chuỗi Taylor khó có khả năng vẫn chính xác với ε lớn, vì vậy trong trường hợp này ta cần dùng các lựa chọn ε mang tính kinh nghiệm. Khi $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ dương, việc giải kích thước bước tối ưu làm giảm tối đa xấp xỉ Taylor của hàm một cách tối đa sẽ cho

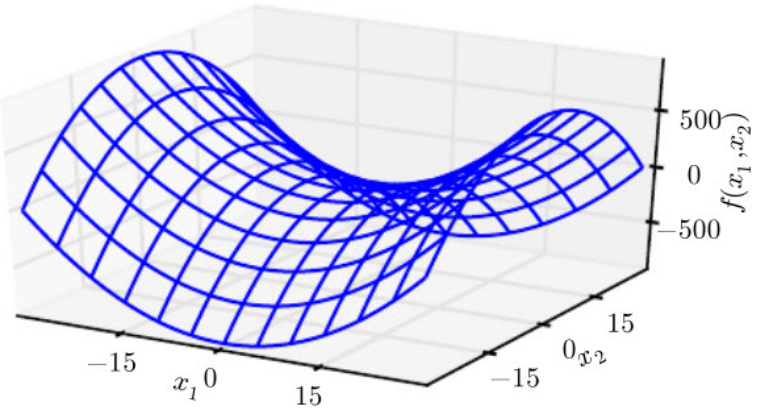
$$\varepsilon^* = \frac{\boldsymbol{g}^T\boldsymbol{g}}{\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}}. \tag{4.10}$$

Trong trường hợp xấu nhất, khi $\boldsymbol{g}$ thẳng hàng với véctơ riêng của $\boldsymbol{H}$ tương ứng với giá trị riêng lớn nhất $\lambda_{\max}$, thì kích thước bước tối ưu sẽ là $\frac{1}{\lambda_{\max}}$. Trong chừng mực mà hàm cần cực tiểu hóa có thể được xấp xỉ tốt bằng một hàm bậc hai, các giá trị riêng của ma trận Hess sẽ xác định mức độ của tốc độ học.

Đạo hàm cấp hai có thể được sử dụng để xác định liệu một điểm tới hạn là cực đại địa phương, cực tiểu địa phương, hay là điểm yên ngựa. Nhớ rằng tại một điểm tới hạn, $f'(x) = 0$. Khi đạo hàm cấp hai $f''(x) > 0$, đạo hàm cấp một $f'(x)$ tăng khi ta di chuyển sang phải và giảm khi ta di chuyển sang trái. Điều này có nghĩa là $f'(x - \varepsilon) < 0$ và $f'(x + \varepsilon) > 0$ với ε đủ nhỏ. Nói cách khác, khi di chuyển sang phải, độ dốc bắt đầu hướng lên phải, và khi di chuyển sang trái, độ dốc bắt đầu hướng lên trái. Do đó, khi $f'(x) = 0$ và $f''(x) > 0$, ta có thể kết luận x là một cực tiểu địa phương. Tương tự, khi $f'(x) = 0$ và $f''(x) < 0$, ta có thể kết luận x là một cực đại địa phương. Đây được gọi là **kiểm tra đạo hàm cấp hai**. Tuy nhiên, khi $f''(x) = 0$, phép kiểm tra không đưa ra kết luận. Trong trường hợp này, x có thể là một điểm yên ngựa hoặc là một phần của một vùng phẳng.

Trong trường hợp hàm số nhiều biến, ta cần xem xét tất cả các đạo hàm cấp hai của hàm. Bằng cách phân tích giá trị riêng cho ma trận Hess, ta có thể tổng quát hóa phép kiểm tra đạo hàm cấp hai cho hàm số nhiều chiều. Tại một điểm tới hạn, nơi $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{0}$, ta có thể xem xét các trị riêng của ma trận Hess để xác định liệu điểm tới hạn đó là cực đại địa phương, cực tiểu địa phương, hay điểm yên ngựa. Khi ma trận Hess xác định dương (tất cả các trị riêng của nó đều dương), điểm đó là cực tiểu địa phương. Điều này có thể thấy rõ khi đạo hàm cấp hai theo mọi hướng đều dương, tương tự như phép kiểm tra đạo hàm cấp hai trong trường hợp hàm số một biến. Tương tự, khi ma trận Hess là xác định âm (tất cả các trị riêng của nó đều âm), điểm đó là cực đại địa phương. Trong trường hợp hàm số nhiều biến, ta cũng có thể tìm thấy bằng chứng xác thực về điểm yên ngựa trong

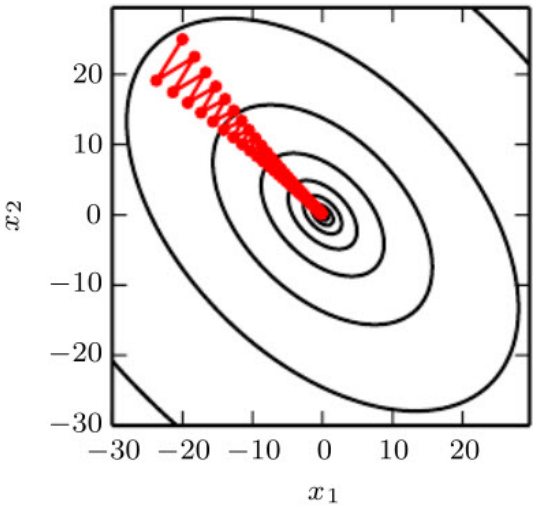
một số trường hợp. Khi ít nhất một trị riêng là dương và ít nhất một trị riêng là âm, ta biết rằng $\boldsymbol{x}$ là cực đại địa phương trên một mặt cắt của f nhưng là cực tiểu địa phương trên mặt cắt khác. Xem Hình 4.5 để minh họa. Cuối cùng, phép kiểm tra đạo hàm cấp hai trong trường hợp hàm số nhiều biến có thể không đưa ra được kết luận, giống như trong trường hợp hàm số một biến, khi tất cả các trị riêng khác không có cùng dấu, nhưng ít nhất một trị riêng bằng không. Điều này xảy ra vì phép kiểm tra đạo hàm cấp hai trong trường hợp hàm số một biến cũng không đưa ra kết luận trên mặt cắt tương ứng với giá trị riêng bằng không.



Hình 4.5: Một điểm yên ngựa có cả độ cong dương và độ cong âm. Hàm số trong ví dụ này là $f(\boldsymbol{x}) = x_1^2 - x_2^2$. Dọc theo trục tương ứng với x_1, hàm số có độ cong hướng lên. Trục này là một véctơ riêng của ma trận Hess và có trị riêng dương. Dọc theo trục tương ứng với x_2, hàm số có độ cong hướng xuống. Hướng này là một véctơ riêng của ma trận Hess với trị riêng âm. Tên gọi "điểm yên ngựa" xuất phát từ hình dạng giống yên ngựa của hàm số này. Đây là ví dụ tiêu biểu của một hàm có điểm yên ngựa. Trong trường hợp hàm số nhiều biến, không cần phải có giá trị riêng bằng 0 để xuất hiện điểm yên ngựa: chỉ cần có cả trị riêng dương và âm. Ta có thể hình dung điểm yên ngựa với cả hai dấu của các giá trị riêng như một cực đại địa phương trong một mặt cắt và cực tiểu địa phương trong một mặt cắt khác.

Trong trường hợp hàm số nhiều biến, tại một điểm, có các đạo hàm cấp hai khác nhau theo từng hướng. Số điều kiện của ma trận Hess tại điểm này đo lường mức độ khác nhau của các đạo hàm cấp hai này. Khi ma trận Hess có số điều kiện xấu, phương pháp hướng giảm hoạt động kém hiệu quả. Nguyên nhân là do ở một hướng, đạo hàm tăng nhanh, trong khi ở hướng khác, đạo hàm tăng chậm. Phương pháp hướng giảm không nhận biết được sự thay đổi này, vì vậy không biết rằng cần phải ưu tiên khám phá hướng mà đạo hàm vẫn âm trong thời gian lâu hơn. Điều

này cũng gây khó khăn trong việc chọn kích thước bước hợp lý. Kích thước bước phải đủ nhỏ để tránh vượt quá cực tiểu và đi lên ở các hướng có độ cong dương lớn. Điều này thường dẫn đến kích thước bước quá nhỏ để đạt tiến bộ đáng kể ở các hướng có độ cong thấp hơn. Xem Hình 4.6 để minh họa.



Hình 4.6: Phương pháp hướng giảm không tận dụng được thông tin về độ cong có trong ma trận Hess. Ở đây, ta dùng phương pháp hướng giảm để cực tiểu hóa một hàm bậc hai $f(\boldsymbol{x})$ có ma trận Hess với số điều kiện là 5. Điều này có nghĩa là hướng có độ cong lớn nhất có độ cong gấp năm lần so với hướng có độ cong nhỏ nhất. Trong trường hợp này, độ cong lớn nhất là theo hướng $(1, 1)^T$, và độ cong nhỏ nhất là theo hướng $(1, -1)^T$. Các đường màu đỏ cho thấy đường đi mà phương pháp hướng giảm đã theo. Hàm bậc hai rất kéo dài này giống như một hẻm núi dài. Phương pháp hướng giảm mất thời gian liên tục đi xuống các vách hẻm núi, vì đó là đặc điểm dốc nhất. Do kích thước bước đi hơi lớn, phương pháp hướng giảm có xu hướng vượt quá đáy hàm và do đó cần phải đi xuống vách núi đối diện trong lần lặp tiếp theo. Giá trị riêng dương lớn của ma trận Hess tương ứng với véctơ riêng chỉ theo hướng này cho thấy rằng đạo hàm theo hướng này đang tăng nhanh, vì vậy một thuật toán tối ưu dựa trên ma trận Hess có thể dự đoán rằng hướng dốc nhất không phải là một hướng tìm kiếm triển vọng trong ngữ cảnh này.

Vấn đề này có thể được giải quyết bằng cách sử dụng thông tin từ ma trận Hess để hướng dẫn quá trình tìm kiếm. Phương pháp đơn giản nhất để làm điều này được gọi là **phương pháp Newton**. Phương pháp Newton dựa trên việc sử

dụng khai triển Taylor bậc hai để xấp xỉ $f(\boldsymbol{x})$ gần một điểm $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f\left(\boldsymbol{x}^{(0)}\right) + \left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right) + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \left[\nabla_{\boldsymbol{x}}^2 f\left(\boldsymbol{x}^{(0)}\right)\right] \left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right). \tag{4.11}$$

Nếu sau đó ta giải tìm điểm dừng của hàm này, ta thu được:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \left[\nabla_{\boldsymbol{x}}^2 f\left(\boldsymbol{x}^{(0)}\right)\right]^{-1} \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right). \tag{4.12}$$

Khi f là một hàm bậc hai xác định dương, phương pháp Newton bao gồm việc áp dụng công thức (4.12) một lần để nhảy trực tiếp đến điểm cực tiểu của hàm. Khi f không thực sự là hàm bậc hai nhưng có thể xấp xỉ địa phương bởi một hàm bậc hai xác định dương, phương pháp Newton bao gồm việc áp dụng công thức (4.12) nhiều lần. Việc cập nhật xấp xỉ và nhảy đến điểm cực tiểu của xấp xỉ này lặp đi lặp lại có thể giúp tiếp cận điểm tới hạn nhanh hơn nhiều so với phương pháp hướng giảm. Điều này là một tính chất hữu ích gần điểm cực tiểu địa phương, nhưng có thể trở nên bất lợi gần điểm yên ngựa. Như thảo luận trong Mục 9.1.1, phương pháp Newton chỉ phù hợp khi điểm tới hạn gần đó là một điểm cực tiểu (tất cả các giá trị riêng của ma trận Hess là dương), trong khi phương pháp hướng giảm không bị hấp dẫn bởi các điểm yên ngựa trừ khi gradient hướng về phía chúng.

Các **thuật toán tối ưu cấp một** như phương pháp hướng giảm chỉ dựa vào gradient của hàm mục tiêu để điều chỉnh các tham số theo hướng giảm mất mát. Do chỉ sử dụng gradient, các thuật toán này có thể không hiệu quả khi đối mặt với những vùng có độ dốc thấp hoặc các điểm yên ngựa. Trong khi đó, **thuật toán tối ưu hóa cấp hai**, như phương pháp Newton, tận dụng thêm ma trận Hess để ước tính độ cong của hàm mục tiêu và điều chỉnh hướng đi sao cho phù hợp hơn với địa hình của hàm (*Numerical Optimization*, Nocedal và Wright, 2006, [5]). Do sử dụng thêm thông tin từ ma trận Hess, các thuật toán này có thể hội tụ nhanh hơn khi ở gần cực trị, nhưng cũng yêu cầu tính toán nặng nề hơn, đặc biệt khi ma trận Hess rất lớn.

Tuy nhiên, trong học sâu, việc sử dụng các thuật toán tối ưu hóa phức tạp và yêu cầu nhiều tính toán này thường không khả thi. Lý do là các hàm mục tiêu trong học sâu rất phức tạp, đa chiều và phi tuyến, dẫn đến việc thiếu các đảm bảo về hiệu quả hội tụ. Ngược lại, các lĩnh vực khác thường thiết kế các thuật toán tối ưu hóa cho một họ hàm mục tiêu nhất định để đạt hiệu quả cao hơn và có được những đảm bảo về hội tụ, điều mà học sâu hiếm khi đạt được.

Trong bối cảnh học sâu, đôi khi ta có thể đảm bảo một số đặc tính nhất định bằng cách tự giới hạn các hàm là hàm **liên tục Lipschitz** hoặc có đạo hàm liên tục

Lipschitz. Một hàm liên tục Lipschitz là một hàm f có tốc độ thay đổi bị giới hạn bởi một **hằng số Lipschitz** $\mathcal{L}$:

$$\forall \boldsymbol{x}, \quad \forall \boldsymbol{y}, \quad |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L} \|\boldsymbol{x} - \boldsymbol{y}\|_2. \tag{4.13}$$

Tính chất này rất hữu ích vì nó cho phép ta định lượng giả định rằng một thay đổi nhỏ trong đầu vào do một thuật toán, như phương pháp hướng giảm, tạo ra sẽ chỉ dẫn đến một thay đổi nhỏ trong đầu ra. Tính liên tục theo Lipschitz cũng là một ràng buộc khá yếu, và nhiều bài toán tối ưu hóa trong học sâu có thể được làm liên tục Lipschitz với những sửa đổi tương đối nhỏ.

Có lẽ lĩnh vực tối ưu hóa chuyên sâu thành công nhất là **tối ưu lồi**. Các thuật toán tối ưu hóa lồi có thể cung cấp nhiều đảm bảo hơn nhờ vào các hạn chế chặt chẽ hơn. Các thuật toán này chỉ áp dụng cho các hàm lồi — tức là các hàm có ma trận Hess là nửa xác định dương ở khắp nơi. Những hàm này có đặc tính tốt vì không có điểm yên ngựa và tất cả các cực tiểu địa phương của chúng đều là cực tiểu toàn cục. Tuy nhiên, hầu hết các bài toán trong học sâu khó có thể biểu diễn bằng tối ưu lồi. Tối ưu lồi chỉ được sử dụng như một quy trình phụ trong một số thuật toán học sâu. Các ý tưởng từ giải tích thuật toán tối ưu lồi có thể hữu ích để chứng minh sự hội tụ của các thuật toán học sâu. Tuy nhiên, nhìn chung, tầm quan trọng của tối ưu lồi bị giảm đi nhiều trong bối cảnh học sâu. Để tìm hiểu thêm về tối ưu hóa lồi, xem *Convex Optimization* (Boyd và Vandenberghe, 2004, [6]) hoặc *Convex Analysis* (Rockafellar, 1997, [7]).

4.4 Tối ưu có ràng buộc

Đôi khi ta không chỉ muốn cực đại hóa hoặc cực tiểu hóa một hàm $f(\boldsymbol{x})$ với tất cả các giá trị có thể của $\boldsymbol{x}$. Thay vào đó, có thể ta muốn tìm giá trị lớn nhất hoặc nhỏ nhất của $f(\boldsymbol{x})$ với các giá trị của $\boldsymbol{x}$ trong một tập hợp S nào đó. Bài toán này được gọi là **tối ưu có ràng buộc**. Các điểm $\boldsymbol{x}$ nằm trong tập hợp S được gọi là các điểm **khả thi**.

Ta thường muốn tìm một nghiệm nhỏ theo một nghĩa nào đó. Một cách tiếp cận phổ biến trong các tình huống này là áp dụng ràng buộc chuẩn, chẳng hạn $\|\boldsymbol{x}\| \leq 1$.

Một cách đơn giản để tối ưu hóa có ràng buộc là chỉ cần điều chỉnh phương pháp hướng giảm để tính đến ràng buộc. Nếu sử dụng kích thước bước nhỏ ε không đổi, ta có thể thực hiện các bước theo hướng giảm và sau đó chiếu kết quả

trở lại S. Nếu sử dụng tìm kiếm trên đường thẳng, ta chỉ có thể tìm kiếm các kích thước bước ε mà các điểm $\boldsymbol{x}$ mới là khả thi hoặc chiếu từng điểm trên đường đó trở lại miền ràng buộc. Khi có thể, phương pháp này có thể được thực hiện hiệu quả hơn bằng cách chiếu gradient lên không gian tiếp tuyến của miền khả thi trước khi thực hiện bước đi hoặc bắt đầu tìm kiếm trên đường thẳng (*The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints*, Rosen, 1960, [8]).

Một cách tiếp cận tinh vi hơn là thiết kế một bài toán tối ưu không ràng buộc khác, trong đó nghiệm của bài toán này có thể được chuyển đổi thành nghiệm của bài toán tối ưu có ràng buộc ban đầu. Chẳng hạn, nếu ta muốn cực tiểu hóa $f(\boldsymbol{x})$ với $\boldsymbol{x} \in \mathbb{R}^2$ bị ràng buộc có chuẩn L^2 đúng bằng 1, thay vào đó ta có thể cực tiểu hóa $g\left(\theta\right) = f\left(\left[\cos\theta, \sin\theta\right]^T\right)$ theo biến θ, sau đó trả về $\left[\cos\theta, \sin\theta\right]$ như nghiệm của bài toán ban đầu. Cách tiếp cận này đòi hỏi sự sáng tạo; phép biến đổi giữa các bài toán tối ưu cần được thiết kế riêng biệt cho từng trường hợp ta gặp phải.

Phương pháp Karush – Kuhn – Tucker (KKT)* cung cấp một giải pháp rất tổng quát cho bài toán tối ưu có ràng buộc. Với phương pháp KKT, ta xây dựng một hàm mới gọi là **hàm Lagrange suy rộng**.

Để xác định hàm Lagrange, trước tiên ta cần mô tả tập S dưới dạng các phương trình và bất phương trình. Ta muốn biểu diễn S thông qua m hàm $g^{(i)}$ và n hàm $h^{(j)}$ sao cho $S = \left\{x \mid \forall i,\ g^{(i)}(x) = 0 \text{ và } \forall j,\ h^{(j)}(x) \leq 0\right\}$. Các phương trình liên quan đến $g^{(i)}$ được gọi là ràng buộc đẳng thức, và các bất phương trình liên quan đến $h^{(j)}$ được gọi là ràng buộc bất đẳng thức.

Ta đặt các biến mới λ_i và α_j cho mỗi ràng buộc, được gọi là các nhân tử KKT. Hàm Lagrange suy rộng được định nghĩa bởi

$$L\left(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}\right) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.14}$$

Bây giờ, ta có thể giải bài toán cực tiểu hóa có ràng buộc bằng cách tối ưu không ràng buộc hàm Lagrange suy rộng. Lưu ý rằng, miễn là tồn tại ít nhất một điểm khả thi và $f(x)$ không có giá trị vô cùng, thì bài toán

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} L\left(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}\right) \tag{4.15}$$

*Phương pháp KKT mở rộng phương pháp **nhân tử Lagrange**, phương pháp mà như đã biết cho phép áp dụng với các ràng buộc đẳng thức nhưng không áp dụng được với các ràng buộc bất đẳng thức.

có cùng giá trị hàm mục tiêu tối ưu và tập hợp các điểm tối ưu $\boldsymbol{x}$ với bài toán

$$\min_{\boldsymbol{x} \in S} f(\boldsymbol{x}). \tag{4.16}$$

Điều này đúng vì bất kỳ lúc nào các ràng buộc được thỏa mãn, thì

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} L\left(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}\right) = f(\boldsymbol{x}), \tag{4.17}$$

trong khi bất kỳ lúc nào một ràng buộc bị vi phạm, thì

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} L\left(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}\right) = \infty. \tag{4.18}$$

Các tính chất này đảm bảo rằng không có điểm không khả thi nào có thể là điểm tối ưu, và điểm tối ưu trong tập các điểm khả thi không thay đổi.

Để thực hiện việc cực đại có ràng buộc, ta có thể xây dựng hàm Lagrange suy rộng cho $-f(\boldsymbol{x})$, dẫn đến bài toán tối ưu:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.19}$$

Ta cũng có thể chuyển bài toán này thành bài toán với cực đại ở khâu cuối cùng:

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.20}$$

Dấu của các hạng tử trong ràng buộc đẳng thức không quan trọng; ta có thể định nghĩa chúng với dấu cộng hoặc dấu trừ tùy ý, vì bài toán tối ưu hóa cho phép lựa chọn bất kỳ dấu nào cho mỗi λ_i.

Các ràng buộc bất đẳng thức đặc biệt thú vị. Ta nói rằng một ràng buộc $h^{(i)}(\boldsymbol{x})$ là **hoạt động** nếu $h^{(i)}(\boldsymbol{x}^*) = 0$. Nếu một ràng buộc không hoạt động, thì nghiệm của bài toán tìm được khi sử dụng ràng buộc đó sẽ vẫn là một nghiệm (ít nhất là nghiệm cục bộ) nếu ràng buộc đó bị loại bỏ. Tuy nhiên, có khả năng một ràng buộc không hoạt động có thể loại trừ các nghiệm khác. Ví dụ, một bài toán tối ưu lồi có một miền các nghiệm tối ưu toàn cục (miền rộng và phẳng với giá trị chi phí bằng nhau) có thể có một phần của miền này bị loại trừ bởi các ràng buộc, hoặc một bài toán không lồi có thể có các điểm dừng địa phương tốt hơn bị loại trừ bởi một ràng buộc không hoạt động tại điểm hội tụ. Tuy nhiên, điểm tìm được tại điểm hội tụ vẫn là một điểm dừng bất kể có hay không các ràng buộc không hoạt động. Do một $h^{(i)}$ không hoạt động có giá trị âm, nghiệm của $\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}:\, \boldsymbol{\alpha} \geq \mathbf{0}} L\left(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}\right)$ sẽ có $\alpha_i = 0$. Do đó, ta có thể quan sát rằng tại nghiệm, $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}$. Nói cách

khác, với mọi i, ta biết rằng ít nhất một trong các ràng buộc $\alpha_i \geq 0$ và $h^{(i)}(\boldsymbol{x}) \leq 0$ phải hoạt động tại nghiệm. Để có trực giác về ý tưởng này, ta có thể nói rằng hoặc là nghiệm nằm trên biên do bất đẳng thức áp đặt và ta phải sử dụng nhân tử KKT của bất đẳng thức đó để ảnh hưởng đến nghiệm $\boldsymbol{x}$, hoặc bất đẳng thức không ảnh hưởng đến nghiệm và ta biểu diễn điều này bằng cách làm cho nhân tử KKT của nó bằng không.

Một tập các tính chất đơn giản mô tả các điểm tối ưu của các bài toán tối ưu có ràng buộc. Các tính chất này được gọi là các điều kiện Karush – Kuhn – Tucker (KKT) (*Minima of Functions of Several Variables with Inequalities as Side Conditions*, Karush, 1939, [9]; *Nonlinear Programming*, Kuhn và Tucker, 1951, [22]). Đây là các điều kiện cần, nhưng không phải lúc nào cũng là các điều kiện đủ, để một điểm trở thành điểm tối ưu. Các điều kiện này bao gồm:

- Gradient của hàm Lagrange suy rộng bằng không.
- Tất cả các ràng buộc cho cả $\boldsymbol{x}$ và các nhân tử KKT đều thỏa mãn.
- Các ràng buộc bất đẳng thức thể hiện "tính bù yếu": $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$. Để biết thêm thông tin về phương pháp KKT, xem *Numerical Optimization*, (Nocedal và Wright, 2006, [5]).

4.5 Ví dụ: phương pháp bình phương tối thiểu tuyến tính

Giả sử ta muốn tìm giá trị của $\boldsymbol{x}$ làm cực tiểu

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2. \tag{4.21}$$

Có những thuật toán chuyên biệt về đại số tuyến tính có thể giải quyết bài toán này một cách hiệu quả. Tuy nhiên, ta cũng có thể khám phá cách giải quyết nó bằng phương pháp tối ưu dựa trên gradient như một ví dụ đơn giản về cách hoạt động của các kỹ thuật này.

Đầu tiên, ta cần tính gradient:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{b}. \tag{4.22}$$

Sau đó, ta có thể đi theo hướng xuống dốc bằng cách thực hiện các bước nhỏ. Xem Thuật toán 1 để biết thêm chi tiết.

Ta cũng có thể giải bài toán này bằng phương pháp Newton. Trong trường hợp này, vì hàm $f(\boldsymbol{x})$ thực sự là hàm bậc hai, nên xấp xỉ bậc hai được sử dụng trong

Thuật toán 1: Một thuật toán để cực tiểu hóa $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ theo $\boldsymbol{x}$ bằng phương pháp hướng giảm, xuất phát từ một giá trị tuỳ ý $\boldsymbol{x}^{(0)}$ của $\boldsymbol{x}$.

Đầu vào:

- Hàm cần cực tiểu hóa $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$, xấp xỉ ban đầu $\boldsymbol{x}^{(0)}$
- Kích thước bước ε, ngưỡng sai số δ, số bước lặp tối đa N

Đầu ra: Xấp xỉ nghiệm tối ưu $\boldsymbol{x}^*$ của bài toán $\arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$

```
k ← 0;
x ← x^(0);
while true do
    k ← k + 1;
    x ← x − ε(A^T Ax − A^T b);
    if ||A^T Ax − A^T b||_2 < δ ∨ k = N then
        break;
    end
end
```

phương pháp Newton là chính nó, và thuật toán sẽ hội tụ đến cực tiểu toàn cục chỉ trong một bước. Cụ thể, từ công thức (4.22) ta có thể tính được ma trận Hess của $f(\boldsymbol{x})$ là $\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x}) = \boldsymbol{A}^T\boldsymbol{A}$. Xuất phát từ điểm $\boldsymbol{x}$ bất kỳ, nghiệm tối ưu chính xác của bài toán xác định bởi

$$\boldsymbol{x}^* = \boldsymbol{x} - \left[\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})\right]^{-1}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b}. \tag{4.23}$$

Bây giờ, giả sử ta muốn cực tiểu hóa vẫn hàm đó, nhưng với điều kiện $\boldsymbol{x}^T\boldsymbol{x} \leq 1$. Để làm điều này, ta xây dựng hàm Lagrange suy rộng:

$$L\left(\boldsymbol{x}, \lambda\right) = f(\boldsymbol{x}) + \lambda\left(\boldsymbol{x}^T\boldsymbol{x} - 1\right). \tag{4.24}$$

Ta có thể giải bài toán này bằng cách giải bài toán thay thế, là cực tiểu hóa theo $\boldsymbol{x}$ của cực đại theo λ với điều kiện $\lambda \geq 0$:

$$\min_{\boldsymbol{x}} \max_{\lambda:\, \lambda \geq 0} L\left(\boldsymbol{x}, \lambda\right). \tag{4.25}$$

Điều này giúp đưa bài toán tối ưu có ràng buộc về bài toán tối ưu không ràng buộc của hàm Lagrange suy rộng, và sử dụng các nhân tử Lagrange để kiểm soát điều kiện ràng buộc $\boldsymbol{x}^T\boldsymbol{x} \leq 1$.

Nghiệm có chuẩn nhỏ nhất của bài toán bình phương tối thiểu không ràng buộc có thể được tìm bằng cách sử dụng giả nghịch đảo Moore – Penrose: $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b}$. Nếu điểm này thỏa mãn điều kiện khả thi, thì nó là nghiệm của bài toán có ràng buộc. Nếu không, chúng ta phải tìm một nghiệm mà ràng buộc hoạt động. Bằng cách lấy đạo hàm của hàm Lagrange theo $\boldsymbol{x}$, ta thu được phương trình

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{b} + 2\lambda\boldsymbol{x} = \boldsymbol{0}. \tag{4.26}$$

Điều này cho ta biết nghiệm sẽ có dạng

$$\boldsymbol{x} = \left(\boldsymbol{A}^T\boldsymbol{A} + 2\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b}. \tag{4.27}$$

Độ lớn của λ phải được chọn sao cho nghiệm thỏa mãn ràng buộc $\boldsymbol{x}^T\boldsymbol{x} \leq 1$. Để tìm giá trị này, ta có thể thực hiện phương pháp hướng tăng theo λ. Để làm điều này, ta quan sát

$$\frac{\partial}{\partial\lambda}L\left(\boldsymbol{x}, \lambda\right) = \boldsymbol{x}^T\boldsymbol{x} - 1. \tag{4.28}$$

Khi chuẩn của $\boldsymbol{x}$ vượt quá 1, đạo hàm này sẽ dương, cho nên để tăng giá trị của hàm Lagrange theo λ, ta tăng λ. Vì hệ số của hạng tử phạt $\boldsymbol{x}^T\boldsymbol{x}$ đã tăng lên, nên khi giải phương trình tuyến tính theo $\boldsymbol{x}$, ta sẽ thu được một nghiệm có chuẩn nhỏ hơn. Quá trình giải phương trình này và điều chỉnh λ liên tục cho đến khi $\boldsymbol{x}$ có chuẩn đúng bằng 1, và đạo hàm của hàm Lagrange theo λ bằng 0.

Chúng ta kết thúc các kiến thức cơ bản về toán học sử dụng để phát triển các thuật toán học máy tại đây. Giờ đây, chúng ta đã sẵn sàng để xây dựng và phân tích một số hệ thống học máy hoàn chỉnh.

Chương 8

Thuật toán phân loại tuyến tính

Chương này bắt đầu bằng việc phân tích các bài toán phân loại tuyến tính, tập trung đặc biệt vào hồi quy logistic (mặc dù tên gọi của nó, đây là một thuật toán phân loại) và phương pháp **hướng giảm ngẫu nhiên** (stochastic gradient descent – SGD). Dù các chiến lược này có vẻ đơn giản, chúng vẫn là những lựa chọn chính trong nhiều nhiệm vụ phân loại.

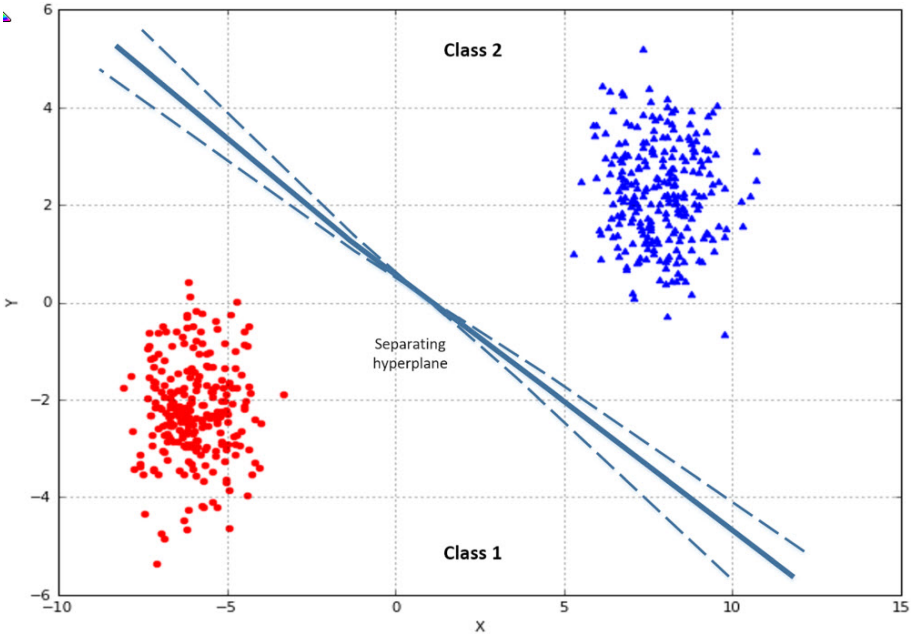
Nhân tiện, cần nhớ đến một nguyên tắc triết học rất quan trọng: **Lưỡi dao Occam**.

Trong bối cảnh của chúng ta, nguyên tắc này nói rằng lựa chọn đầu tiên phải luôn là phương pháp đơn giản nhất, và chỉ khi nó không phù hợp thì mới cần chuyển sang các mô hình phức tạp hơn. Trong phần thứ hai của chương, chúng ta sẽ thảo luận về một số thước đo phổ biến hữu ích khi đánh giá một nhiệm vụ phân loại. Chúng không chỉ giới hạn trong các mô hình tuyến tính, vì vậy ta sẽ sử dụng chúng khi nói về các chiến lược khác nhau. Cụ thể, chúng ta sẽ thảo luận về:

- Cấu trúc tổng quát của một bài toán phân loại tuyến tính
- Hồi quy logistic (có và không có điều chỉnh)
- Thuật toán SGD và perceptron
- Thuật toán passive – aggressive
- Tìm kiếm các siêu tham số tối ưu trên lưới
- Các thước đo phân loại quan trọng nhất
- Đường cong ROC (**Receiver Operating Characteristic**)

8.1 Phân loại tuyến tính

Hãy xem xét một bài toán phân loại tuyến tính tổng quát với hai lớp. Ví dụ trong biểu đồ sau:



Hình 8.1: Kịch bản hai chiều của một bài toán phân loại tuyến tính

Mục tiêu của ta là tìm một siêu phẳng tối ưu, phân tách hai lớp. Trong các bài toán phân loại nhiều lớp, chiến lược một chống tất cả (one-vs-all) thường được áp dụng, vì vậy ta có thể chỉ tập trung vào phân loại nhị phân. Giả sử ta có tập dữ liệu gồm m mẫu, mỗi mẫu có n chiều:

$$\boldsymbol{X} = \left(\boldsymbol{x}^{(1)^T}, \boldsymbol{x}^{(2)^T}, \dots, \boldsymbol{x}^{(m)^T}\right)^T, \quad \boldsymbol{x}^{(i)} \in \mathbb{R}^n. \tag{8.1}$$

Tập dữ liệu này được liên kết với tập mục tiêu:

$$\boldsymbol{Y} = (y_1, y_2, \dots, y_m)^T \quad \text{trong đó} \quad y_i \in \{0, 1\} \quad \text{hoặc} \quad y_i \in \{-1, 1\}. \tag{8.2}$$

Thông thường, có hai lựa chọn tương đương: đầu ra nhị phân và đầu ra lưỡng cực, và các thuật toán khác nhau dựa trên một trong hai mà không có sự khác biệt đáng kể. Thông thường, lựa chọn này được thực hiện để đơn giản hóa việc tính toán và không ảnh hưởng đến kết quả. Bây giờ, ta có thể định nghĩa một véctơ trọng số bao gồm n thành phần:

$$\boldsymbol{\theta} = \left(\theta_1, \theta_2, \dots, \theta_n\right)^T \quad \text{trong đó} \quad \theta_i \in \mathbb{R}. \tag{8.3}$$

Ta cũng có thể định nghĩa đại lượng z:

$$z = \boldsymbol{\theta}^T \boldsymbol{x} = \sum_{i=1}^{n} \theta_i x_i \quad \forall \boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n. \tag{8.4}$$

Nếu $\boldsymbol{x}$ là một biến, thì z là giá trị được xác định bởi phương trình siêu phẳng. Do đó, trong kịch bản lưỡng cực, nếu tập hợp các hệ số $\boldsymbol{\theta}$ đã được xác định là chính xác, thì điều sau sẽ xảy ra:

$$y = \text{sign}(z) = \begin{cases} +1 & \text{nếu } \boldsymbol{x} \in \text{ Class 1} \\ -1 & \text{nếu } \boldsymbol{x} \in \text{ Class 2.} \end{cases} \tag{8.5}$$

Khi làm việc với các đầu ra nhị phân, quyết định thường được đưa ra dựa trên một ngưỡng. Ví dụ, nếu đầu ra $z \in (0, 1)$, điều kiện trên trở thành:

$$y = \begin{cases} 1 & \text{nếu } z \geq 0.5 \\ 0 & \text{nếu } z < 0.5. \end{cases} \tag{8.6}$$

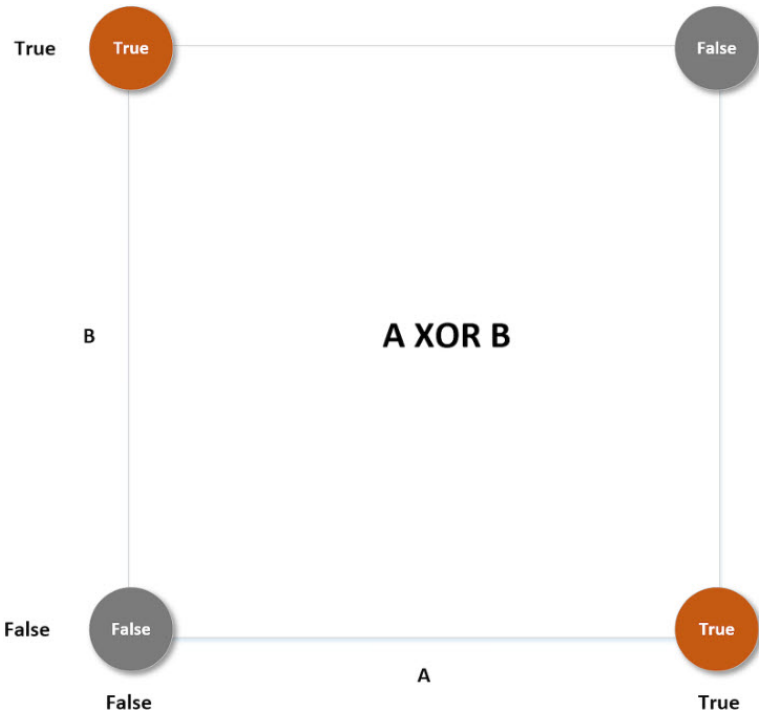
Bây giờ, ta phải tìm cách tối ưu hóa $\boldsymbol{\theta}$ để giảm sai số phân loại. Nếu tồn tại một sự kết hợp như vậy (với một ngưỡng sai số nhất định), ta nói rằng bài toán là có thể phân tách tuyến tính. Ngược lại, khi không thể tìm thấy một bộ phân loại tuyến tính, bài toán được định nghĩa là không thể phân tách tuyến tính.

Một ví dụ rất đơn giản nhưng nổi tiếng thuộc vào lớp thứ hai là phép toán logic XOR, như mô tả trên Hình 8.2:

Ta có thể thấy, bất kỳ đường thẳng nào cũng sẽ luôn bao gồm một mẫu sai. Do đó, để giải quyết bài toán này, cần phải áp dụng các kỹ thuật phi tuyến liên quan đến các đường cong bậc cao (ví dụ, hai đường parabol). Tuy nhiên, trong nhiều trường hợp thực tế, ta có thể sử dụng các kỹ thuật tuyến tính (thường đơn giản và nhanh hơn) cho các bài toán phi tuyến, với điều kiện chấp nhận một mức sai số phân loại có thể chấp nhận được.

8.2 Hồi quy logistic

Mặc dù được gọi là hồi quy, đây thực chất là một phương pháp phân loại dựa trên xác suất của một mẫu thuộc về một lớp. Vì xác suất phải liên tục trong $\mathbb{R}$ và bị giới hạn trong khoảng $(0, 1)$, nên cần phải xây dựng một hàm ngưỡng để lọc đại lượng z. Tương tự như đã thực hiện với hồi quy tuyến tính, ta có thể loại bỏ tham



Hình 8.2: Sơ đồ biểu bài toán không thể phân tách tuyến tính của phép toán nhị phân XOR: xét một tập hợp các điểm dữ liệu với hai đầu ra nhị phân (0 và 1) trong mặt phẳng hai chiều, nơi mà không có đường thẳng nào có thể phân tách chính xác hai nhóm điểm này. Cụ thể: các điểm đầu vào với giá trị XOR bằng 0 (00 hoặc 11) nằm ở hai góc chéo của sơ đồ và thường được biểu diễn bằng cùng một ký hiệu, chẳng hạn như hình tròn màu xám; trong khi các điểm đầu vào với giá trị XOR bằng 1 (01 hoặc 10) nằm ở hai góc chéo còn lại và thường được biểu diễn bằng ký hiệu khác, chẳng hạn như hình tròn màu cam. Không có đường thẳng nào có thể phân chia các điểm 0 và 1 mà không tạo ra sai số, do đó, bài toán XOR là ví dụ điển hình của bài toán không thể phân tách tuyến tính.

số bổ sung tương ứng với hệ số tự do bằng cách thêm một phần tử 1 vào trước mỗi véctơ đầu vào:

$$\boldsymbol{x} \Rightarrow \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} \tag{8.7}$$

Bằng cách này, ta có thể xét một véctơ tham số tương ứng $\boldsymbol{\theta}$ mới, chứa $m + 1$ phần tử, bao gồm cả hệ số tự do θ_0:

$$\boldsymbol{\theta} \Rightarrow \begin{bmatrix} \theta_0 \\ \boldsymbol{\theta} \end{bmatrix} \tag{8.8}$$

và tính giá trị z như sau:

$$z = \boldsymbol{\theta}^T \boldsymbol{x}. \tag{8.9}$$

Bây giờ, ta giả sử $p(\boldsymbol{x})$ là xác suất để phần tử $\boldsymbol{x}$ thuộc về lớp 1. Rõ ràng, phần tử đó thuộc về lớp 0 với xác suất $1 - p(\boldsymbol{x})$. Hồi quy logistic chủ yếu dựa trên ý tưởng mô hình hóa tỷ lệ chấp nhận odds của việc thuộc về lớp 1 bằng cách sử dụng hàm mũ:

$$\text{odds}(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})} = e^{\theta^T \boldsymbol{x}} = e^z \tag{8.10}$$

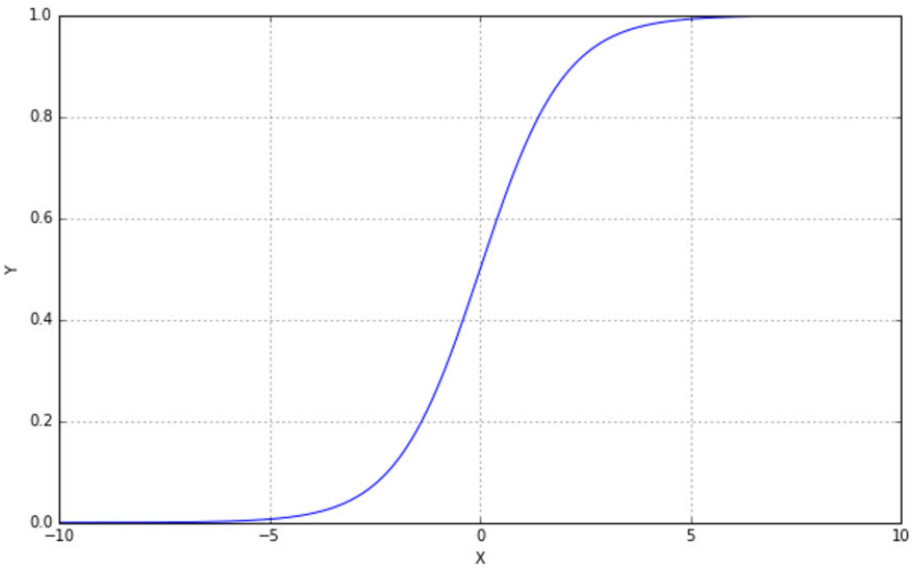
Hàm này liên tục và khả vi trên $\mathbb{R}$, luôn dương và có tiến tới vô hạn khi đối số $x \to \infty$. Những điều kiện này là cần thiết để mô hình hóa đúng tỷ lệ odds, vì khi $p \to 0$, odds cũng sẽ tiến tới 0, nhưng khi $p \to 1$, odds sẽ tiến tới vô hạn. Nếu ta lấy logit (là logarit tự nhiên) của odds, ta có được biểu thức cho cấu trúc của z:

$$z = \ln \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}. \tag{8.11}$$

Một hàm có miền xác định trên $\mathbb{R}$, bị giới hạn trong khoảng từ 0 đến 1, và thỏa mãn biểu thức trước đó là hàm logistic sigmoid:

$$p(\boldsymbol{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}. \tag{8.12}$$

Một phần đồ thị của hàm sigmoid được mô tả trên Hình 8.3.



Hình 8.3: Đồ thị hàm sigmoid; $\sigma(x) \to 0$ khi $x \to -\infty$ và $\sigma(x) \to 1$ khi $x \to \infty$

Như có thể thấy, hàm số cắt trục tung $x = 0$ ở điểm $y = 0.5$, với $y < 0.5$ khi $x < 0$ và $y > 0.5$ khi $x > 0$. Hơn nữa, ta có thể tính toán hàm này cho mọi giá

trị x và nó gần như tuyến tính quanh điểm $x = 0$. Điều này rất hữu ích khi sử dụng phép co rút, bởi vì hàm số trở nên nhạy hơn khi x gần bằng 0.

Vậy, ta có thể định nghĩa xác suất để một mẫu $\boldsymbol{x}$ thuộc về lớp c (từ giờ ta sẽ gọi chúng là lớp 0 và 1) như sau:

$$p\left(c \mid \boldsymbol{x}; \boldsymbol{\theta}\right) = P\left(y = c \mid \boldsymbol{x}; \boldsymbol{\theta}\right) = \begin{cases} \sigma\left(z\right) & \text{nếu } c = 1 \\ 1 - \sigma\left(z\right) & \text{nếu } c = 0 \end{cases} \tag{8.13}$$

Lúc này, việc tìm các tham số tối ưu tương đương với việc cực đại hóa logit của hàm hợp lý dựa trên lớp đầu ra mục tiêu cho trước.

$$L\left(\boldsymbol{\theta}; \boldsymbol{Y}, \boldsymbol{X}\right) = \ln \prod_{i=1}^{m} p\left(y_i \mid \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) = \sum_{i=1}^{m} \ln p\left(y_i \mid \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right). \tag{8.14}$$

Mỗi sự kiện dựa trên một phân phối Bernoulli; do đó, bằng cách sử dụng ký hiệu chỉ, bài toán tối ưu có thể được biểu diễn như là việc cực tiểu hóa hàm mất mát:

$$\begin{aligned} L &= -L\left(\boldsymbol{\theta}; \boldsymbol{Y}, \boldsymbol{X}\right) = -\sum_{i=1}^{m} \ln p\left(y_i \mid \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) \\ &= -\sum_{i=1}^{m} \left[y_i \ln \sigma\left(z_i\right) + \left(1 - y_i\right) \ln\left(1 - \sigma\left(z_i\right)\right)\right] \end{aligned} \tag{8.15}$$

Nếu $y_i = 0$, thì hạng tử đầu tiên của số hạng trở thành không, và hạng tử thứ hai là $\ln\left(1 - \sigma\left(z_i\right)\right)$, đại diện cho logit của xác suất của lớp 0. Ngược lại, nếu $y_i = 1$, thì hạng tử thứ hai bằng 0 và hạng tử đầu tiên biểu diễn logit của xác suất của lớp 1 là $\ln\left(\sigma\left(z_i\right)\right)$. Theo cách này, cả hai trường hợp, mỗi số hạng được gói gọn trong một hạng tử duy nhất. Tối ưu hóa bài toán này có thể đạt được bằng cách tính gradient ∇L theo các tham số, và đặt nó bằng $\mathbf{0}$. Toàn bộ quá trình lấy đạo hàm không quá phức tạp, nó chỉ yêu cầu một số thao tác chi tiết; do đó được để lại như một bài tập cho người đọc.

Về mặt lý thuyết thông tin, việc tối ưu hóa này tương đương với việc cực tiểu hóa cross−entropy giữa phân phối mục tiêu $p\left(x\right)$ và phân phối ước lượng $q\left(x\right)$, theo công thức (3.53):

$$H\left(p, q\right) = -E\left[\ln q\left(X\right)\right] = -\sum_{x \in X} p\left(x\right) \ln q\left(x\right). \tag{8.16}$$

Đặc biệt, nếu sử dụng $\log_2$, cross−entropy sẽ biểu diễn số lượng bit bổ sung cần thiết để mã hóa phân phối gốc bằng phân phối dự đoán. Rõ ràng khi $L = 0$,

hai phân phối là như nhau, và không có lỗi dự đoán. Do đó, cực tiểu hóa cross–entropy là cách tiếp cận tinh tế để tối ưu hóa sai số dự đoán trong các bài toán phân loại mà nhãn mục tiêu là rời rạc.

8.3 Triển khai và tối ưu

Xét một tập dữ liệu huấn luyện với 5 mẫu đã gán nhãn nhị phân:

$\boldsymbol{x}^{(i)T}$	$x_1^{(i)}$	$x_2^{(i)}$	y_i
$\boldsymbol{x}^{(1)T}$	1	2	1
$\boldsymbol{x}^{(2)T}$	2	3	1
$\boldsymbol{x}^{(3)T}$	3	1	1
$\boldsymbol{x}^{(4)T}$	2	5	0
$\boldsymbol{x}^{(5)T}$	4	3	0
$\boldsymbol{x}$	1	4	y

Ta sẽ dự báo nhãn cho điểm $\boldsymbol{x}$, và xây dựng đường phân tách hai lớp. Từ đó trực quan hóa các thành phần của bài toán bao gồm

a) Các điểm gán nhãn 1;

b) Các điểm gán nhãn 0;

c) Nhãn dự báo cho các điểm; và

d) Đường phân tách hai lớp.

8.3.1 Thư viện scikit-learn

`scikit-learn` triển khai lớp `LogisticRegression`, có thể giải quyết bài toán này bằng cách sử dụng các thuật toán tối ưu.

Trước hết ta khai báo các điểm dữ liệu và nhãn tương ứng của chúng:

```
1 import numpy as np
2 X = np.array([[1, 2], [2, 3], [3, 1], [2, 5], [4,
     3]])
3 Y = np.array([1, 1, 1, 0, 0])
```

Tiếp theo, khởi tạo và huấn luyện mô hình hồi quy logistic

```
4 from sklearn.linear_model import LogisticRegression

5 logreg = LogisticRegression()
6 logreg.fit(X, Y)
```

Để dự báo nhãn cho điểm $\boldsymbol{x} = (1, 4)^T$:

```
X_pred = np.array([[1, 4]])
Y_pred = logreg.predict(X_pred)
```

```
Y_pred
```

```
array([1])
```

tức là nhãn dự báo của điểm $\boldsymbol{x}$ này là $\widehat{y} = \widehat{f}(\boldsymbol{x}) = 1$.

Phương trình đường phân tách có dạng $z = \boldsymbol{\theta}^T\boldsymbol{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, trong đó θ_1, θ_2 là các hệ số tương ứng của các thuộc tính x_1, x_2, được xác định bởi lệnh

```
logreg.coef_
```

```
array([[-0.64552319, -0.88436625]])
```

tức là $\theta_1 = -0.64552, \theta_2 = -0.88437$:

```
# Gán giá trị cho θ1, θ2
t1, t2 = logreg.coef_[0]
```

và θ_0 là hệ số tự do, được xác định bởi lệnh:

```
logreg.intercept_
```

```
array([4.59233122])
```

tức là $\theta_0 = 4.5923$.

```
# Gán giá trị cho θ0
t0 = logreg.intercept_[0]
```

Ta đưa phương trình này về dạng $x_2 = ax_1 + b$, trong đó $a = -\dfrac{\theta_1}{\theta_2} = -0.72993$, $b = -\dfrac{\theta_0}{\theta_2} = 5.1928$:

```
a = - t1 / t2
b = - t0 / t2
```

Bây giờ ta đã đủ thông tin để trực quan hóa các thành phần của bài toán, đặc biệt là đường phân tách, trên Hình 8.4:

```
# Vẽ biểu đồ
plt.figure()
```

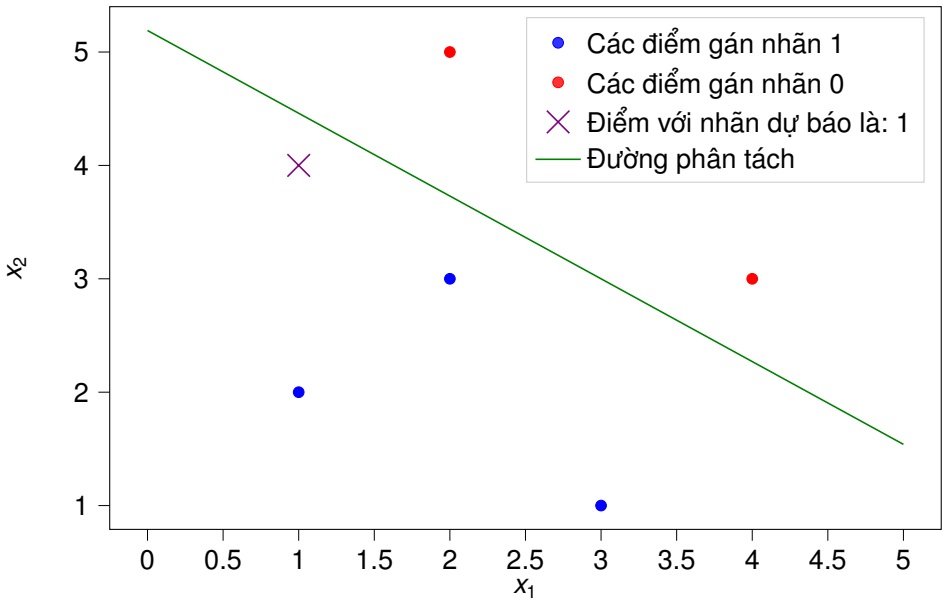
```
# (1) Vẽ các điểm gán nhãn 1
plt.scatter(X[Y == 1][:, 0], X[Y == 1][:, 1], color='blue')

# (2) Vẽ các điểm gán nhãn 0
plt.scatter(X[Y == 0][:, 0], X[Y == 0][:, 1], color='red')

# (3) Vẽ một điểm mới với nhãn dự báo
plt.scatter(X_pred[0, 0], X_pred[0, 1], color='purple', marker='x', s=100)

# (4) Xây dựng đường phân tách
X1 = np.linspace(0, 5, 100)
X2 = a * X1 + b
plt.plot(X1, X2, color='green')

# Gán nhãn cho các thành phần, đặt tên trục
plt.legend(['Các điểm nhãn 1', 'Các điểm nhãn 0', f'Điểm với nhãn dự báo là: {Y_pred[0]}', 'Đường phân tách'])
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')

# Hiển thị biểu đồ
plt.show()
```

8.3.2 Thư viện scipy

Để giải mô hình hồi quy logistic sử dụng thư viện `scipy`, ta có thể tận dụng hàm `minimize` từ module `scipy.optimize` để thực hiện quá trình tối ưu hóa. Ta cần định nghĩa hàm mất mát và gradient của nó, sau đó sử dụng hàm `minimize` để tìm các tham số tối ưu.

Từ công thức (8.15) của hàm mất mát L, ta tính được gradient của L theo tham số $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} L = \sum_{i=1}^{m} \left[\sigma\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right) - y_i\right] \boldsymbol{x}^{(i)} \tag{8.17}$$



Hình 8.4: Hồi quy logistic

Thật vậy, với $z = \boldsymbol{\theta}^T \boldsymbol{x}$, thì

$$\nabla_{\boldsymbol{\theta}} z = \boldsymbol{x}. \tag{8.18}$$

Theo công thức đạo hàm của hàm hợp

$$\nabla_{\boldsymbol{\theta}} \ln \sigma(z) = \frac{1}{\sigma(z)} \frac{d\sigma(z)}{dz} \nabla_{\boldsymbol{\theta}} z. \tag{8.19}$$

Áp dụng công thức (3.37) về tính chất của hàm sigmoid, $\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$, suy ra:

$$\nabla_{\boldsymbol{\theta}} \ln \sigma(z) = \frac{1}{\sigma(z)} \sigma(z)(1 - \sigma(z))\, \boldsymbol{x} = (1 - \sigma(z))\, \boldsymbol{x}. \tag{8.20}$$

Tương tự

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \ln(1 - \sigma(z)) &= \frac{1}{1 - \sigma(z)} \frac{d(1 - \sigma(z))}{z} \nabla_{\boldsymbol{\theta}} z = \frac{1}{1 - \sigma(z)} \frac{-d\sigma(z)}{z} \boldsymbol{x} \\ &= -\frac{1}{1 - \sigma(z)} \sigma(z)(1 - \sigma(z))\, \boldsymbol{x} = -\sigma(z)\, \boldsymbol{x}. \end{aligned} \tag{8.21}$$

```python
import numpy as np

# Tạo dữ liệu
X = np.array([[1, 2], [2, 3], [3, 1], [2, 5], [4, 3]])
```

```
Y = np.array([1, 1, 1, 0, 0])  # nhãn

# Thêm cột bias (cột đơn vị toàn số 1) vào X
X = np.hstack((np.ones((X.shape[0], 1)), X))  # X giờ có cỡ m × (n + 1)
```

```
array([[1., 1., 2.],
       [1., 2., 3.],
       [1., 3., 1.],
       [1., 2., 5.],
       [1., 4., 3.]])
```

```
# Hàm sigmoid
def sigmoid(z):
    return 1 / (1 + np.exp(-z))


# Hàm mất mát (log-likelihood hay cross-entropy loss)
def loss(theta):
    L = 0
    for x, y in zip(X, Y):
        if y == 0:
            L += - np.log(1 - sigmoid(np.dot(theta, x)))
        else:
            L += - np.log(sigmoid(np.dot(theta, x)))
    return L

# Hàm tính gradient của hàm mất mát
def gradient(theta):
    G = 0 # mặc dù là số nhưng sau khi thực hiện vòng lặp lại là véctơ
    for x, y in zip(X, Y):
        G += (sigmoid(np.dot(theta, x)) - y) * x
    return G


from scipy.optimize import minimize
```

```
# Khởi tạo các tham số
initial_theta = np.zeros(X.shape[1])  # véctơ tham số ban đầu với kích thước (n+1)
# Sử dụng scipy.optimize.minimize để tối ưu hóa hàm mất mát
result = minimize(loss, initial_theta, method='BFGS', jac=gradient)
```

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 1.3090013249363731e-06
        x: [ 9.493e+01 -1.749e+01 -1.472e+01]
      nit: 26
      jac: [ 1.028e-06  2.066e-06  5.413e-06]
 hess_inv: [[ 1.488e+07 -2.802e+06 -2.233e+06]
            [-2.802e+06  5.276e+05  4.204e+05]
            [-2.233e+06  4.204e+05  3.350e+05]]
     nfev: 28
     njev: 28
```

Trong kết quả trên, tham số tối ưu θ của mô hình được chỉ định tại thành phần x:

```
theta = result.x
```

```
array([ 94.92806523, -17.49268406, -14.72125792])
```

tức là $\boldsymbol{\theta} = (94.928, -17.493, -14.721)^T$.

Để dự đoán nhãn cho điểm mới $\boldsymbol{x} = (1, 4)^T$, theo công thức (8.12), ta dùng lệnh

```
x_pred = np.array([1, 1, 4])  # thêm 1 cho bias
prob_pred = sigmoid(np.dot(theta, x_pred))
y_pred = 1 if prob_pred >= 0.5 else 0
```

và được kết quả nhãn dự báo là $\widehat{y} = \widehat{f}(\boldsymbol{x}) = 1$.

Đưa phương trình tổng quát của đường phân tách $z = \boldsymbol{\theta}^T\boldsymbol{x} = \theta_0+\theta_1x_1+\theta_2x_2 = 0$ về dạng $x_2 = ax_1 + b$ thì $a = -1.1883$, $b = 6.4484$:

```
a = - theta[1] / theta[2]
b = - theta[0] / theta[2]
```

Trong một số tài liệu, hàm mất mát và gradient của nó được xây dựng như sau. Bạn đọc hãy tự suy luận tính đúng đắn hay hợp lý của mã lệnh này.

```
# Hàm mất mát (log-likelihood hay cross-entropy loss)
def loss(theta, X, Y):
    m = len(Y)
    h = sigmoid(np.dot(X, theta))
    epsilon = 1e-5  # thêm để tránh log(0)
    loss = -1/m * (np.dot(Y, np.log(h + epsilon)) +
   np.dot((1 - Y), np.log(1 - h + epsilon)))
    return loss

# Hàm tính gradient của hàm mất mát
def gradient(theta, X, Y):
    m = len(Y)
    h = sigmoid(np.dot(X, theta))
    gradient = np.dot(X.T, (h - Y)) / m
    return gradient
```

8.3.3 Phương pháp hướng giảm

Áp dụng phương pháp hướng giảm (4.5) cho bài toán cực tiểu hàm mất mát L:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \varepsilon \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}^{(k)}\right). \tag{8.22}$$

Thay vì xác định tham số tối ưu θ của mô hình bằng thuật toán tối ưu trong thư viện `scipy`, ta có thể sử dụng Thuật toán 1.

Mã thay thế cho dòng 25 – 30 trong mã lệnh ở Mục 8.3.2:

```
initial_theta = np.zeros(X.shape[1])   # θ^(0) = 0
learning_rate = 0.1                    # ε = 0.1
tolerance = 1e-5                       # δ = 10^−5
max_iters = 1e4                        # N = 10_000

theta = initial_theta
num_iters = 0
loss_history = [loss(theta)]
while True:
    num_iters += 1
    theta -= learning_rate * gradient(theta)
    loss_history.append(loss(theta))
```

Thuật toán 1: Phương pháp hướng giảm cho bài toán hồi quy logistic

Đầu vào:

- Hàm mất mát $L\left(\boldsymbol{\theta}\right)$, xấp xỉ ban đầu $\boldsymbol{\theta}^{(0)}$, tốc độ học ε
- Ngưỡng sai số δ, số bước lặp tối đa N

Đầu ra: Xấp xỉ nghiệm tối ưu $\boldsymbol{\theta}^*$ của bài toán $\arg\min_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)$

$k \leftarrow 0$;
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$;
while *true* **do**
  $k \leftarrow k + 1$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)$;
  if $\left\|\nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)\right\| < \delta \vee k = N$ **then**
    break;
  end
end

```
36      if np.linalg.norm(gradient(theta)) < tolerance or
        num_iters == max_iters:
37          break
```

8.3.4 Phương pháp Newton

Từ công thức xác định gradient của hàm mất mát (8.17), và công thức xác định ma trận Hess của hàm số nhiều biến (4.6), ta có thể thu được

$$\nabla^2_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right) = \sum_{i=1}^{m} \sigma\left(z_i\right)\left(1 - \sigma\left(z_i\right)\right) \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)^T}. \tag{8.23}$$

Phương pháp Newton (4.12) cho bài toán cực tiểu hàm mất mát L có dạng

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \left[\nabla^2_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)\right]^{-1} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}^{(k)}\right). \tag{8.24}$$

Mã thay thế cho dòng 25 – 30 trong mã lệnh ở Mục 8.3.2, trong đó ta cần lưu ý khai báo thêm ma trận Hess:

```
25  def Hess(theta):
26      H = 0
```

Thuật toán 2: Phương pháp Newton cho bài toán hồi quy logistic

Đầu vào:

- Hàm mất mát $L\left(\boldsymbol{\theta}\right)$, xấp xỉ ban đầu $\boldsymbol{\theta}^{(0)}$
- Ngưỡng sai số δ, số bước lặp tối đa N

Đầu ra: Xấp xỉ nghiệm tối ưu $\boldsymbol{\theta}^*$ của bài toán $\arg\min_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)$

$k \leftarrow 0$;
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$;
while *true* **do**
  $k \leftarrow k + 1$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \left[\boldsymbol{H}(L)\left(\boldsymbol{\theta}\right)\right]^{-1} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)$;
  if $\left\|\nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}\right)\right\| < \delta \vee k = N$ **then**
    break;
  end
end

```
    for x in X:
        s = sigmoid(np.dot(theta, x))
        H += s * (1 - s) * np.dot( np.expand_dims(x,
   axis=1), np.expand_dims(x, axis=0) )
    return H / len(X)

initial_theta = np.zeros(X.shape[1])   # θ^(0) = 0
tolerance = 1e-5                       # δ = 10^-5
max_iters = 1e4                        # N = 10_000

theta = initial_theta
num_iters = 0
loss_history = [loss(theta)]
while True:
    num_iters += 1
    theta -= np.dot(np.linalg.inv(Hess(theta)),
   gradient(theta))
    loss_history.append(loss(theta))
    if np.linalg.norm(gradient(theta)) < tolerance or
    num_iters == max_iters:
```

```
42            break
```

Phần II

Mạng nơron sâu: các thực hành hiện đại

Phần III

Nghiên cứu học sâu

Tài liệu tham khảo

1. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., và Bengio, Y. *Theano: A CPU and GPU Math Compiler in Python* **in** *Proceedings of the 9th Python in Science Conference* (**editors** van der Walt, S., và Millman, J.) (2010), trang 18
bibrangessep —
bibrangessep 24.

2. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., và Bengio, Y. *Theano: new features and speed improvements* 2012. arXiv: 1211.5590 [cs.SC]. https://arxiv.org/abs/1211.5590.

3. A., C. Méthode générale pour la résolution de systèmes d'équations simultanées. *Compte rendu des séances de l'académie des sciences* Tập 25, trang 536
bibrangessep —
bibrangessep 538. https://cir.nii.ac.jp/crid/1573668925829538688 (1847).

4. Stuart J. Russell, P. N. *Artificial Intelligence: A Modern Approach, Global Edition* In lần thứ 4. 1167 trang (Pearson, 2021).

5. Jorge Nocedal, S. W. *Numerical Optimization* In lần thứ 2. 685 trang (Springer, 2006).

6. S. Boyd, L. V. *Convex Optimization* 732 trang (Cambridge, 2004).

7. Rockafellar, R. T. *Convex Analysis* 470 trang (Princeton University Press, 1970).

8. Rosen, J. B. The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints. *Journal of the Society for Industrial and Applied Mathematics* Tập 8, trang 181
bibrangessep —
bibrangessep 217. https://doi.org/10.1137/0108011 (1960).

9. Karush, W. *Minima of Functions of Several Variables with Inequalities as Side Conditions* mathesis (Department of Mathematics, University of Chicago, Chicago, IL, USA, 1939). 29 trang.
10. Bonaccorso, G. *Machine Learning Algorithms* In lần thứ 2. 514 trang (Packt, 2018).
11. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* In lần thứ 3. 864 trang (O'Reilly, 2022).
12. Goodfellow, I. *Deep Learning* 801 trang (2016).
13. Kaare Brandt Petersen, M. S. P. *The Matrix Cookbook* 72 trang (2012).
14. Shilov, G. E. *Linear Algebra* 864 trang (Dover Publications, 1977).
15. Jaynes, E. T. *Probability Theory: The Logic of Science* In lần thứ 22. 759 trang (Cambridge University Press, 2003).
16. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* 573 trang (Morgan Kaufmann, 1988).
17. Ramsey, F. P. *The Foundations of Mathematics and Other Logical Essays* 310 trang (Routledge, 1926).
18. Murphy, K. P. *Machine Learning: A Probabilistic Perspective* 1098 trang (MIT Press, 2012).
19. Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., và Garcia, R. *Incorporating Second-Order Functional Knowledge for Better Option Pricing* **in** *Advances in Neural Information Processing Systems* (**editors** Leen, T., Dietterich, T., và Tresp, V.) Tập 13 (MIT Press, 2000).
20. Cover, T. M., và Thomas, J. A. *Elements of Information Theory* In lần thứ 2. 774 trang (Wiley-Interscience, 2006).
21. D.J.C., M. *Information Theory, Inference and Learning Algorithms* 642 trang (Cambridge University Press, 2003).
22. Kuhn, H. W., và Tucker, A. W. *Nonlinear Programming* **in** *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (University of California Press, Berkeley **and** Los Angeles, 1951), trang 481
bibrangessep -–
bibrangessep 492.