HỒI QUY TUYẾN TÍNH VÀ ỨNG DỤNG

Giảng viên: LÊ NHẬT TÙNG

Phần 1: Giới thiệu Hồi quy tuyến tính

Hồi quy tuyến tính là một trong những kỹ thuật thống kê cơ bản và quan trọng nhất, được sử dụng rộng rãi trong nhiều lĩnh vực từ kinh tế, y tế đến khoa học dữ liệu. Phương pháp này giúp chúng ta hiểu và dự đoán mối quan hệ giữa các biến số, tạo nền tảng vững chắc cho việc phát triển các mô hình phức tạp hơn trong học máy.

Trong phần này, chúng ta sẽ tìm hiểu bản chất của hồi quy tuyến tính, tại sao nó quan trọng, và làm thế nào để áp dụng nó vào các bài toán thực tế. Bạn sẽ thấy rằng mặc dù đơn giản, hồi quy tuyến tính vẫn là công cụ mạnh mẽ không thể thiếu trong hộp công cụ của bất kỳ nhà khoa học dữ liệu nào.

Hồi quy tuyến tính là gì?

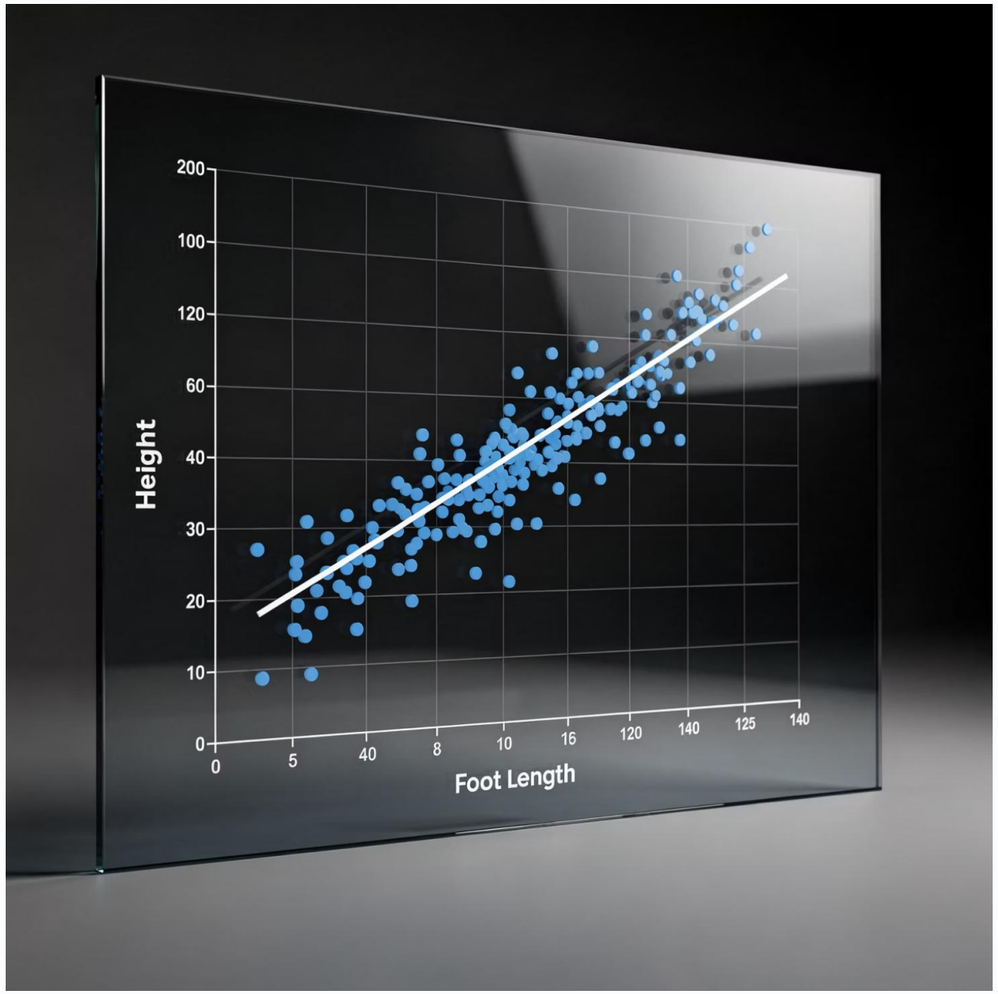
Hồi quy tuyến tính là một phương pháp thống kê được sử dụng để mô hình hóa mối quan hệ tuyến tính giữa một biến phụ thuộc (biến mục tiêu) và một hoặc nhiều biến độc lập (biến đặc trưng). Đây là một kỹ thuật dự đoán cơ bản nhưng cực kỳ hiệu quả trong nhiều tình huống thực tế.

Ví dụ đơn giản: Giả sử chúng ta muốn dự đoán chiều cao của một người dựa trên chiều dài bàn chân của họ. Mô hình có thể có dạng:

Chiều cao = 80 + 3.5 × Chiều dài bàn chân

Trong công thức này, 80 là hệ số chặn (intercept) - chiều cao cơ bản, và 3.5 là hệ số góc (slope) - cho biết chiều cao tăng bao nhiêu cm khi chiều dài bàn chân tăng 1 cm.

Mục tiêu chính của hồi quy tuyến tính là tìm ra đường thẳng phù hợp nhất với tập dữ liệu quan sát được, sao cho tổng khoảng cách từ các điểm dữ liệu đến đường thẳng là nhỏ nhất. Đường thẳng này sau đó có thể được sử dụng để dự đoán các giá trị mới.



Tại sao hồi quy tuyến tính quan trọng?

Dự đoán chính xác

Hồi quy tuyến tính cho phép dự đoán các giá trị liên tục trong nhiều lĩnh vực quan trọng như kinh tế (dự báo doanh thu, giá cả), y tế (ước tính liều lượng thuốc, tiên lượng bệnh), và kỹ thuật (tối ưu hóa quy trình sản xuất).

Đơn giản và hiệu quả

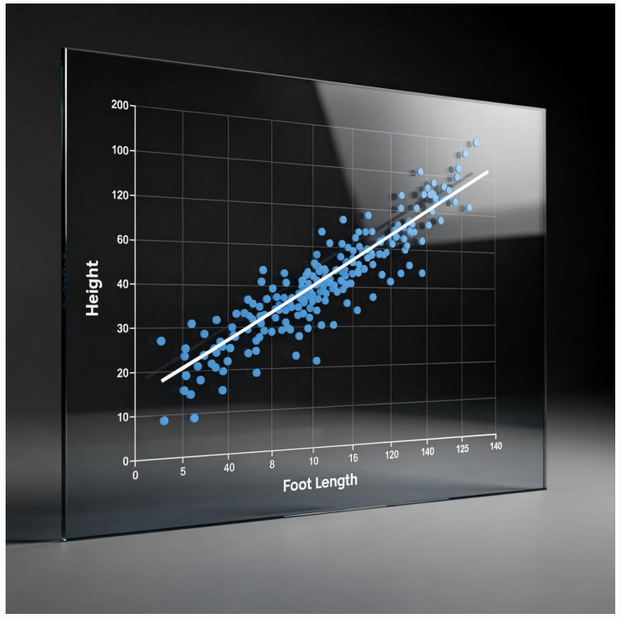
Với độ phức tạp tính toán thấp, mô hình dễ hiểu và giải thích, hồi quy tuyến tính là lựa chọn đầu tiên khi dữ liệu có mối quan hệ tuyến tính. Nó cung cấp kết quả nhanh chóng và đáng tin cậy mà không cần tài nguyên tính toán lớn.

Nền tảng học máy

Hồi quy tuyến tính là điểm xuất phát để hiểu các thuật toán học máy phức tạp hơn như hồi quy logistic, mạng nơ-ron, và support vector machines. Nắm vững nó giúp bạn dễ dàng tiếp cận các kỹ thuật tiên tiến hơn.

Ngoài ra, hồi quy tuyến tính còn được sử dụng rộng rãi trong phân tích thống kê để kiểm định giả thuyết, đo lường mức độ ảnh hưởng của các biến, và xác định các yếu tố quan trọng trong một hệ thống phức tạp. Khả năng giải thích rõ ràng các hệ số của mô hình giúp người ra quyết định hiểu được tác động của từng yếu tố đến kết quả cuối cùng.

Minh họa mối quan hệ tuyến tính



Đồ thị bên cạnh minh họa một ví dụ điển hình về mối quan hệ tuyến tính giữa chiều dài bàn chân (trục x) và chiều cao (trục y). Mỗi điểm xanh đại diện cho một quan sát thực tế, trong khi đường thẳng màu đỏ là đường hồi quy - đường thẳng phù hợp nhất với dữ liệu.

Quan sát đồ thị, chúng ta có thể thấy:

- Các điểm dữ liệu phân tán xung quanh đường thẳng
- Xu hướng tăng rõ ràng: chiều dài bàn chân càng lớn thì chiều cao càng cao
- Mối quan hệ gần như tuyến tính, phù hợp với giả định của mô hình

Ý nghĩa thực tế

Trong thực tế, mối quan hệ tuyến tính như thế này rất phổ biến. Ví dụ: chi tiêu quảng cáo và doanh số bán hàng, nhiệt độ và tiêu thụ điện năng, số giờ học và điểm thi.

Đường hồi quy cho phép chúng ta dự đoán giá trị y (chiều cao) cho bất kỳ giá trị x (chiều dài bàn chân) nào, ngay cả những giá trị chưa xuất hiện trong tập dữ liệu huấn luyện.

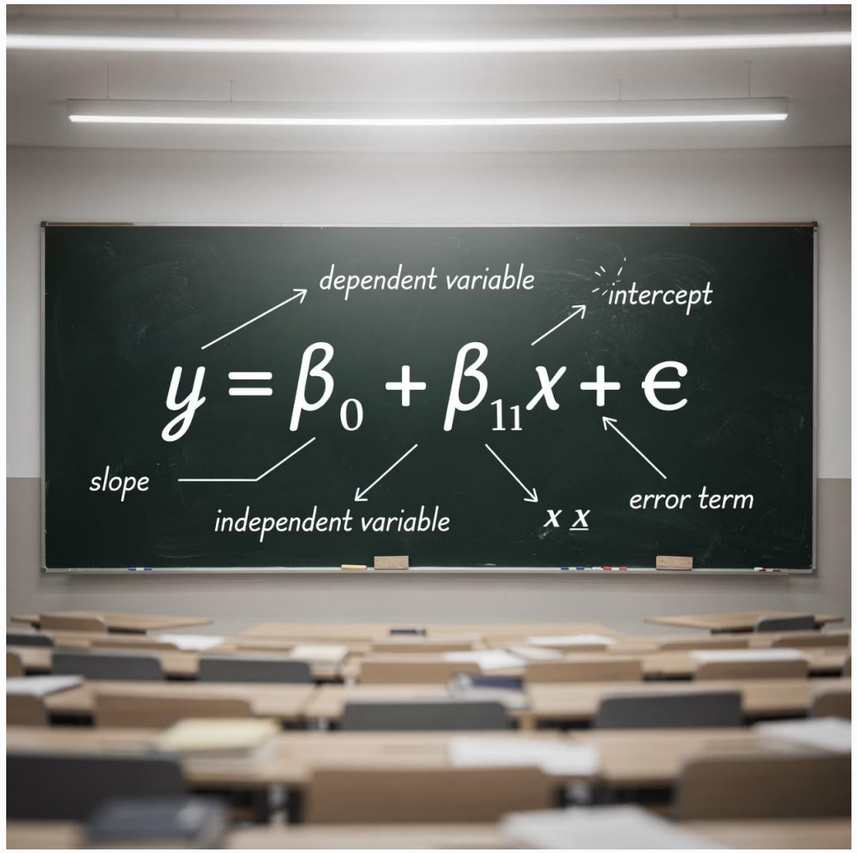
Phần 2: Toán học cơ bản của Hồi quy tuyến tính

Để thực sự hiểu và áp dụng hiệu quả hồi quy tuyến tính, chúng ta cần nắm vững nền tảng toán học đằng sau nó. Mặc dù có vẻ phức tạp, nhưng các khái niệm toán học của hồi quy tuyến tính thực ra khá trực quan và dễ hiểu khi được giải thích rõ ràng.

Trong phần này, chúng ta sẽ đi sâu vào các khía cạnh toán học quan trọng bao gồm phương trình hồi quy, ý nghĩa của các hệ số, và cách đo lường độ phù hợp của mô hình. Hiểu rõ những khái niệm này không chỉ giúp bạn sử dụng hồi quy tuyến tính một cách chính xác mà còn trang bị cho bạn kiến thức để tiếp cận các mô hình phức tạp hơn trong tương lai.

Hãy cùng khám phá từng thành phần của mô hình hồi quy tuyến tính một cách chi tiết và hệ thống.

Phương trình hồi quy tuyến tính đơn biến



Phương trình cơ bản của hồi quy tuyến tính đơn biến được biểu diễn như sau:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

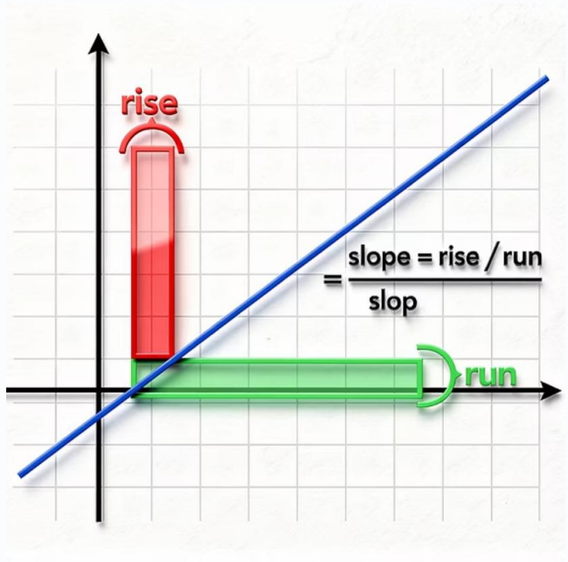
Trong đó:

- **y**: Biến phụ thuộc (biến mục tiêu cần dự đoán)
- **x**: Biến độc lập (biến đặc trưng, biến đầu vào)
- **β_0 (beta không)**: Hệ số chặn (intercept) - giá trị của y khi x = 0
- **β_1 (beta một)**: Hệ số góc (slope) - độ dốc của đường thẳng
- **ε (epsilon)**: Sai số ngẫu nhiên - phần không giải thích được bởi mô hình

Phương trình này mô tả một đường thẳng trong không gian hai chiều, với β_0 xác định vị trí đường thẳng cắt trục y và β_1 xác định độ nghiêng của đường thẳng.

Lưu ý quan trọng: Sai số ε thể hiện sự khác biệt giữa giá trị thực tế và giá trị dự đoán bởi mô hình. Trong thực tế, không có mô hình nào hoàn hảo 100%, do đó sai số này luôn tồn tại và được giả định tuân theo phân phối chuẩn với trung bình bằng 0.

Ý nghĩa các hệ số

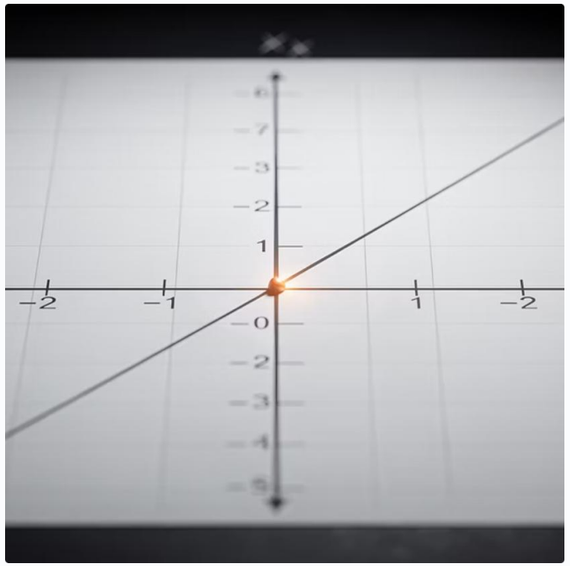


Hệ số góc β_1

Hệ số góc β_1 thể hiện mức độ thay đổi của biến phụ thuộc y khi biến độc lập x tăng thêm 1 đơn vị. Đây là chỉ số quan trọng nhất để hiểu mối quan hệ giữa hai biến.

Ví dụ: Nếu $\beta_1 = 3.5$ trong mô hình dự đoán chiều cao từ chiều dài bàn chân, điều này có nghĩa là khi chiều dài bàn chân tăng 1 cm, chiều cao dự kiến tăng 3.5 cm.

- $\beta_1 > 0$: Mối quan hệ dương (x tăng, y tăng)
- $\beta_1 < 0$: Mối quan hệ âm (x tăng, y giảm)
- $\beta_1 = 0$: Không có mối quan hệ tuyến tính



Hệ số chặn β_0

Hệ số chặn β_0 là giá trị của biến phụ thuộc y khi tất cả các biến độc lập bằng 0. Đây là điểm mà đường hồi quy cắt trục y trên đồ thị.

Ví dụ: Nếu $\beta_0 = 80$ trong mô hình chiều cao, điều này có nghĩa là khi chiều dài bàn chân bằng 0 (giả định lý thuyết), chiều cao dự kiến là 80 cm.

Lưu ý: Trong nhiều trường hợp thực tế, việc giải thích β_0 có thể không có ý nghĩa thực tiễn nếu x = 0 không phải là giá trị hợp lý hoặc có thể xảy ra. Tuy nhiên, β_0 vẫn rất quan trọng cho độ chính xác của mô hình.

Hàm mất mát (Loss Function)

Để tìm ra các hệ số β_0 và β_1 tối ưu, chúng ta cần một cách để đo lường độ chính xác của mô hình. Đó chính là vai trò của hàm mất mát - một hàm toán học định lượng sự khác biệt giữa giá trị thực tế và giá trị dự đoán.

Hàm MSE (Mean Squared Error) là hàm mất mát phổ biến nhất trong hồi quy tuyến tính, được định nghĩa như sau:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Trong đó:

- n: Số lượng điểm dữ liệu
- y_i: Giá trị thực tế thứ i
- ŷ_i: Giá trị dự đoán thứ i

MSE tính trung bình của bình phương các sai số. Việc bình phương sai số có hai lợi ích:

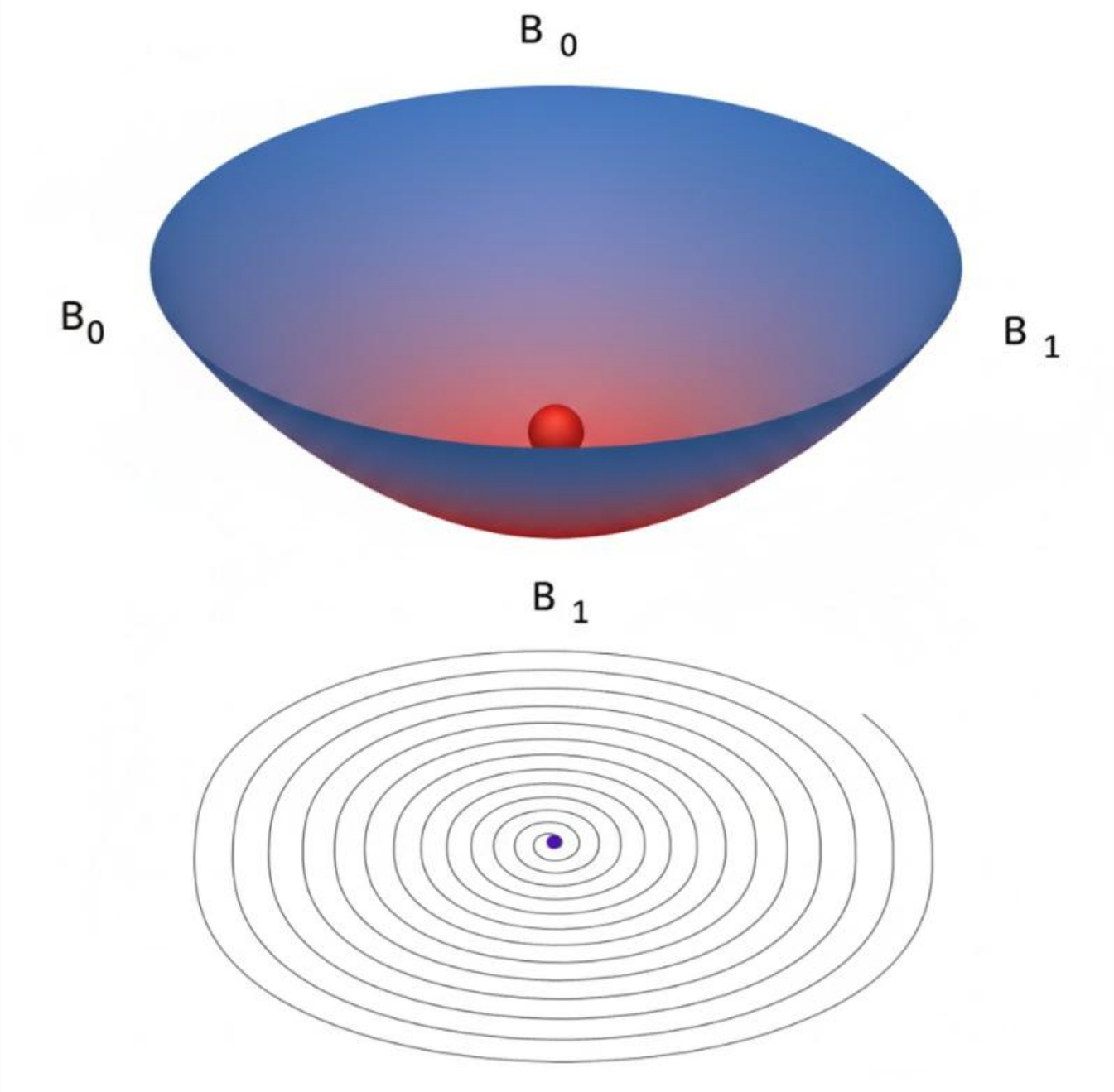
1. Loại bỏ dấu âm (sai số dương và âm đều được tính)
2. Phạt nặng hơn các sai số lớn (sai số càng lớn càng bị phạt nhiều)

Mục tiêu tối ưu hóa

Mục tiêu của quá trình huấn luyện mô hình là tìm các giá trị β_0 và β_1 sao cho MSE đạt giá trị nhỏ nhất. Điều này có nghĩa là:

- Đường hồi quy phải đi gần các điểm dữ liệu nhất có thể
- Tổng khoảng cách từ các điểm đến đường thẳng là nhỏ nhất
- Mô hình có độ chính xác cao nhất trên tập dữ liệu huấn luyện

Các hàm mất mát khác: Ngoài MSE, còn có MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), và Huber Loss. Mỗi hàm có ưu nhược điểm riêng tùy thuộc vào đặc điểm dữ liệu và mục tiêu bài toán.


B 0
B0
B 1
B 1

Trực quan hóa hàm mất mát

Đồ thị trên minh họa bề mặt của hàm mất mát MSE trong không gian ba chiều, với hai trục đại diện cho các tham số β_0 và β_1, và trục thứ ba đại diện cho giá trị MSE. Hình dạng parabol (hay chính xác hơn là paraboloid trong 3D) là đặc điểm quan trọng của hàm mất mát này.

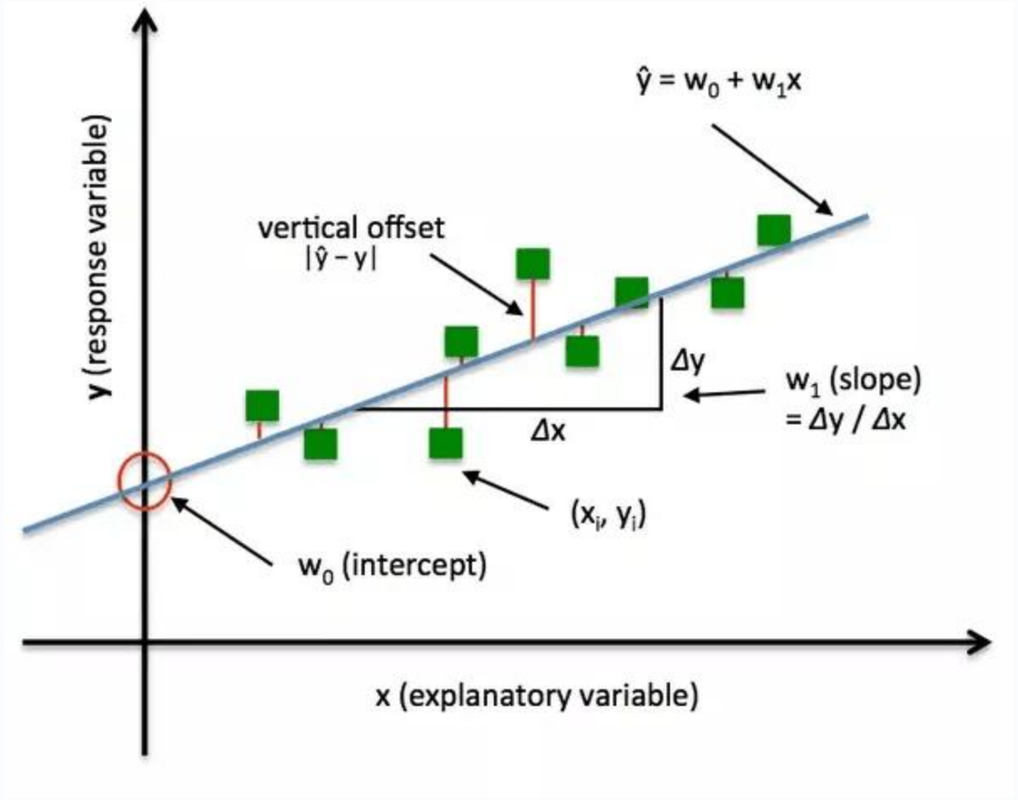
Đặc điểm quan trọng của đồ thị

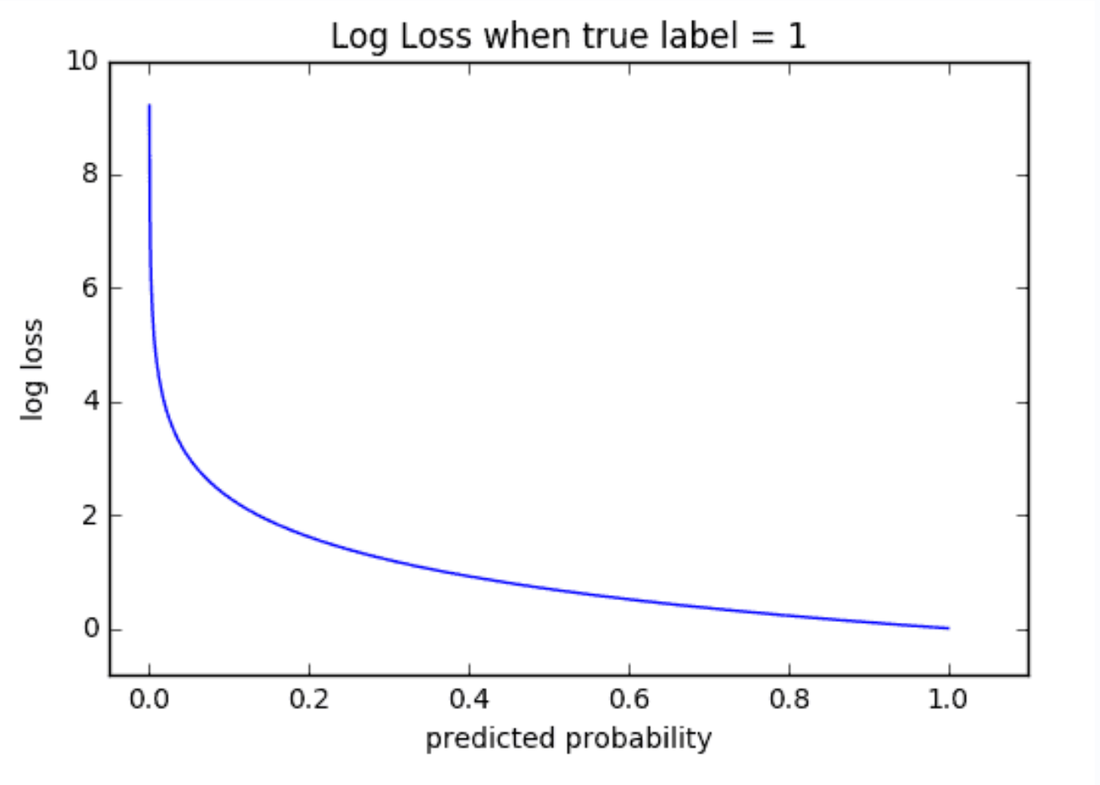
- **Hình dạng lồi (convex):** Hàm MSE có dạng bát úp, đảm bảo chỉ có một điểm cực tiểu duy nhất - điểm toàn cục (global minimum)
- **Điểm cực tiểu toàn cục:** Điểm thấp nhất trên bề mặt, đây chính là bộ tham số tối ưu (β_0*, β_1*) mà chúng ta cần tìm
- **Gradient:** Hướng dốc nhất trên bề mặt, chỉ ra hướng mà chúng ta cần di chuyển để giảm MSE nhanh nhất
- **Đường đồng mức:** Các đường tròn đồng tâm khi nhìn từ trên xuống, mỗi đường tương ứng với một giá trị MSE cố định

Ý nghĩa thực tế

Việc tối ưu hóa mô hình hồi quy tuyến tính tương đương với việc tìm điểm thấp nhất trên bề mặt này. Các thuật toán như Gradient Descent sẽ "đi xuống" theo hướng dốc nhất để nhanh chóng đạt đến điểm cực tiểu.

Hình dạng lồi của hàm MSE là lý do tại sao hồi quy tuyến tính luôn tìm được nghiệm tối ưu toàn cục, không bị mắc kẹt ở cực tiểu địa phương như nhiều mô hình phức tạp khác.


ŷ = w0 + w1x
vertical offset
|ŷ − y|
y (response variable)
Δy
w1 (slope)
= Δy / Δx
Δx
(xi, yi)
w0 (intercept)
x (explanatory variable)

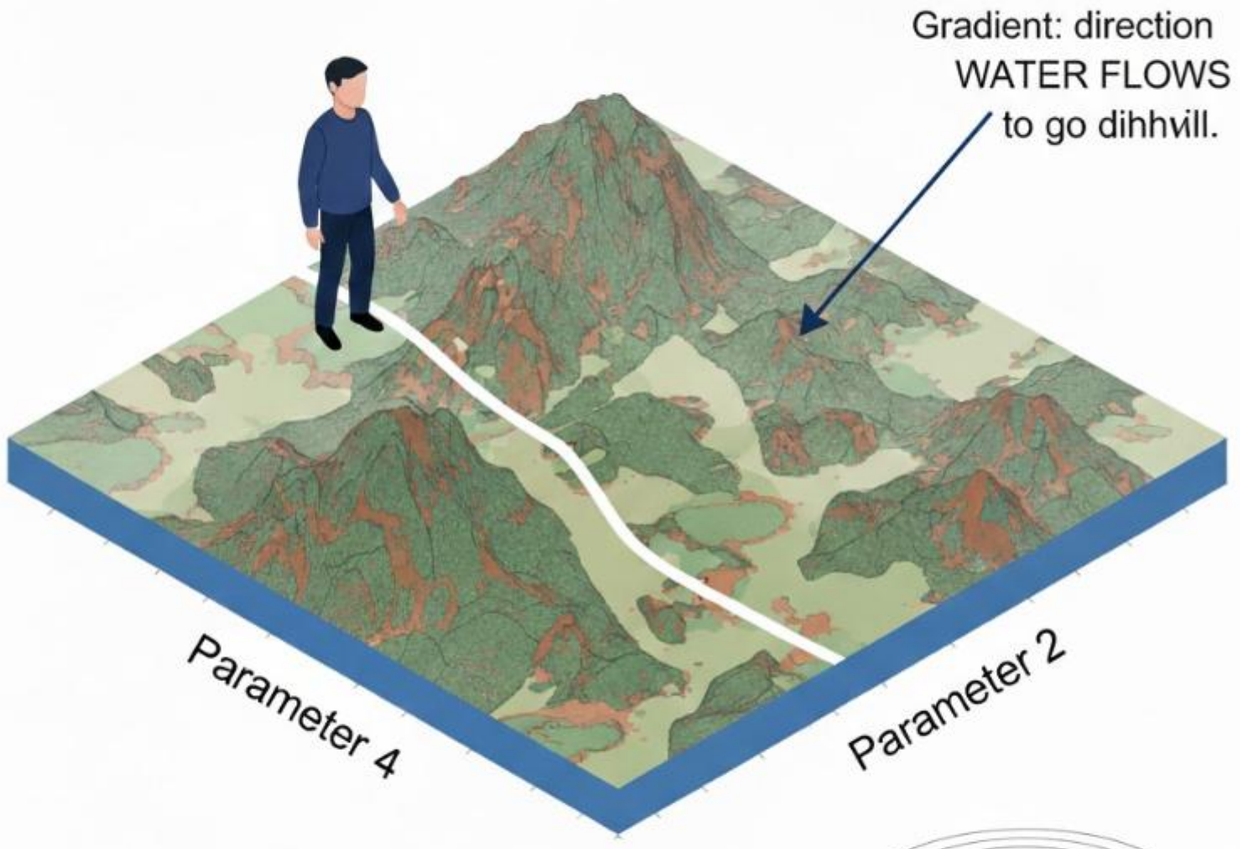

Log Loss when true label = 1
log loss
predicted probability
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

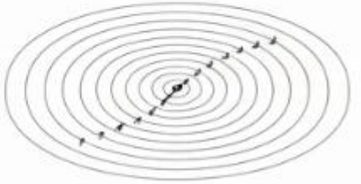
Phần 3: Phương pháp tìm tham số - Hạ Gradient

Sau khi hiểu về hàm mất mát và mục tiêu tối ưu hóa, câu hỏi tiếp theo là: Làm thế nào để tìm ra các giá trị β_0 và β_1 tối ưu? Có hai phương pháp chính: phương pháp giải tích (closed-form solution) sử dụng Normal Equation, và phương pháp lặp (iterative method) sử dụng Gradient Descent.

Gradient Descent (Hạ Gradient) là một thuật toán tối ưu hóa mạnh mẽ và linh hoạt, không chỉ được sử dụng trong hồi quy tuyến tính mà còn là nền tảng cho việc huấn luyện hầu hết các mô hình học máy và học sâu hiện đại. Thuật toán này mô phỏng quá trình "đi xuống đồi" - bắt đầu từ một điểm ngẫu nhiên trên bề mặt hàm mất mát, sau đó liên tục di chuyển theo hướng dốc nhất cho đến khi đạt đến điểm thấp nhất.

Hãy cùng tìm hiểu chi tiết về cách hoạt động, ưu nhược điểm, và các biến thể của thuật toán Gradient Descent.





View from above: Each ring = same error
Moving to the center= minimum error.

Ý tưởng hạ gradient

Bước 1: Tính đạo hàm

Tính đạo hàm riêng của hàm mất mát MSE theo từng tham số β_0 và β_1. Đạo hàm này cho biết hướng và tốc độ tăng của hàm mất mát.

$$\frac{\partial MSE}{\partial \beta_0}, \frac{\partial MSE}{\partial \beta_1}$$

Bước 2: Cập nhật tham số

Cập nhật các tham số theo hướng ngược lại với gradient (hướng giảm), với một bước nhảy được kiểm soát bởi learning rate α:

$$\beta_0 := \beta_0 - \alpha \frac{\partial MSE}{\partial \beta_0}$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial MSE}{\partial \beta_1}$$

Bước 3: Lặp lại

Lặp lại quá trình tính gradient và cập nhật tham số cho đến khi hàm mất mát hội tụ (thay đổi rất nhỏ giữa các lần lặp) hoặc đạt số vòng lặp tối đa.

Learning Rate (Tốc độ học)

Learning rate α là siêu tham số quan trọng nhất của Gradient Descent:

- **α quá nhỏ:** Quá trình hội tụ chậm, cần nhiều vòng lặp
- **α quá lớn:** Có thể bỏ qua điểm cực tiểu, không hội tụ hoặc dao động
- **α vừa phải:** Hội tụ nhanh và ổn định đến điểm cực tiểu

Các biến thể phổ biến

- **Batch Gradient Descent:** Sử dụng toàn bộ dữ liệu mỗi lần cập nhật
- **Stochastic Gradient Descent (SGD):** Sử dụng từng điểm dữ liệu
- **Mini-batch Gradient Descent:** Sử dụng một tập con nhỏ dữ liệu

Ưu điểm và nhược điểm của hạ gradient

Ưu điểm

- **Tính linh hoạt cao:** Có thể áp dụng cho nhiều loại mô hình phức tạp khác nhau, không chỉ hồi quy tuyến tính. Đây là nền tảng cho hầu hết các thuật toán học sâu hiện đại.
- **Khả năng mở rộng tốt:** Hoạt động hiệu quả với tập dữ liệu lớn, đặc biệt khi sử dụng các biến thể như SGD hoặc Mini-batch GD. Không cần lưu toàn bộ dữ liệu trong bộ nhớ.
- **Dễ cài đặt:** Thuật toán đơn giản, trực quan và dễ lập trình. Có sẵn trong hầu hết các thư viện học máy phổ biến.
- **Cho phép huấn luyện online:** Có thể cập nhật mô hình liên tục khi có dữ liệu mới mà không cần huấn luyện lại từ đầu.
- **Đảm bảo hội tụ:** Với hàm lồi như MSE trong hồi quy tuyến tính, gradient descent đảm bảo tìm được nghiệm tối ưu toàn cục.

Nhược điểm và thách thức

- **Phụ thuộc learning rate:** Cần phải điều chỉnh cẩn thận tốc độ học. Giá trị không phù hợp có thể dẫn đến không hội tụ hoặc hội tụ quá chậm.
- **Tốc độ hội tụ chậm:** Với learning rate nhỏ hoặc dữ liệu lớn, có thể cần rất nhiều vòng lặp. Đặc biệt chậm khi gần đến điểm cực tiểu.
- **Nhạy cảm với khởi tạo:** Mặc dù với hàm lồi kết quả cuối cùng giống nhau, nhưng điểm khởi đầu ảnh hưởng đến số vòng lặp cần thiết.
- **Vấn đề về scaling:** Nếu các đặc trưng có thang đo khác nhau nhiều, gradient descent có thể dao động và hội tụ chậm. Cần chuẩn hóa dữ liệu trước.
- **Cực tiểu địa phương:** Với các hàm không lồi (như trong neural networks), có thể bị mắc kẹt ở cực tiểu địa phương thay vì tìm được cực tiểu toàn cục.

Giải pháp cải tiến: Nhiều biến thể tiên tiến của Gradient Descent đã được phát triển để khắc phục các nhược điểm trên, bao gồm Adam, RMSprop, AdaGrad, và Momentum. Các thuật toán này tự động điều chỉnh learning rate và cải thiện tốc độ hội tụ.

Phần 4: Hồi quy tuyến tính đa biến

Trong thực tế, hầu hết các bài toán dự đoán đều phức tạp hơn và liên quan đến nhiều yếu tố ảnh hưởng. Hồi quy tuyến tính đa biến (Multiple Linear Regression) mở rộng mô hình đơn biến để xử lý nhiều biến độc lập cùng một lúc, cho phép chúng ta mô hình hóa các mối quan hệ phức tạp hơn trong dữ liệu thực tế.

Ví dụ, để dự đoán giá nhà, thay vì chỉ dựa vào một yếu tố như diện tích, chúng ta có thể sử dụng nhiều yếu tố khác nhau: số phòng ngủ, số phòng tắm, tuổi nhà, khoảng cách đến trung tâm, chất lượng xây dựng, và nhiều yếu tố khác. Mỗi yếu tố có mức độ ảnh hưởng khác nhau đến giá nhà, và mô hình đa biến giúp chúng ta định lượng chính xác ảnh hưởng của từng yếu tố.

Mặc dù phức tạp hơn về mặt toán học, các nguyên lý cơ bản vẫn giống với hồi quy đơn biến - chúng ta vẫn đang tìm một mối quan hệ tuyến tính, chỉ là trong không gian nhiều chiều thay vì hai chiều.

Mở rộng mô hình đa biến

Phương trình hồi quy tuyến tính đa biến mở rộng mô hình đơn biến để bao gồm nhiều biến độc lập:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Trong đó:

- **y:** Biến phụ thuộc (mục tiêu dự đoán)
- **$x_1, x_2, \ldots, x_n$:** n biến độc lập (đặc trưng)
- **β_0:** Hệ số chặn
- **$\beta_1, \beta_2, \ldots, \beta_n$:** Hệ số tương ứng với mỗi biến độc lập
- **ε:** Sai số ngẫu nhiên

Mỗi hệ số β_i thể hiện mức độ thay đổi của y khi x_i thay đổi 1 đơn vị, giữ nguyên các biến khác. Điều này cho phép chúng ta phân tích ảnh hưởng riêng biệt của từng yếu tố.

Biểu diễn ma trận

Để tính toán hiệu quả, mô hình thường được biểu diễn dưới dạng ma trận:

$$Y = X\beta + \varepsilon$$

Trong đó:

- **Y:** Vector cột chứa tất cả giá trị y
- **X:** Ma trận thiết kế chứa tất cả giá trị x
- **β:** Vector cột chứa tất cả hệ số

Dạng ma trận này không chỉ gọn gàng hơn mà còn cho phép sử dụng các phép toán ma trận hiệu quả để tối ưu hóa và dự đoán.

Nghiệm giải tích cho β có thể tìm được trực tiếp bằng Normal Equation: $\beta = (X^TX)^{-1}X^TY$

Ví dụ thực tế: Dự đoán giá nhà

Diện tích (m²)

Diện tích sàn là yếu tố quan trọng nhất. Hệ số β_1 = 25 triệu VNĐ/m² có nghĩa là mỗi m² tăng thêm làm giá nhà tăng 25 triệu đồng, giữ nguyên các yếu tố khác.

Số phòng ngủ

Số phòng ngủ ảnh hưởng đáng kể. Với β_2 = 200 triệu VNĐ/phòng, mỗi phòng ngủ thêm tăng giá nhà lên 200 triệu, phản ánh nhu cầu không gian sinh hoạt.

Khoảng cách đến trung tâm (km)

Vị trí là vàng. Hệ số β_3 = -50 triệu VNĐ/km (âm) cho thấy mỗi km xa trung tâm thêm làm giá giảm 50 triệu, phản ánh giá trị vị trí.

Mô hình hoàn chỉnh: Giá nhà (triệu VNĐ) = 500 + 25×Diện_tích + 200×Số_phòng_ngủ - 50×Khoảng_cách_TT + 100×Tuổi_nhà + ε

Với mô hình này, chúng ta có thể dự đoán giá của bất kỳ ngôi nhà nào dựa trên các đặc điểm của nó, và hiểu được mức độ đóng góp của từng yếu tố vào giá trị tổng thể.

Giả định của mô hình hồi quy tuyến tính

Để mô hình hồi quy tuyến tính hoạt động hiệu quả và đưa ra dự đoán chính xác, dữ liệu cần thỏa mãn một số giả định quan trọng. Việc kiểm tra các giả định này là bước không thể thiếu trong quá trình phân tích hồi quy.

1. Tính tuyến tính

Mối quan hệ giữa biến độc lập và biến phụ thuộc phải là tuyến tính. Nếu mối quan hệ thực sự là phi tuyến (dạng bậc hai, mũ, logarit...), mô hình tuyến tính sẽ không phù hợp.

Kiểm tra: Vẽ biểu đồ phân tán giữa x và y, kiểm tra biểu đồ residual plot.

2. Tính độc lập

Các quan sát phải độc lập với nhau. Các sai số (residuals) không được có tương quan với nhau. Giả định này thường bị vi phạm trong dữ liệu chuỗi thời gian hoặc dữ liệu không gian.

Kiểm tra: Durbin-Watson test, ACF plot của residuals.

3. Phương sai đồng nhất

Phương sai của sai số phải không đổi (homoscedasticity). Nếu phương sai thay đổi theo giá trị dự đoán (heteroscedasticity), các ước lượng sẽ không hiệu quả và khoảng tin cậy không chính xác.

Kiểm tra: Residual plot, Breusch-Pagan test, White test.

4. Phân phối chuẩn của sai số

Sai số nên tuân theo phân phối chuẩn với trung bình bằng 0. Giả định này quan trọng cho việc kiểm định giả thuyết và tính khoảng tin cậy.

Kiểm tra: Q-Q plot, Shapiro-Wilk test, Kolmogorov-Smirnov test.

5. Không đa cộng tuyến

Các biến độc lập không nên có tương quan cao với nhau (multicollinearity). Đa cộng tuyến làm cho việc ước lượng các hệ số không ổn định và khó giải thích.

Kiểm tra: Ma trận tương quan, VIF (Variance Inflation Factor), Condition Number.

6. Không có outliers ảnh hưởng

Các điểm ngoại lệ (outliers) có thể ảnh hưởng mạnh đến mô hình hồi quy, làm lệch các hệ số ước lượng. Cần xác định và xử lý các outliers một cách thích hợp.

Kiểm tra: Cook's distance, leverage plots, studentized residuals.

Lưu ý: Nếu các giả định bị vi phạm, có thể áp dụng các kỹ thuật như biến đổi dữ liệu (log, căn bậc hai), loại bỏ outliers, sử dụng robust regression, hoặc chuyển sang các mô hình phi tuyến phù hợp hơn.

Phần 5: Đánh giá mô hình hồi quy

Sau khi xây dựng mô hình hồi quy tuyến tính, việc đánh giá hiệu suất của mô hình là bước cực kỳ quan trọng. Chúng ta cần các chỉ số định lượng để trả lời các câu hỏi: Mô hình có dự đoán chính xác không? Nó tốt như thế nào so với các mô hình khác? Có nên tin tưởng vào dự đoán của nó không?

Có nhiều chỉ số đánh giá khác nhau, mỗi chỉ số cung cấp một góc nhìn riêng về hiệu suất mô hình. Việc sử dụng kết hợp nhiều chỉ số giúp chúng ta có cái nhìn toàn diện và đưa ra quyết định đúng đắn về chất lượng mô hình. Các chỉ số phổ biến bao gồm R^2 (R-squared), RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), và nhiều chỉ số khác.

Trong phần này, chúng ta sẽ tìm hiểu chi tiết về các chỉ số đánh giá quan trọng nhất và cách diễn giải chúng trong thực tế.

Các chỉ số đánh giá quan trọng

R^2 - Hệ số xác định

R^2 (R-squared hoặc coefficient of determination) đo lường tỷ lệ biến thiên của biến phụ thuộc được giải thích bởi mô hình. Công thức:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Ý nghĩa:

- Giá trị từ 0 đến 1 (hoặc 0% đến 100%)
- $R^2 = 0.8$ nghĩa là 80% biến thiên được giải thích
- R^2 cao hơn thường tốt hơn (nhưng không phải lúc nào cũng vậy)
- Trong khoa học xã hội, $R^2 = 0.5$-0.6 đã được coi là tốt
- Trong khoa học tự nhiên, thường mong đợi $R^2 > 0.8$

Hạn chế: R^2 luôn tăng khi thêm biến mới, dù biến đó có ý nghĩa hay không. Vì vậy, người ta thường dùng Adjusted R^2 cho mô hình đa biến.

RMSE - Root Mean Squared Error

RMSE là căn bậc hai của trung bình bình phương sai số, thể hiện độ lệch chuẩn của các sai số dự đoán:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Ý nghĩa:

- Đơn vị giống với biến phụ thuộc (dễ diễn giải)
- RMSE nhỏ hơn nghĩa là mô hình chính xác hơn
- Phạt nặng các sai số lớn (do bình phương)
- Nhạy cảm với outliers

Ví dụ: Nếu dự đoán giá nhà với RMSE = 500 triệu VNĐ, có nghĩa là trung bình dự đoán sai lệch khoảng 500 triệu so với giá thực tế.

MAE - Mean Absolute Error

MAE là trung bình của trị tuyệt đối các sai số, thể hiện sai số trung bình:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Ý nghĩa:

- Đơn vị giống với biến phụ thuộc
- Dễ hiểu và diễn giải hơn RMSE
- Ít nhạy cảm với outliers hơn RMSE
- Phạt đều tất cả các sai số (không bình phương)

So sánh: RMSE thường lớn hơn MAE. Nếu chênh lệch giữa chúng lớn, có thể có nhiều outliers trong dữ liệu.

Các chỉ số bổ sung: Adjusted R^2, MAPE (Mean Absolute Percentage Error), AIC (Akaike Information Criterion), BIC (Bayesian Information Criterion) - mỗi chỉ số phù hợp với mục đích và bối cảnh khác nhau.

Ý nghĩa và cách sử dụng R^2

Sơ đồ trên minh họa ba mức độ R^2 khác nhau và ý nghĩa của chúng trong thực tế. R^2 tăng dần cho thấy khả năng giải thích của mô hình được cải thiện.

Diễn giải R^2 trong thực tế

R^2 cho biết tỷ lệ phần trăm biến động của biến phụ thuộc có thể được dự đoán từ các biến độc lập:

- **R^2 = 0.3 (30%):** Mô hình yếu, chỉ giải thích được 30% biến động. 70% còn lại do các yếu tố khác không có trong mô hình.
- **R^2 = 0.6 (60%):** Mô hình khá tốt, giải thích được phần lớn biến động. Phù hợp với nhiều ứng dụng thực tế.
- **R^2 = 0.9 (90%):** Mô hình rất tốt, giải thích hầu hết biến động. Hiếm gặp trong khoa học xã hội nhưng phổ biến trong khoa học tự nhiên.

Những lưu ý quan trọng

- R^2 cao không đảm bảo nhân quả - chỉ cho thấy tương quan
- Cần xem xét cùng các chỉ số khác (RMSE, MAE)
- R^2 có thể cao do overfitting - kiểm tra trên tập validation
- So sánh R^2 chỉ có ý nghĩa với cùng tập dữ liệu

Adjusted R^2: Trong mô hình đa biến, nên sử dụng Adjusted R^2 thay vì R^2 thông thường. Adjusted R^2 điều chỉnh cho số lượng biến trong mô hình, tránh tình trạng R^2 tăng giả tạo khi thêm biến không cần thiết.

Phần 6: Minh họa hồi quy tuyến tính với Python (sklearn)

Bây giờ chúng ta sẽ chuyển từ lý thuyết sang thực hành, sử dụng Python và thư viện scikit-learn (sklearn) - một trong những thư viện học máy phổ biến và mạnh mẽ nhất. Sklearn cung cấp các công cụ đơn giản nhưng hiệu quả để xây dựng, huấn luyện và đánh giá mô hình hồi quy tuyến tính.

Python đã trở thành ngôn ngữ lập trình tiêu chuẩn cho khoa học dữ liệu và học máy nhờ vào sự đơn giản, dễ học, và hệ sinh thái thư viện phong phú. Kết hợp với các thư viện như NumPy (tính toán số học), Pandas (xử lý dữ liệu), Matplotlib/Seaborn (trực quan hóa), và Scikit-learn (học máy), Python cho phép chúng ta thực hiện toàn bộ quy trình từ thu thập dữ liệu đến triển khai mô hình một cách hiệu quả.

Trong các phần tiếp theo, chúng ta sẽ đi qua từng bước: chuẩn bị môi trường và dữ liệu, xây dựng mô hình, huấn luyện, dự đoán, và đánh giá kết quả với ví dụ code cụ thể.

Cài đặt và chuẩn bị dữ liệu

Bước 1: Import các thư viện cần thiết

```
# Thư viện tính toán số học
import numpy as np

# Thư viện xử lý dữ liệu dạng bảng
import pandas as pd

# Thư viện trực quan hóa
import matplotlib.pyplot as plt
import seaborn as sns

# Import mô hình và các công cụ từ sklearn
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

Bước 2: Tạo hoặc nạp dữ liệu

Có hai cách phổ biến để có dữ liệu:

- **Tạo dữ liệu mẫu:** Hữu ích cho việc học và thử nghiệm
- **Nạp dữ liệu thực tế:** Từ file CSV, Excel, database, hoặc API

Ví dụ tạo dữ liệu mẫu

```
# Tạo dữ liệu giả với mối quan hệ tuyến tính
np.random.seed(42)
X = 2 * np.random.rand(100, 1)  # 100 điểm dữ liệu
y = 4 + 3 * X + np.random.randn(100, 1)  # y = 4 + 3x + nhiễu
```

Ví dụ nạp dữ liệu từ file CSV

```
# Đọc file CSV
df = pd.read_csv('housing_data.csv')

# Xem vài dòng đầu
print(df.head())

# Kiểm tra thông tin cơ bản
print(df.info())
print(df.describe())

# Chọn các cột làm X và y
X = df[['area', 'bedrooms', 'bathrooms']]
y = df['price']
```

Bước 3: Chia tách dữ liệu

```
# Chia 80% huấn luyện, 20% kiểm tra
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Best Practice: Luôn chia dữ liệu thành tập huấn luyện (training set) và tập kiểm tra (test set) để đánh giá hiệu suất mô hình trên dữ liệu chưa từng thấy, tránh overfitting.

Xây dựng mô hình hồi quy tuyến tính đơn giản

01

Khởi tạo mô hình LinearRegression

Tạo một đối tượng LinearRegression từ sklearn. Đây là bước đơn giản nhất, chỉ cần một dòng code:

```
# Khởi tạo mô hình hồi quy tuyến tính
model = LinearRegression()

# Có thể điều chỉnh các tham số
(optional)
# model =
LinearRegression(fit_intercept=True,
normalize=False)
```

Tham số **fit_intercept=True** (mặc định) cho phép mô hình tính toán hệ số chặn β_0. Nếu set False, mô hình sẽ bắt buộc đường thẳng đi qua gốc tọa độ.

02

Huấn luyện mô hình với fit()

Sử dụng phương thức fit() để huấn luyện mô hình trên tập dữ liệu huấn luyện. Sklearn tự động tính toán các hệ số tối ưu:

```
# Huấn luyện mô hình
model.fit(X_train, y_train)

# Sau khi fit, mô hình đã học được các
tham số
print(f"Hệ số chặn (β₀):
{model.intercept_}")
print(f"Hệ số góc (β₁): {model.coef_}")
```

Quá trình fit() sử dụng phương pháp Ordinary Least Squares (OLS) để tìm các hệ số tối ưu, tối thiểu hóa tổng bình phương sai số.

03

Dự đoán với predict()

Sử dụng phương thức predict() để dự đoán giá trị trên dữ liệu mới (tập kiểm tra hoặc dữ liệu thực tế):

```
# Dự đoán trên tập kiểm tra
y_pred = model.predict(X_test)

# Dự đoán cho một điểm dữ liệu mới
new_data = np.array([[1.5]])  # Ví dụ:
x = 1.5
prediction = model.predict(new_data)
print(f"Dự đoán cho x=1.5:
{prediction[0]}")
```

Mô hình sử dụng công thức đã học $\hat{y} = \beta_0 + \beta_1 x$ để tính toán dự đoán cho mỗi điểm dữ liệu đầu vào.

Lưu ý: Với sklearn, quá trình này cực kỳ đơn giản và hiệu quả. Thư viện đã tối ưu hóa các phép tính ma trận, cho phép xử lý nhanh ngay cả với tập dữ liệu lớn.

Đánh giá mô hình và trực quan hóa kết quả

Tính toán các chỉ số đánh giá

```
# Tính R²
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.4f}")

# Tính RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse:.4f}")

# Tính MAE
mae = mean_absolute_error(y_test, y_pred)
print(f"MAE: {mae:.4f}")

# So sánh giá trị thực tế và dự đoán
comparison = pd.DataFrame({
    'Actual': y_test.flatten(),
    'Predicted': y_pred.flatten(),
    'Error': y_test.flatten() - y_pred.flatten()
})
print(comparison.head())
```

Trực quan hóa kết quả

```
# Vẽ đồ thị scatter plot với đường hồi quy
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue',
            label='Dữ liệu thực tế', alpha=0.6)
plt.plot(X_test, y_pred, color='red',
         linewidth=2, label='Đường hồi quy')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Hồi quy tuyến tính')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Vẽ biểu đồ so sánh actual vs predicted
plt.figure(figsize=(8, 8))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--', linewidth=2)
plt.xlabel('Giá trị thực tế')
plt.ylabel('Giá trị dự đoán')
plt.title('Actual vs Predicted')
plt.show()
```

Biểu đồ trực quan giúp chúng ta dễ dàng nhận diện các vấn đề như: điểm ngoại lệ (outliers), mẫu hình trong residuals (cho thấy mối quan hệ phi tuyến), hoặc phương sai không đồng nhất.

Code hoàn chỉnh: Hồi quy tuyến tính đơn biến với sklearn

```
# Import các thư viện cần thiết
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# === BƯỚC 1: TẠO DỮ LIỆU MẪU ===
np.random.seed(42)
X = 2 * np.random.rand(100, 1)  # 100 điểm, 1 đặc trưng
y = 4 + 3 * X + np.random.randn(100, 1)  # y = 4 + 3x + nhiễu

# === BƯỚC 2: CHIA TÁCH DỮ LIỆU ===
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# === BƯỚC 3: KHỞI TẠO VÀ HUẤN LUYỆN MÔ HÌNH ===
model = LinearRegression()
model.fit(X_train, y_train)

# Hiển thị các hệ số đã học
print(f"Hệ số chặn (β₀): {model.intercept_[0]:.4f}")
print(f"Hệ số góc (β₁): {model.coef_[0][0]:.4f}")

# === BƯỚC 4: DỰ ĐOÁN ===
y_pred = model.predict(X_test)

# === BƯỚC 5: ĐÁNH GIÁ MÔ HÌNH ===
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"\n=== KẾT QUẢ ĐÁNH GIÁ ===")
print(f"R² Score: {r2:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
# === BƯỚC 6: TRỰC QUAN HÓA ===
plt.figure(figsize=(12, 5))

# Subplot 1: Dữ liệu và đường hồi quy
plt.subplot(1, 2, 1)
plt.scatter(X_train, y_train, color='lightblue',
            label='Tập huấn luyện', alpha=0.6)
plt.scatter(X_test, y_test, color='blue',
            label='Tập kiểm tra', alpha=0.6)
plt.plot(X_test, y_pred, color='red',
         linewidth=2, label='Đường hồi quy')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Hồi quy tuyến tính')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 2: Actual vs Predicted
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred, alpha=0.6, color='green')
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--', linewidth=2, label='Dự đoán hoàn hảo')
plt.xlabel('Giá trị thực tế')
plt.ylabel('Giá trị dự đoán')
plt.title('Actual vs Predicted')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# === BƯỚC 7: DỰ ĐOÁN CHO DỮ LIỆU MỚI ===
new_X = np.array([[0.5], [1.0], [1.5]])
predictions = model.predict(new_X)
print(f"\n=== DỰ ĐOÁN CHO DỮ LIỆU MỚI ===")
for x_val, pred in zip(new_X, predictions):
    print(f"X = {x_val[0]:.1f} → ŷ = {pred[0]:.4f}")
```

Kết quả mẫu: Chạy code trên sẽ cho $R^2 \approx 0.90$, RMSE ≈ 1.0, cho thấy mô hình phù hợp tốt với dữ liệu. Các hệ số $\beta_0 \approx 4$ và $\beta_1 \approx 3$ gần với giá trị thực đã dùng để tạo dữ liệu.

Phần 7: Minh họa hồi quy tuyến tính với PyTorch

PyTorch là một framework học sâu (deep learning) mạnh mẽ và linh hoạt, được phát triển bởi Facebook AI Research. Mặc dù thường được sử dụng cho các mô hình phức tạp như mạng nơ-ron sâu, PyTorch cũng có thể được dùng để xây dựng hồi quy tuyến tính, giúp chúng ta hiểu sâu hơn về cách các mô hình học máy hoạt động ở mức độ chi tiết nhất.

Khác với sklearn (cung cấp các hàm sẵn có, dễ sử dụng), PyTorch cho phép chúng ta xây dựng mô hình từ cơ bản, kiểm soát hoàn toàn quá trình huấn luyện, và hiểu rõ từng bước của thuật toán gradient descent. Điều này đặc biệt hữu ích khi bạn muốn:

- Hiểu sâu về cơ chế hoạt động của mô hình học máy
- Tùy chỉnh quá trình huấn luyện theo nhu cầu cụ thể
- Chuẩn bị nền tảng cho việc học các mô hình phức tạp hơn (neural networks)
- Tận dụng sức mạnh của GPU để tăng tốc tính toán

Trong phần này, chúng ta sẽ xây dựng mô hình hồi quy tuyến tính từ đầu bằng PyTorch, theo dõi quá trình học của mô hình qua các epoch.

Xây dựng mô hình với PyTorch

Định nghĩa lớp LinearRegressionModel

```
import torch
import torch.nn as nn
import torch.optim as optim

# Định nghĩa mô hình kế thừa từ nn.Module
class LinearRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegressionModel, self).__init__()
        # Lớp tuyến tính: y = Wx + b
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        # Hàm forward tính toán output
        return self.linear(x)

# Khởi tạo mô hình
model = LinearRegressionModel(input_dim=1, output_dim=1)
print(model)
```

Lớp **nn.Linear** tự động khởi tạo các tham số W (weights) và b (bias) một cách ngẫu nhiên. Phương thức **forward()** định nghĩa cách tính toán từ input đến output.

Chuẩn bị dữ liệu và hyperparameters

```
# Tạo dữ liệu mẫu
X_train = torch.randn(100, 1)
y_train = 4 + 3 * X_train + torch.randn(100, 1) * 0.5

# Định nghĩa hàm mất mát và optimizer
criterion = nn.MSELoss()  # Mean Squared Error
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Hyperparameters
num_epochs = 1000
print_every = 100
```

Thành phần quan trọng:

- **criterion (MSELoss):** Hàm mất mát tính sai số
- **optimizer (Adam):** Thuật toán tối ưu để cập nhật tham số
- **lr (learning rate):** Tốc độ học = 0.01
- **num_epochs:** Số vòng lặp huấn luyện

Lợi ích của PyTorch: PyTorch sử dụng automatic differentiation (tự động tính đạo hàm) thông qua autograd engine, giúp chúng ta không cần tính toán gradient thủ công. Chỉ cần định nghĩa forward pass, PyTorch tự động tính backward pass.

Huấn luyện mô hình với MSELoss và Adam optimizer

Vòng lặp huấn luyện (Training Loop)

```
# Danh sách lưu loss để vẽ đồ thị
losses = []

# Vòng lặp qua các epochs
for epoch in range(num_epochs):
    # === FORWARD PASS ===
    # Dự đoán
    y_pred = model(X_train)

    # Tính loss
    loss = criterion(y_pred, y_train)
    losses.append(loss.item())

    # === BACKWARD PASS ===
    # Xóa gradient cũ
    optimizer.zero_grad()

    # Tính gradient
    loss.backward()

    # Cập nhật tham số
    optimizer.step()

    # In thông tin
    if (epoch + 1) % print_every == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}]')
        print(f'Loss: {loss.item():.4f}')
        print(f'W: {model.linear.weight.item():.4f}')
        print(f'b: {model.linear.bias.item():.4f}\n')

print("Huấn luyện hoàn tất!")
```

Các bước trong mỗi epoch

01

Forward Pass

Tính toán dự đoán y_pred từ input X_train bằng cách đi qua mô hình, sau đó tính loss bằng MSELoss.

02

Zero Gradient

Xóa gradient từ vòng lặp trước bằng optimizer.zero_grad(). PyTorch tích lũy gradient nên phải xóa trước khi tính mới.

03

Backward Pass

Tính gradient của loss theo tất cả tham số bằng loss.backward(). PyTorch tự động tính đạo hàm qua computational graph.

04

Update Parameters

Cập nhật tham số theo hướng giảm gradient bằng optimizer.step(). Adam tự động điều chỉnh learning rate.

Quan sát quá trình học: Loss giảm dần qua các epoch, từ giá trị cao ban đầu (ví dụ: 20-30) xuống gần 0.25 sau 1000 epochs. Các tham số W và b hội tụ dần về giá trị thực (W≈3, b≈4).

Code hoàn chỉnh: Hồi quy tuyến tính với PyTorch

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

# === BƯỚC 1: ĐỊNH NGHĨA MÔ HÌNH ===
class LinearRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        return self.linear(x)

# === BƯỚC 2: TẠO DỮ LIỆU ===
torch.manual_seed(42)
X_train = torch.randn(100, 1)
y_train = 4 + 3 * X_train + torch.randn(100, 1) * 0.5

X_test = torch.randn(20, 1)
y_test = 4 + 3 * X_test + torch.randn(20, 1) * 0.5

# === BƯỚC 3: KHỞI TẠO MÔ HÌNH VÀ CÔNG CỤ ===
model = LinearRegressionModel(input_dim=1, output_dim=1)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# === BƯỚC 4: HUẤN LUYỆN MÔ HÌNH ===
num_epochs = 1000
losses = []

for epoch in range(num_epochs):
    # Forward pass
    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)
    losses.append(loss.item())

    # Backward pass và optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # In thông tin mỗi 100 epochs
    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# === BƯỚC 5: ĐÁNH GIÁ MÔ HÌNH ===
model.eval()  # Chuyển sang chế độ evaluation
with torch.no_grad():  # Tắt gradient computation
    y_pred_test = model(X_test)
    test_loss = criterion(y_pred_test, y_test)

    # Tính R² score
    ss_res = torch.sum((y_test - y_pred_test) ** 2)
    ss_tot = torch.sum((y_test - torch.mean(y_test)) ** 2)
    r2_score = 1 - ss_res / ss_tot

print(f"\n=== KẾT QUẢ ===")
print(f"Test Loss (MSE): {test_loss.item():.4f}")
print(f"R² Score: {r2_score.item():.4f}")
print(f"Learned W: {model.linear.weight.item():.4f}")
print(f"Learned b: {model.linear.bias.item():.4f}")

# === BƯỚC 6: TRỰC QUAN HÓA ===
plt.figure(figsize=(15, 5))

# Subplot 1: Loss curve
plt.subplot(1, 3, 1)
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss qua các Epochs')
plt.grid(True, alpha=0.3)

# Subplot 2: Training data và đường hồi quy
plt.subplot(1, 3, 2)
plt.scatter(X_train.numpy(), y_train.numpy(),
            alpha=0.5, label='Dữ liệu huấn luyện')
X_range = torch.linspace(X_train.min(),
           X_train.max(), 100).reshape(-1, 1)
with torch.no_grad():
    y_range = model(X_range)
plt.plot(X_range.numpy(), y_range.numpy(),
         'r-', linewidth=2, label='Đường hồi quy')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Mô hình hồi quy tuyến tính')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 3: Actual vs Predicted
plt.subplot(1, 3, 3)
with torch.no_grad():
    plt.scatter(y_test.numpy(), y_pred_test.numpy()
, alpha=0.6)
    plt.plot([y_test.min(), y_test.max()],
             [y_test.min(), y_test.max()],
             'r--', linewidth=2)
plt.xlabel('Giá trị thực tế')
plt.ylabel('Giá trị dự đoán')
plt.title('Actual vs Predicted (Test Set)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Ưu điểm của PyTorch: Code này có thể dễ dàng mở rộng để xây dựng các mô hình phức tạp hơn (neural networks), tận dụng GPU, và tùy chỉnh quá trình huấn luyện theo nhu cầu cụ thể.

Kết luận và hướng phát triển

Hồi quy tuyến tính - Nền tảng vững chắc

Hồi quy tuyến tính không chỉ là một thuật toán đơn giản mà là nền tảng quan trọng trong thống kê và học máy. Hiểu rõ hồi quy tuyến tính giúp bạn:

- Nắm vững các khái niệm cơ bản: tối ưu hóa, hàm mất mát, gradient descent
- Phát triển tư duy phân tích dữ liệu và mô hình hóa vấn đề
- Chuẩn bị tốt cho việc học các mô hình phức tạp hơn
- Áp dụng vào nhiều bài toán thực tế với hiệu quả cao

Python - Công cụ mạnh mẽ

Python cùng với các thư viện sklearn và PyTorch cung cấp môi trường lý tưởng để thực hành và triển khai mô hình học máy:

- **Sklearn:** Đơn giản, nhanh chóng, phù hợp cho nguyên mẫu và sản xuất
- **PyTorch:** Linh hoạt, mạnh mẽ, phù hợp cho nghiên cứu và mô hình phức tạp
- Cộng đồng lớn, tài liệu phong phú, dễ dàng tìm kiếm giải pháp
- Tích hợp tốt với các công cụ khoa học dữ liệu khác

Hướng phát triển tiếp theo

Sau khi nắm vững hồi quy tuyến tính, bạn có thể tiếp tục học:

- **Hồi quy logistic:** Phân loại nhị phân và đa lớp
- **Hồi quy đa thức:** Mô hình hóa mối quan hệ phi tuyến
- **Regularization:** Ridge, Lasso, Elastic Net để tránh overfitting
- **Neural Networks:** Mạng nơ-ron sâu cho các bài toán phức tạp
- **Ensemble Methods:** Random Forest, Gradient Boosting
- **Deep Learning:** CNN, RNN, Transformers cho các ứng dụng tiên tiến

Tài nguyên học tập thêm

- Sách: "Pattern Recognition and Machine Learning" - Christopher Bishop
- Khóa học: Coursera Machine Learning - Andrew Ng
- Thực hành: Kaggle competitions và datasets
- Documentation: Sklearn, PyTorch, TensorFlow

Lời khuyên cuối cùng

Học máy là một hành trình dài đòi hỏi kiên trì và thực hành thường xuyên. Đừng vội vàng, hãy dành thời gian hiểu sâu từng khái niệm. Thực hành coding với nhiều bộ dữ liệu khác nhau để củng cố kiến thức. Chúc bạn thành công trên con đường khoa học dữ liệu!