UỶ BAN NHÂN DÂN TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

-----□🕮□-----

BÁO CÁO CUỐI KỲ

MÔN HỌC:
ĐỒ ÁN CHUYÊN NGÀNH (841476)

Giảng viên: TS. Nguyễn Quốc Huy.

Năm học: 2024 - 2025.

Thực hiện: Nhóm 9

Nguyễn Anh Tuấn – 3120410586

Hà Quốc Vĩ – 3120410607

Kiều Minh Tuyển – 3120410598

TPHCM, ngày 22 tháng 11 năm 2024

MỤC LỤC

LỜI CẢM ƠN

Lời đầu tiên chúng em xin chân thành cảm ơn các thầy cô trong khoa Công Nghệ Thông Tin của trường Đại Học Sài Gòn, những người đã trực tiếp giảng dạy cung cấp kiến thức và phương pháp trong 4 năm qua, đó là những nền tảng cơ bản, là những hành trình vô cùng quý giá để chúng em có thể bước vào sự nghiệp trong tương lai.

Để có được kết quả này chúng em xin đặc biệt gửi lời cảm ơn chân thành nhất tới thầy **Nguyễn Quốc Huy** đã quan tâm giúp đỡ, vạch kế hoạch hướng dẫn chúng em hoàn thành một cách tốt nhất báo cáo môn học trong thời gian qua. Cuối cùng chúng em xin chân thành cảm ơn gia đình, bạn bè đã động viên chia sẻ, giúp đỡ nhiệt tình và đóng góp nhiều ý kiến quý báu để chúng em có thể hoàn thành báo cáo môn học này. Trong quá trình hoàn thành báo cáo, vì chưa có kinh nghiệm thực tế chỉ dựa vào lý thuyết đã học, cùng với thời gian có hạn nên báo cáo sẽ không tránh khỏi những sai sót. Kính mong nhận được sự góp ý, nhận xét từ các thầy cô để kiến thức của chúng em ngày càng hoàn thiện hơn và rút ra được nhiều kinh nghiệm bổ ích có thể áp dụng vào thực tiễn một cách hiệu quả trong tương lai.

Chúng em xin chân thành cảm ơn!

CHƯƠNG I: HỒI QUY TUYẾN TÍNH

1.1. TỔNG QUAN VỀ HỒI QUY TUYẾN TÍNH

Hồi quy tuyến tính (Linear regression) là một loại phân tích thống kê được sử dụng để dự đoán mối quan hệ giữa hai biến. Nó giả định mối quan hệ tuyến tính giữa một hoặc nhiều biến độc lập (X) và biến phụ thuộc (y) và nhằm mục đích tìm ra đường phù hợp nhất mô tả mối quan hệ. Đường này được xác định bằng cách giảm thiểu tổng bình phương chênh lệch giữa giá trị dự đoán và giá trị thực tế. Hồi quy tuyến tính thường được sử dụng trong nhiều lĩnh vực, bao gồm kinh tế, tài chính và khoa học xã hội, để phân tích và dự đoán xu hướng trong dữ liệu.

Biến mà ta muốn dự đoán được gọi là biến phụ thuộc. Biến mà ta đang sử dụng để dự đoán giá trị của biến khác được gọi là biến độc lập. Đó là một cách tiếp cận dựa trên ranh giới *(Boundary-Based Approach)*. Thuật toán thuộc nhóm học có giám sát *(Supervised learning)*.

Thuật toán này giải thích mối quan hệ tuyến tính giữa biến phụ thuộc *(đầu ra)* Y và biến độc lập *(dự đoán)* X bằng cách sử dụng mô hình $Y = \beta_0 + \beta_1 X + \varepsilon$. Mục tiêu của thuật toán hồi quy tuyến tính là lấy các giá trị tốt nhất của β_0 và β_1 để tìm ra mô hình phù hợp nhất. Đường phù hợp nhất là đường có ít sai số nhất, nghĩa là sai số giữa giá trị dự đoán và giá trị thực tế phải ở mức tối thiểu hay còn gọi là Best Fit-line.

Sai số ngẫu nhiên *(phần dư)*: Trong hồi quy, chênh lệch giữa giá trị thực của biến phụ thuộc Y_i và giá trị dự đoán *(dự đoán)* được gọi là phần dư. Phần dư đo lường khoảng cách từ một điểm đến đường hồi quy. $\varepsilon = \widehat{y_i} - y_i$

Với $\widehat{y_i}$ là kết quả dự đoán của tập dữ liệu X_i qua mô hình $Y = \beta_0 + \beta_1 X + \varepsilon$

Tầm quan trọng của hồi quy tuyến tính:

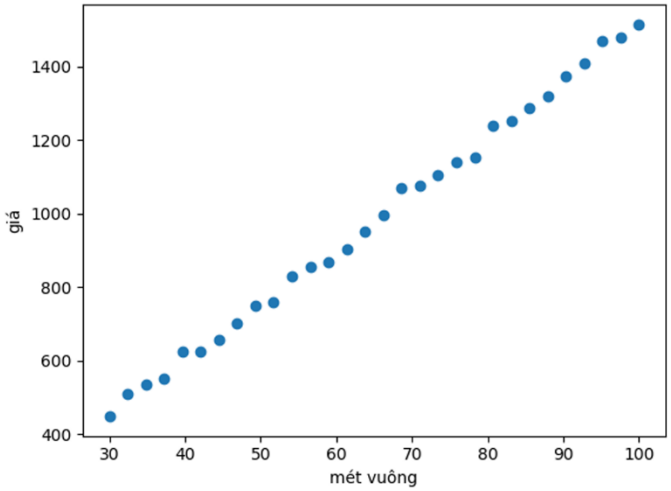
- Tính đơn giản và dễ hiểu: Đây là một khái niệm tương đối dễ hiểu và dễ áp dụng. Mô hình hồi quy tuyến tính đơn giản thu được là một phương trình đơn giản cho thấy một biến ảnh hưởng như thế nào đến một biến khác. Điều này giúp việc giải thích và tin cậy vào kết quả dễ dàng hơn so với các mô hình phức tạp hơn.

- Khả năng dự đoán: Hồi quy tuyến tính cho phép bạn dự đoán các giá trị trong tương lai dựa trên dữ liệu hiện có. Ví dụ: bạn có thể sử dụng nó để dự đoán doanh số bán hàng dựa trên chi tiêu tiếp thị hoặc giá nhà dựa trên mét vuông.
- Nền tảng cho các kỹ thuật khác: Nó đóng vai trò là nền tảng cho nhiều phương pháp học máy và khoa học dữ liệu khác. Ngay cả các thuật toán phức tạp cũng thường dựa vào hồi quy tuyến tính làm điểm bắt đầu hoặc cho mục đích so sánh.
- Khả năng ứng dụng rộng rãi: Hồi quy tuyến tính có thể được sử dụng trong nhiều lĩnh vực khác nhau, từ tài chính, kinh tế đến khoa học và khoa học xã hội. Đây là một công cụ linh hoạt để khám phá mối quan hệ giữa các biến trong nhiều tình huống thực tế.
- Về bản chất, hồi quy tuyến tính cung cấp nền tảng vững chắc để hiểu dữ liệu và đưa ra dự đoán. Đó là một kỹ thuật nền tảng mở đường cho các phương pháp phân tích dữ liệu nâng cao hơn.

1.2. MÔ HÌNH HỒI QUY TUYẾN TÍNH ĐƠN GIẢN

Trong hồi quy tuyến tính đơn giản, có ***1 biến độc lập*** và ***1 biến phụ thuộc***. Mô hình ước tính độ dốc và điểm giao cắt của đường phù hợp nhất, thể hiện mối quan hệ giữa các biến. Độ dốc biểu thị sự thay đổi của biến phụ thuộc đối với mỗi thay đổi đơn vị trong biến độc lập, trong khi điểm chặn biểu thị giá trị dự đoán của biến phụ thuộc khi biến độc lập bằng 0.

Hồi quy tuyến tính cho thấy mối quan hệ tuyến tính giữa biến độc lập (dự báo) tức là trục X và biến phụ thuộc (đầu ra) tức là trục Y. Nếu có một biến đầu vào X (biến độc lập), hồi quy tuyến tính như vậy được gọi là hồi quy tuyến tính đơn giản.



Đây là dạng đơn giản nhất của hồi quy tuyến tính và nó chỉ liên quan đến một biến độc lập và một biến phụ thuộc. Phương trình hồi quy tuyến tính đơn giản là:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Trong đó:

Y *: Biến phụ thuộc*

X *: Biến độc lập*

β_0 *: Hệ số chặn – giá trị mà biến phụ thuộc y sẽ nhận được khi biến độc lập x bằng 0. Hệ số chặn thường giúp mô hình hồi quy bắt đầu từ một điểm không phải là gốc tọa độ (0,0).*

β_1 *: Độ dốc của đường hồi quy – cho biết mỗi đơn vị thay đổi của x sẽ làm thay đổi giá trị của y bao nhiêu đơn vị. Hệ số góc thể hiện mối quan hệ tỷ lệ giữa các biến. Nếu* β_1 *là dương, điều này có nghĩa là khi x tăng, y cũng tăng. Ngược lại, nếu β1 là âm, thì khi x tăng, y giảm.*

ε *: Phần dư – là sự khác biệt giữa giá trị thực tế của y và giá trị dự đoán từ mô hình. Phần dư cho phép mô hình hồi quy không hoàn hảo, vì thường có các yếu tố không thể giải thích hoàn toàn chỉ bằng biến độc lập x.*

1.3. MÔ HÌNH HỒI QUY TUYẾN TÍNH ĐA BIẾN

Hồi quy tuyến tính đa biến giả định có mối quan hệ tuyến tính giữa hai hoặc nhiều biến độc lập và một biến phụ thuộc.

Phương trình hồi quy tuyến tính nhiều biến là:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon$$

Trong đó:

Y *: Biến phụ thuộc*

X_i *: Biến độc lập*

β_0 *: Hệ số chặn – giá trị mà biến phụ thuộc y sẽ nhận được khi biến độc lập x bằng 0. Hệ số chặn thường giúp mô hình hồi quy bắt đầu từ một điểm không phải là gốc tọa độ (0,0).*

β_1 *: Độ dốc của đường hồi quy – cho biết mỗi đơn vị thay đổi của x sẽ làm thay đổi giá trị của y bao nhiêu đơn vị. Hệ số góc thể hiện mối quan hệ tỷ lệ giữa các biến. Nếu* β_1 *là dương, điều này có nghĩa là khi x tăng, y cũng tăng. Ngược lại, nếu β1 là âm, thì khi x tăng, y giảm.*

ε *: Phần dư – là sự khác biệt giữa giá trị thực tế của y và giá trị dự đoán từ mô hình. Phần dư cho phép mô hình hồi quy không hoàn hảo, vì thường có các yếu tố không thể giải thích hoàn toàn chỉ bằng biến độc lập x.*

Mô hình hồi quy tuyến tính đa biến có thể được biểu diễn dưới dạng mặt phẳng (2 chiều) hoặc siêu phẳng (ở chiều cao hơn).

1.4. LOSS FUNCTION

Như đã đề cập ở phần Định nghĩa: *"Đường phù hợp nhất là đường có ít sai số nhất, nghĩa là sai số giữa giá trị dự đoán và giá trị thực tế phải ở mức tối thiểu hay còn gọi là Best Fit-line."*

Vậy đường phù hợp nhất là gì?

Tồn tại vô số dòng phù hợp với dữ liệu. Trong số vô số khả năng, chúng ta cần tìm một đường có sai số tối thiểu – đường tìm được chính là Best Fit-line.

Vậy việc tìm mô hình $Y = \beta_0 + \beta_1 X + \varepsilon$ trở thành việc tìm β_0 và β_1 sao cho sai số của mô hình đối với các điểm dữ liệu là nhỏ nhất.

Việc việc tìm β_0 và β_1 có thể đơn giản nếu làm bằng mắt nhưng máy tính không biết điều đấy, nên ban đầu giá trị được chọn ngẫu nhiên ví dụ $\beta_0 = 0, \beta_1 = 1$ sau đấy được chỉnh dần. Khi đó ta có mô hình dự đoán $Y = X$. Lúc này ta cần 1 hàm để đánh giá mô hình tìm được với các điểm dữ liệu là tốt hay không. Với mỗi điểm dữ liệu (X_i, Y_i) độ chênh lệch giữa giá trị thật và giá trị dự đoán được tính bằng $\frac{1}{2} * (\hat{y_i} - y_i)^2$.

Và độ chênh lệch trên toàn bộ dữ liệu được tính bằng tổng chênh lệch của từng điểm

$$J = \frac{1}{2} * \frac{1}{N} * \sum_{i=1}^{N} (\hat{y_i} - y_i)^2$$

(N là số điểm dữ liệu).

⇒ Ta có thể thấy J luôn không âm, và J càng nhỏ thì đường thẳng càng gần các điểm dữ liệu, nếu $J = 0$ thì sẽ qua tất cả các điểm dữ liệu.

Hàm J chính là hàm chi phí sai số bình phương trung bình *(MSE – Mean Squared Error)*. Chúng ta sẽ dùng nó để làm Loss function – hàm để đánh giá xem bộ tham số hiện tại có tốt với dữ liệu không.

Ngoài MSE ra chúng ta có thể dùng các hàm khác để làm Loss function cho mô hình hồi quy tuyến tính như: Mean Absolute Error (MAE), Huber Loss, Log-Cosh Loss, Quantile Loss,… Mỗi loss function này có đặc trưng riêng và sẽ phù hợp cho từng loại dữ liệu khác nhau trong bài toán hồi quy tuyến tính.

Ngoài ra có sự khác biệt nhỏ giữa hàm mất mát *(loss function)* và hàm chi phí *(cost function)* là về sai số trong quá trình đào tạo các mô hình học máy, vì hàm mất mát đề cập đến sai số của một ví dụ đào tạo, trong khi hàm chi phí tính toán sai số trung bình trên toàn bộ tập huấn luyện.

Sau khi đã có Loss Function bài toán tìm mô hình gần các điểm dữ liệu nhất trở thành tìm β_0 và β_1 sao cho hàm J đạt giá trị nhỏ nhất. Giờ cần một thuật toán để tìm giá trị nhỏ nhất của hàm $J(\beta_0, \beta_1)$ Đó chính là thuật toán Gradient Descent.

1.5. THUẬT TOÁN GRADIENT DESCENT

Giảm dần độ dốc (Gradient Descent) là một trong những thuật toán tối ưu hóa hàm chi phí (cost function) để đạt được lời giải tối thiểu tối ưu. Để tìm ra giải pháp tối ưu, chúng ta cần giảm hàm chi phí cho tất cả các điểm dữ liệu. Điều này được thực hiện bằng cách cập nhật lặp đi lặp lại các giá trị của hệ số độ dốc β_0 và hệ số không đổi β_1 cho đến khi chúng ta có được nghiệm tối ưu cho hàm tuyến tính.

Mô hình hồi quy tối ưu hóa thuật toán giảm độ dốc để cập nhật các hệ số của đường bằng cách giảm hàm chi phí bằng cách chọn ngẫu nhiên các giá trị hệ số và sau đó cập nhật lặp lại các giá trị hệ số để đạt hàm chi phí tối thiểu.

Để dễ hiểu, ta hãy tưởng tượng một cái hố hình chữ U. Ta đang đứng ở điểm cao nhất trong hố và động cơ của bạn là chạm tới đáy hố. Giả sử có một kho báu ở dưới đáy hố và bạn chỉ có thể đi một số bước riêng biệt để chạm tới đáy. Nếu bạn chọn thực hiện từng bước một, cuối cùng bạn sẽ chạm tới đáy hố nhưng việc này sẽ mất nhiều thời gian hơn. Nếu bạn quyết định thực hiện các bước lớn hơn mỗi lần, bạn có thể chạm tới đáy sớm hơn nhưng có khả năng là bạn có thể vượt quá đáy hố và thậm chí không ở gần đáy. Trong thuật toán giảm độ dốc, số bước bạn thực hiện có thể được coi là tốc độ học tập và điều này quyết định tốc độ hội tụ của thuật toán đến mức tối thiểu.

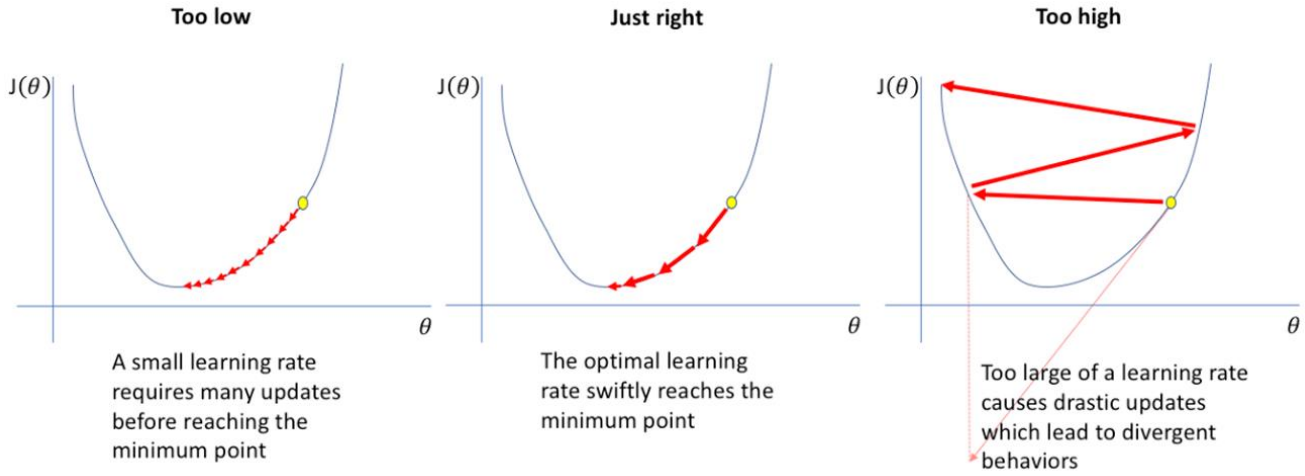
Thuật toán:

1. Khởi tạo giá trị $x = x_0$ tùy ý
2. Gán $x = x - learning\_rate * f'(x)$, với $learning\_rate > 0$
3. Tính lại $f(x)$: Nếu $f(x)$ đủ nhỏ thì dừng lại, ngược lại tiếp tục bước 2

Thuật toán sẽ lặp lại bước 2 một số lần đủ lớn (100 hoặc 1000 lần tùy vào bài toán và hệ số *learning_rate*) cho đến khi *f(x)* đạt giá trị đủ nhỏ.

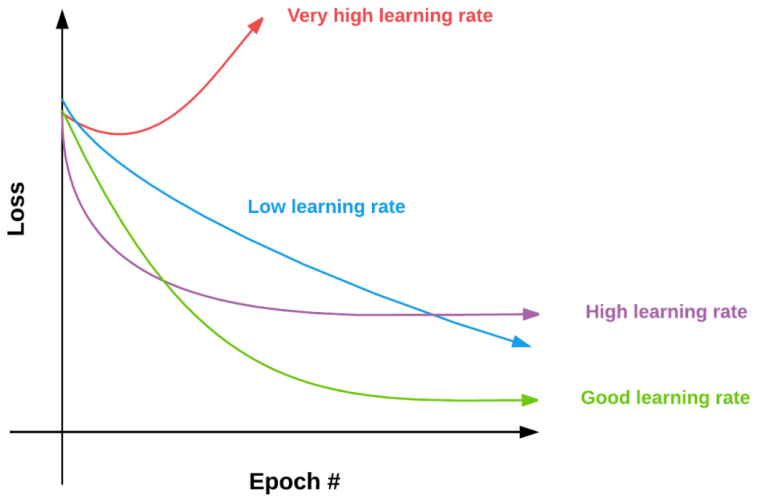
Việc chọn hệ số *learning_rate* cực kì quan trọng, có 3 trường hợp:

- Nếu *learning_rate* nhỏ: mỗi lần hàm số giảm rất ít nên cần rất nhiều lần thực hiện bước 2 để hàm số đạt giá trị nhỏ nhất.
- Nếu *learning_rate* hợp lý: sau một số lần lặp bước 2 vừa phải thì hàm sẽ đạt giá trị đủ nhỏ.
- Nếu *learning_rate* quá lớn: sẽ gây hiện tượng *overshoot* và không bao giờ đạt được giá trị nhỏ nhất của hàm.



Ảnh 2: Các giá trị learning_rate khác nhau

Cách tốt nhất để kiểm tra *learning_rate* hợp lý hay không là kiểm tra giá trị hàm *f(x)* sau mỗi lần thực hiện bước 2 bằng cách vẽ đồ thị với trục x là số lần thực hiện bước 2, trục y là giá trị loss function tương ứng ở bước đấy.



Ảnh 3: Loss là giá trị của hàm cần tìm giá trị nhỏ nhất, Epoch ở đây là số cần thực hiện bước 2

Áp dụng vào bài toán:

Ta cần tìm giá trị nhỏ nhất của hàm:

$$J(\beta_0, \beta_1) = \frac{1}{2} * \frac{1}{N} * \sum_{i=1}^{N} (\hat{y_i} - y_i)^2 = \frac{1}{2} * \frac{1}{N} * \sum_{i=1}^{N} (\beta_0 + \beta_1 X - y_i)^2$$

Tuy nhiên do điểm cực tiểu hàm *f(x)* giống với hàm $\frac{f(x)}{N}$ (với N>0) nên ta sẽ đi tìm điểm cực tiểu của hàm:

$$J(\beta_0, \beta_1) = \frac{1}{2} * \sum_{i=1}^{N} (\beta_0 + \beta_1 X - y_i)^2$$

⇒ Ngoài sử dụng Gradient Descent ta hoàn toàn có thể dùng đại số tuyến tính để tìm giá trị nhỏ nhất của hàm *J*.

Việc quan trọng nhất của thuật toán gradient descent là tính đạo hàm của hàm số nên giờ ta sẽ đi tính đạo hàm của loss function theo từng biến.

$$\frac{dJ}{d\beta_0} = \sum_{i=1}^{N} \beta_0 + \beta_1 x_i - y_i$$

$$\frac{dJ}{d\beta_1} = \sum_{i=1}^{N} x_i(\beta_0 + \beta_1 x_i - y_i)$$

Biểu diễn bài toán dưới dạng ma trận:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

$$\hat{y} = X * \beta = \begin{bmatrix} \beta_0 + \beta_1 x_1 \\ \beta_0 + \beta_1 x_2 \\ \cdots \\ \beta_0 + \beta_1 x_n \end{bmatrix}$$

1.6. SO SÁNH LOSS FUNCTION MAE VÀ MSE

Mean Absolute Error *(MAE)* hay còn được gọi là L1 Loss là một loss function được sử dụng cho các mô hình hồi quy, đặc biệt cho các mô hình hồi quy tuyến tính. MAE được tính bằng tổng các trị tuyệt đối của hiệu giữa giá trị thực y_i *(target)* và giá trị mà mô hình của chúng ra dự đoán $\hat{y_i}$ *(predicted).*

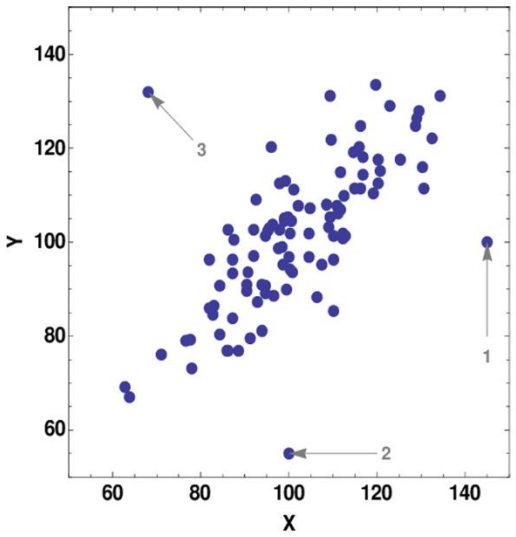
$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}|$$

Mean Square Error *(MSE)* hay còn được gọi là L2 Loss là một loss function cũng được sử dụng cho các mô hình hồi quy, đặc biệt là các mô hình hồi quy tuyến tính. MSE được tính bằng tổng các bình phương của hiệu giữa giá trị thực y_i *(target)* và giá trị mà mô hình của chúng ra dự đoán $\hat{y_i}$ *(predicted).*

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

Để tìm điểm cực tiểu của các hàm số, thông thường cách đơn giản nhất chúng ta nghĩ đến chính là tìm đạo hàm của hàm số rồi tìm điểm mà tại đó đạo hàm của hàm số bằng 0. Như vậy sẽ có 1 bước đạo hàm, hiển nhiên tìm đạo hàm của MSE sẽ đơn giản hơn rất nhiều so với MAE. Tuy

nhiên, MAE thì đem lại kết quả tốt hơn đối với các dữ liệu có Outlier *(điểm dị biệt – là những điểm có giá trị khác xa so với phần còn lại của dữ liệu).*



Đầu tiên hãy để ý đến MSE, ta có $y_i - \hat{y_i} = \varepsilon$, ε^2 sẽ vô cùng lớn nếu $\varepsilon > 1$. Nếu bậc của ε càng lớn thì giá trị hàm Loss cũng càng lớn hơn. Vì vậy nếu như chúng ta có 1 Outlier trong bộ dữ liệu, giá trị của ε sẽ vô cùng lớn, có thể tiến tới ∞ và khi đó, ε^2 có thể sẽ tiến tới ∞, khi đó giá trị MSE sẽ cực kì lớn.

Ngược lại với MSE, đối với MAE nếu ta có giá trị ε lớn ($\varepsilon > 1$) thì $|\varepsilon|$ vẫn sẽ lớn, nhưng hiển nhiên nhỏ hơn nhiều so với ε^2.

Có thể nhận thấy khi tối ưu loss function, L2 *(MSE)* bị ảnh hưởng mạnh hơn với các điểm outlier và model sẽ bị kéo về phía outlier hơn. Do đó MSE bị ảnh hưởng bởi outlier và L1 tốt hơn đối với các dữ liệu có outlier.

Tuy vậy đạo hàm của MAE *(hay L1 Loss)* tính toán khó hơn nhiều so với MSE *(hay L2 Loss)*.

1.7. ÁP DỤNG HỒI QUY TUYẾN TÍNH ĐỂ TẠO 1 MÔ HÌNH DỰ ĐOÁN

1.7.1. Dataset

Temperature	Tourists	SunnyDays	PredictedSales
91	998	4	89.8
87	1256	7	90.2
86	791	6	81.1
88	705	5	83
92.8	1089	3	90.9
95.2	1135	6	119
93.3	1076	4	94.9
97.7	1198	7	132.4

Ngữ cảnh:

Tại một thị trấn phía nam gần một bãi biển xinh đẹp, Tom – người quản lý của một siêu thị đã phát hiện ra mối quan hệ giữa nhiệt độ cao trung bình hàng ngày hàng tuần và việc bán kem vào mùa hè. Nhưng dự đoán của anh ấy ban đầu không đủ tốt. Sau một số nghiên cứu, anh ấy đã thu thập thông tin bổ sung như số lượng khách du lịch *(lấy từ các khách sạn)* và số ngày nắng trong một tuần.

Với dataset trên ta có:

- Các biến độc lập *(X)* là: Temperature *(Nhiệt độ trung bình trong tuần)*, Tourists *(Số lượng khách du lịch)*, SunnyDays *(Số ngày nắng trong tuần)*
- Biến phụ thuộc *(Y)* là: PredictedSales *(Doanh thu)*
- Số lượng mẫu *(N)*: 8

1.7.2. Mẫu cần dự đoán

Temperature	Tourists	SunnyDays
93	1212	6

1.7.3. Xây dựng mô hình bằng đại số tuyến tính

Ta có mô hình hồi quy tuyến tính tổng quát với 3 biến như sau:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$$

Loss function:

$$J(w_0, w_1, w_2, w_3) = \frac{1}{2} * \sum_{i=1}^{7} (w_0 + w_1x_1 + w_2x_2 + w_3x_3 - y_i)^2$$

Đạo hàm theo từng biến w_0, w_1, w_2, w_3

$$\frac{dJ}{dw_0} = \sum_{i=1}^{7} w_0 + w_1x_1^i + w_2x_2^i + w_3x_3^i - y_i$$

$$\frac{dJ}{dw_1} = \sum_{i=1}^{7} x_1^i(w_0 + w_1x_1^i + w_2x_2^i + w_3x_3^i - y_i)$$

$$\frac{dJ}{dw_2} = \sum_{i=1}^{7} x_2^i(w_0 + w_1x_1^i + w_2x_2^i + w_3x_3^i - y_i)$$

$$\frac{dJ}{dw_3} = \sum_{i=1}^{7} x_3^i(w_0 + w_1x_1^i + w_2x_2^i + w_3x_3^i - y_i)$$

Tính các tổng sau:

$\sum_{i=1}^{7} x_1^i = 731, \quad \sum_{i=1}^{7} x_2^i = 8248, \quad \sum_{i=1}^{7} x_3^i = 42, \quad \sum_{i=1}^{7} y_i = 781.3$

$\sum_{i=1}^{7} (x_1^i)^2 = 66915.06, \quad \sum_{i=1}^{7} (x_2^i)^2 = 8763372, \quad \sum_{i=1}^{7} (x_3^i)^2 = 236$

$\sum_{i=1}^{7} x_1^i x_2^i = 756702.6, \quad \sum_{i=1}^{7} x_1^i x_3^i = 3835.7, \quad \sum_{i=1}^{7} x_2^i x_3^i = 43822$

$\sum_{i=1}^{7} x_1^i y_i = 71851.77, \quad \sum_{i=1}^{7} x_2^i y_i = 820359.4, \quad \sum_{i=1}^{7} x_3^i y_i = 4185.3$

Khi đó ta có:

$$\frac{dJ}{dw_0} = 8w_0 + w_1\sum_{i=1}^{7} x_1^i + w_2\sum_{i=1}^{7} x_2^i + w_3\sum_{i=1}^{7} x_3^i - \sum_{i=1}^{7} y_i$$
$$= 8w_0 + 731w_1 + 8248w_2 + 42w_3 - 781.3$$

$$\frac{dJ}{dw_1} = \sum_{i=1}^{7} x_1^i w_0 + w_1 {x_1^i}^2 + w_2 x_1^i x_2^i + w_3 x_1^i x_3^i - x_1^i y_i$$
$$= w_0\sum_{i=1}^{7} x_1^i + w_1\sum_{i=1}^{7} {x_1^i}^2 + w_2\sum_{i=1}^{7} x_1^i x_2^i + w_3\sum_{i=1}^{7} x_1^i x_3^i - \sum_{i=1}^{7} x_1^i y_i$$
$$= 731w_0 + 66915.06w_1 + 756702.6w_2 + 3835.7w_3 - 71851.77$$

$$\frac{dJ}{dw_2} = \sum_{i=1}^{7} x_2^i w_0 + w_2 {x_2^i}^2 + w_1 x_2^i x_1^i + w_3 x_2^i x_3^i - x_2^i y_i$$
$$= w_0\sum_{i=1}^{7} x_2^i + w_1\sum_{i=1}^{7} x_2^i x_1^i + w_2\sum_{i=1}^{7} {x_2^i}^2 + w_3\sum_{i=1}^{7} x_2^i x_3^i - \sum_{i=1}^{7} x_2^i y_i$$
$$= 8248w_0 + 756702.6w_1 + 8763372w_2 + 43822w_3 - 820359.4$$

$$\frac{dJ}{dw_3} = \sum_{i=1}^{7} x_3^i w_0 + w_3 {x_3^i}^2 + w_1 x_3^i x_1^i + w_2 x_3^i x_2^i - x_3^i y_i$$

$$= w_0 \sum_{i=1}^{7} x_3^i + w_1 \sum_{i=1}^{7} x_3^i x_1^i + w_2 \sum_{i=1}^{7} x_3^i x_2^i + w_3 \sum_{i=1}^{7} {x_3^i}^2 - \sum_{i=1}^{7} x_3^i y_i$$

$$= 42w_0 + 3835.7w_1 + 43822w_2 + 236w_3 - 4185.3$$

Giải hệ 4 phương trình để tìm các hệ số:

$$\begin{cases} 8w_0 + 731w_1 + 8248w_2 + 42w_3 = 781.3 \\ 731w_0 + 66915.06w_1 + 756702.6w_2 + 3835.7w_3 = 71851.77 \\ 8248w_0 + 756702.6w_1 + 8763372w_2 + 43822w_3 = 820359.4 \\ 42w_0 + 3835.7w_1 + 43822w_2 + 236w_3 = 4185.3 \end{cases}$$

Giải hệ phương trình trên ta được bộ nghiệm:

$$w_0 = -295.47332415$$

$$w_1 = 3.97541435$$

$$w_2 = -0.00134734$$

$$w_3 = 5.95646553$$

Vậy ta có model:

$$y = -295.47332415 + 3.97541435x_1 - 0.00134734x_2 + 5.95646553x_3$$

1.7.4. Xây dựng mô hình bằng hàm excel

Viết lại tên mỗi cột thành x, y. Với x là các biến độc lập và y là biến phụ thuộc:

y	x1	x2	x3
89.8	91	998	4
90.2	87	1256	7
81.1	86	791	6
83	88	705	5
90.9	92.8	1089	3

119	95.2	1135	6
94.9	93.3	1076	4
132.4	97.7	1198	7

Nhân mỗi giá trị của cột x với nhau và với cột y: $= \mathrm{X1} * \mathrm{X2}$

x1x2	x1x3	x2x3	x1y	x2y	x3y
90818	364	3992	8171.8	89620.4	359.2
109272	609	8792	7847.4	113291.2	631.4
68026	516	4746	6974.6	64150.1	486.6
62040	440	3525	7304	58515	415
101059.2	278.4	3267	8435.52	98990.1	272.7
108052	571.2	6810	11328.8	135065	714
100390.8	373.2	4304	8854.17	102112.4	379.6
117044.6	683.9	8386	12935.48	158615.2	926.8

Tính tổng của mỗi cột ở hàng cuối cùng với công thức: $= SUM(X)$. Trong đó X là vùng dữ liệu cần tính tổng:

y	x1	x2	x3	x1x2	x1x3	x2x3	x1y	x2y	x3y
781.3	731	8248	42	756702.6	3835.7	43822	71851.77	820359.4	4185.3

Chúng ta sẽ viết ma trận theo công thức như sau

$$X^T X = \begin{bmatrix} n & \sum X_1 & \sum X_2 & \sum X_3 \\ \sum X_1 & \sum X_1^2 & \sum X_1 X_2 & \sum X_1 X_3 \\ \sum X_2 & \sum X_1 X_2 & \sum X_2^2 & \sum X_2 X_3 \\ \sum X_3 & \sum X_1 X_3 & \sum X_2 X_3 & \sum X_3^2 \end{bmatrix}$$

Trong đó:

- n là số mẫu dữ liệu.
- $\sum X_1$, $\sum X_2$, và $\sum X_3$ là tổng các giá trị của ba biến đầu vào X_1, X_2, và X_3.
- $\sum X_1^2$, $\sum X_2^2$, và $\sum X_3^2$ là tổng các bình phương của X_1, X_2, và X_3.
- $\sum X_1 X_2$, $\sum X_1 X_3$, và $\sum X_2 X_3$ là tổng của tích các cặp giá trị giữa các biến.

– Giá trị n có thể tính với công thức $COUNT(X)$ dùng để đếm số lượng ô dữ liệu trong vùng được chọn X.

– Còn tổng bình phương và tổng tích, chúng ta sẽ sử dụng hàm $SUMPRODUCT(X1, X2)$ có chức năng nhân các thành phần của 2 mảng (X1, X2) rồi cộng tổng

– Ma trận X^TX giúp mô tả các mối quan hệ giữa các biến độc lập. Khi tính toán các hệ số hồi quy, việc nhân ma trận này với vector Y giúp phân tích ảnh hưởng của từng biến độc lập đến biến phụ thuộc.

Matrix			
8	731	8248	42
731	66915.06	756702.6	3835.7
8248	756702.6	8763372	43822
42	3835.7	43822	236

Sau khi có được ma trận X^TX, ta cần sử dụng hàm $MINVERSE(XTX)$ có chức năng trả về ma trận nghịch đảo, sẽ có kết quả $(X^TX)^{-1}$ như sau:

Inverse Matrix			
87.40263533	-1.01713865	0.009531134038	-0.7929886408
-1.01713865	0.01254291037	-0.0001610523933	0.007061949082

0.009531134038	-0.0001610523933	0.000006196088191	-0.0002291692429
-0.7929886408	0.007061949082	-0.0002291692429	0.07313838722

Tiếp theo, ta cần phải tính ma trận X^TY với các hàm SUM và $SUMPRODUCT$:

$$X^TY = \begin{bmatrix} \sum Y \\ \sum X_1Y \\ \sum X_2Y \\ \sum X_3Y \end{bmatrix}$$

Trong đó:

- $\sum Y$ là tổng các giá trị của biến phụ thuộc Y.
- $\sum X_1Y$, $\sum X_2Y$, và $\sum X_3Y$ lần lượt là tổng của tích giữa các giá trị của X_1, X_2, X_3 với Y.

Sử dụng công thức trên ta có được:

781.3
71851.77
820359.4
4185.3

Để tính hệ số hồi quy, ta sẽ sử dụng công thức

$$\beta = (X^TX)^{-1}X^TY$$

Trong đó:

- X^TX là ma trận tương quan giữa các biến độc lập, thể hiện mức độ liên quan giữa các biến độc lập với nhau.
- X^TY là vector thể hiện mối quan hệ giữa các biến độc lập X và biến phụ thuộc Y.
- $(X^TX)^{-1}$ là nghịch đảo của ma trận X^TX, cho phép chúng ta giải hệ phương trình để tìm giá trị của β.

có công thức Excel: $= MMULT(X1, X2)$

Trong đó:

- X1 là ma trận X^TX
- X2 là ma trận X^TY

Hàm *MMULT* có chức năng nhân hai ma trận với điều kiện là ma trận X1 phải có số cột bằng số hàng của ma trận X2.

-295.4733241	**b0**
3.975414354	**b1**
-0.001347339	**b2**
5.956465529	**b3**

Sau khi có được các hệ số quy hồi, ta có thể sử dụng công thức

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$$

Có công thức Excel là $= X + X1 * Z1 + X2 * Z2 + X3 * Z3$

Trong đó:

- X, X1, X2, X3 là các hệ số quy hồi
- Z1, Z2, Z3 là các giá trị đầu vào

Để dự đoán Ice Cream Sale (output) và so sánh với kết quả thông tin đầu vào

Prediction	
88.77059989	89.8
90.39072561	90.2
81.08535835	81.1
83.19559268	83
89.84727235	90.9
117.1956858	119
97.80896047	94.9
133.0058049	132.4

1.7.5. Xây dựng mô hình bằng code python

1.7.5.1. Sử dụng phương pháp đại số

Bước 1: Tạo class *UseMath.py* với phương thức khởi tạo *_init_* và các phương thức *train(), predict(), predictFor(), test(), getModelInfo()* trong đó:

- Phương thức khởi tạo nhận 2 đối số là *datasetURL* và *train_size* dùng để đọc dataset và chia thành các tập *X_train, y_train, X_test, y_test* bằng các tập train chiếm $train\_size * 100\%$ dataset.
- Phương thức *training()* dùng công thức Normal Equation để tìm hệ số.

$$theta = (X^T * X)^{-1} * X^T * y$$

- Phương thức *predict()* nhận vào 1 dataframe chứa *X_test* để dự đoán và trả về kết quả dự đoán *y_pred*.
- Phương thức *predictFor()* nhận vào 1 mảng chứa *tập X* để dự đoán và trả về kết quả dự đoán *y_pred*.
- Phương thức *test()* dùng để kiểm tra kết quả mô hình dự đoán so với dữ liệu tập *y_test* bằng cách tính hàm loss MSE.
- Phương thức *getModelInfo()* dùng để in ra màn hình console các thông tin của model.

Bước 2: Trong phương thức khởi tạo viết code để đọc dataset và chia thành các tập *X_train, y_train, X_test, y_test*

```
def __init__(self, datasetURL, train_size=0.5):
    csv_handler = CSVHandler(datasetURL)
    dataframe = csv_handler.read_csv()

    # Chuẩn hóa dữ liệu từ string sang số "1,2" -> 1.2
    for col in dataframe.columns:
      if dataframe[col].dtype not in ['int64', 'float64']:
        dataframe[col] = dataframe[col].apply(lambda x: float(x.replace(',', '.')) if isinstance(x,
str) else x)

    X = dataframe.iloc[:, :-1].values  # Các biến độc lập
    y = dataframe.iloc[:, -1].values  # Biến phụ thuộc

    # Thêm cột 1 cho hệ số tự do (intercept)
    X = np.c_[np.ones(X.shape[0]), X]  # Thêm cột 1 vào đầu

    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(dataframe) * train_size)
    self.X_train = X[:split_index]  # Dữ liệu training
    self.y_train = y[:split_index]  # Dữ liệu mục tiêu training
```

```
    self.X_test = X[split_index:]  # Dữ liệu testing
    self.y_test = y[split_index:]  # Dữ liệu mục tiêu testing
```

Bước 3: Trong phương thức *training()* viết code dùng công thức Normal Equation để tìm hệ số:

```
def train(self):
    # Áp dụng công thức Normal Equation để tìm hệ số
    # theta = (X.T * X)^-1 * X.T * y
    X_transpose = self.X_train.T  # Chuyển vị của X
    self.theta = np.linalg.inv(X_transpose.dot(self.X_train)).dot(X_transpose).dot(self.y_train)
```

Bước 4: Trong phương thức *predict()* viết code để dự đoán cho 1 dataframe nhập vào:

```
  def predict(self, data_input):
    # Dự đoán y = X * theta
    y_pred = data_input.dot(self.theta)
    return y_pred
```

Bước 5: Trong phương thức *predictFor()* viết code để dự đoán cho 1 mảng nhập vào:

```
  def predictFor(self, data_input):
    data_input = np.array(data_input)
    # Chuyển data_input thành vector hàng (1, n)
    if data_input.ndim == 1:
      data_input = data_input.reshape(1, -1)
    # Thêm cột hệ số tự do (bias term) vào đầu mảng
    data_input = np.c_[np.ones(data_input.shape[0]), data_input]
    # Dự đoán y = X * theta
    y_pred = data_input.dot(self.theta)
    return y_pred
```

Bước 6: Trong phương thức *test()* viết code kiểm tra mô hình với tập dữ liệu kiểm tra:

```
  def test(self):
    # Kiểm tra mô hình với dữ liệu kiểm tra
    predictions = self.predict(self.X_test)
    loss = mean_squared_error(self.y_test, predictions)
    return loss
```

Bước 7: Trong phương thức *getModelInfo()* viết code để lấy các hệ số của mô hình và số lượng mẫu training

```
  def getModelInfo(self):
    print("Mẫu training:", len(self.y_train))
    print("Hệ số tự do (Intercept):", self.theta[0])  # Hệ số tự do là phần tử đầu tiên của theta
```

```
    print("Hệ số của các biến độc lập (Coefficients):", self.theta[1:])  # Các hệ số còn lại là của các biến độc lập
```

Download source code tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/LinearRegression/UseMath.py

1.7.5.2. Sử dụng Gradient Descent tối ưu trọng số

Bước 1: Tạo class *UseGradientDescent.py* với phương thức khởi tạo *_init_* và các phương thức *training(), test(), predict(), getModelInfo(), _meanSquaredError().*

- Phương thức khởi tạo đọc dataset training và lưu vào dataframe.
- Phương thức *training()* dùng để tạo model.
- Phương thức *test()* dùng để kiểm tra model với tập dữ liệu test bằng cách tính MSE.
- Phương thức *predict()* dùng để dự đoán cho 1 dataframe.
- Phương thức *predictFor()* dùng để dự đoán cho 1 mảng và xuất ra kết quả dự đoán.
- Phương thức *getModelInfo()* dùng để in ra màn hình console các thông tin của model.
- Phương thức *_meanSquaredError()* dùng để tính chi phí hàm mất mát theo MSE.

Bước 2: Trong phương thức khởi tạo viết code để đọc dataset và lưu vào dataframe như sau *(giả sử đã có sẵn lớp CSVHandler)*:

```
  def __init__(self, datasetURL, train_size=0.5, learning_rate=1e-4, iterations=1500):
    csv_handler = CSVHandler(datasetURL)
    dataframe = csv_handler.read_csv()

    for col in dataframe.columns:
      if dataframe[col].dtype not in ['int64', 'float64']:
        dataframe[col] = dataframe[col].apply(lambda x: float(x.replace(',', '.')) if isinstance(x, str) else x)

    X = dataframe.iloc[:, :-1].values  # Chọn tất cả các hàng và cột trừ cột cuối cùng
    y = dataframe.iloc[:, -1].values  # Chọn cột cuối cùng

    # Thêm cột 1 vào X để tính toán hệ số tự do (bias)
    X = np.c_[np.ones(X.shape[0]), X]  # Thêm cột 1 cho hệ số tự do (bias)
```

```
    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(dataframe) * train_size)
    self.X_train = X[:split_index]
    self.y_train = y[:split_index]
    self.X_test = X[split_index:]
    self.y_test = y[split_index:]

    # Khởi tạo các tham số cho Gradient Descent
    self.learning_rate = learning_rate
    self.iterations = iterations
    self.weights = np.zeros(self.X_train.shape[1])  # Khởi tạo trọng số (w) với giá trị 0
```

Bước 3: Trong phương thức *training()* viết code để xây dựng ma trận các biến độc lập X với cột đầu tiên là số 1 *(để thực hiện phép nhân với ma trận w)* và dataframe chứa cột đầu ra y.

```
  def train(self):
    # Gradient Descent để tối ưu trọng số
    m = len(self.X_train)
    loss_values = []
    count_patience = 0
    for i in range(self.iterations):
      predictions = self.X_train.dot(self.weights)  # Dự đoán y = Xw
      errors = predictions - self.y_train  # Tính sai số (error)

      # Cập nhật trọng số bằng cách sử dụng Gradient Descent
      gradient = (2/m) * self.X_train.T.dot(errors)  # Gradient of MSE
      gradient = np.clip(gradient, -1e10, 1e10)  # Giới hạn giá trị gradient trong phạm vi hợp lý tránh lỗi tràn số hoặc NaN
      self.weights -= self.learning_rate * gradient  # Cập nhật trọng số

      # Tính MSE cho mỗi lần lặp
      loss = self._meanSquaredError(self.y_train, predictions)
      loss_values.append(loss)

      threshold = 1e+5  # Ngưỡng thay đổi loss
      if i > 0 and abs(loss_values[-1] - loss_values[-2]) < threshold:
        count_patience += 1
        if count_patience >= 50:
          print(f"Dừng tại vòng lặp {i} vì loss không thay đổi đáng kể")
          break
      else:
        count_patience = 0
    return loss_values
```

Bước 4: Trong phương thức *predict()* viết code để dự đoán cho dataframe đầu vào.

```
def predict(self, data_input):
    # Dự đoán giá trị đầu ra cho dữ liệu đầu vào
    return data_input.dot(self.weights)
```

Bước 5: Trong phương thức *predictFor()* viết code để dự đoán cho 1 mảng đầu vào

```
def predictFor(self, data_input):
    data_input = np.array(data_input)
    # Chuyển data_input thành vector hàng (1, n)
    if data_input.ndim == 1:
        data_input = data_input.reshape(1, -1)
    # Thêm cột hệ số tự do (bias term) vào đầu mảng
    data_input = np.c_[np.ones(data_input.shape[0]), data_input]
    # Dự đoán y = X * theta
    y_pred = data_input.dot(self.theta)
    return y_pred
```

Bước 6: Trong phương thức *test()* viết code để kiểm tra độ chính xác của model.

```
def test(self):
    # Dự đoán trên tập kiểm tra và tính MSE
    predictions = self.predict(self.X_test)
    loss = self._meanSquaredError(self.y_test, predictions)
    return loss
```

Bước 7: Trong phương thức *_meanSquaredError()* viết code để giá trị hàm loss theo MSE

```
def _meanSquaredError(self, y_true, y_pred):
    errors = y_true - y_pred
    errors = np.clip(errors, -1e10, 1e10)  # Giới hạn giá trị lỗi trong phạm vi hợp lý tránh lỗi tràn số
    squared_errors = errors ** 2   # Tính bình phương của các lỗi
    mse = np.mean(squared_errors)  # Tính trung bình các bình phương lỗi
    return mse
```

Bước 6: Trong phương thức getModelInfo() viết code để in ra thông tin các hệ số weights vừa tìm được.

```
def getModelInfo(self):
    # In ra thông tin về mô hình
    print("Số lượng mẫu training:", len(self.y_train))
    print("Hệ số tự do (Intercept):", self.weights[0])
    print("Hệ số của các biến độc lập (Coefficients):", self.weights[1:])
```

Bước 7: Vào file main.py tạo 1 đối tượng của class *UseGradientDescent* và chạy qua các hàm trên để xem kết quả.

1.7.5.3. Sử dụng model LinearRegression thuộc thư viện sklearn

Bước 1: Tạo class *UseSklearn.py* với phương thức *training(), test(), predict(), getModelInfo()* với mục đích tương tự như *UseGradientDescent.py*

Bước 2: Trong phương thức khởi tạo viết code để đọc dataset và lưu vào dataframe như sau:

```
def __init__(self, datasetURL, train_size=0.5):
    self.model = LinearRegression()

    csv_handler = CSVHandler(datasetURL)
    dataframe = csv_handler.read_csv()

    # Chuẩn hóa dữ liệu từ string sang số "1,2" -> 1.2
    for col in dataframe.columns:
      if dataframe[col].dtype not in ['int64', 'float64']:
        dataframe[col] = dataframe[col].apply(lambda x: float(x.replace(',', '.')) if isinstance(x,
str) else x)

    X = dataframe.iloc[:, :-1]   # Chọn tất cả các hàng và cột trừ cột cuối cùng
    y = dataframe.iloc[:, -1]    # Chọn cột cuối cùng

    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(dataframe) * train_size)
    self.X_train = X.iloc[:split_index]
    self.y_train = y.iloc[:split_index]
    self.X_test = X.iloc[split_index:]
    self.y_test = y.iloc[split_index:]
```

Bước 3: Trong phương thức *training()* viết code để cho model học với dữ liệu vừa lấy được bằng lệnh *self.model.fit(self.X, self.y)*

Bước 4: Trong phương thức *predict()* viết code để dự đoán dataframe test qua câu lệnh *self.model.predict(X)*

```
  def predict(self, data_input):
    y_pred = self.model.predict(data_input)
    return y_pred
```

Bước 5: Trong phương thức *predictFor()* viết code để dự đoán mảng đầu vào

```
def predictFor(self, data_input):
    data_input = np.array(data_input)
    # Chuyển data_input thành vector hàng (1, n)
    if data_input.ndim == 1:
        data_input = data_input.reshape(1, -1)
    y_pred = self.model.predict(data_input)
    return y_pred
```

Bước 6: Trong phương thức *test()* viết code để kiểm tra độ chính xác của model.

```
def test(self):
    predictions = self.predict(self.X_test)
    loss = mean_squared_error(self.y_test, predictions)
    return loss
```

Bước 7: Viết code cho phương thức *getModelInfo()*

```
def getModelInfo(self):
    print("Intercept (hệ số tự do):", self.model.intercept_)
    print("Coefficients (hệ số của các biến độc lập):", self.model.coef_)
```

Bước 8: Vào file main.py tạo 1 đối tượng của class *UseSklearn* và chạy qua các hàm trên để xem kết quả.

Download source code tại:

https://github.com/NT02IT/BasicMachineLearning/blob/main/LinearRegression/UseSklearn.py

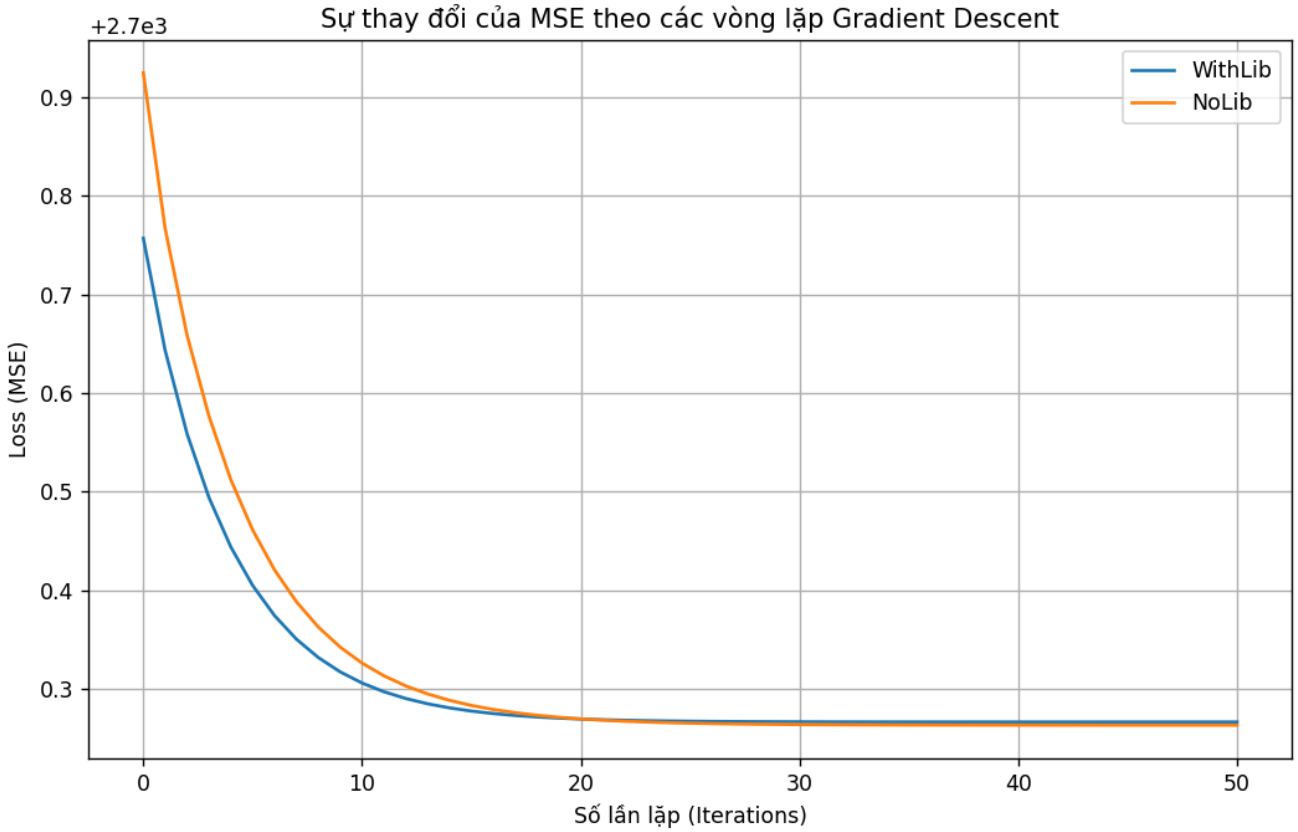
1.8. SO SÁNH CÁC CÁCH THỰC HIỆN

1.8.1. Tổng quan

So sánh trên dataset gồm 3 thuộc tính và 45781 dòng dữ liệu, 3 features (4 cột), so sánh trên các điều kiện môi trường như nhau:

- Thực hiện trên cùng 1 máy tính, cùng môi trường python
- Dùng 50% số dòng (22890 dòng) để huấn luyện và 50% còn lại để test.
- Thời gian được so sánh thông qua thời gian thực hiện hàm *train()*
- Sử dụng hàm Loss MSE để so sánh độ sai lệch giữa *y_pred* và *y_true*

Biểu đồ so sánh sự thay đổi của MSE theo mỗi lần lặp tối ưu Gradient Descent giữa 2 phương pháp sử dụng thư viện Sklearn và sử dụng Gradient Descent không thông qua thư viện



Bảng so sánh chi tiết 3 phương pháp thực hiện LinearRegression:

Tiêu Chí	Use Sklearn	Use Math	Use Gradient Descent
Thời gian chạy	0.110595s	0.110039s	0.131978s
MSE	2886.9703	2.8870e+03	2.8870e+03
Độ chính xác	Cao	Cao	Trung bình (phụ thuộc learning_rate và số vòng lặp)
Dễ triển khai	Dễ	Trung bình, cần tính toán ma trận và áp dụng công thức	Khó, phải tối ưu tham số và đảm bảo hội tụ
Ưu điểm	- Tích hợp tốt, dễ dùng. - Thời gian chạy nhanh nhất.	- Không phụ thuộc thư viện. - Công thức toán học rõ ràng, dễ kiểm tra.	- Linh hoạt, áp dụng được cho bài toán phức tạp. - Có thể tối ưu với dữ liệu lớn.

Nhược điểm	Phụ thuộc vào thư viện.	Không phù hợp với dữ liệu có nhiều cột (do cần tính nghịch đảo ma trận).	- Chậm hơn. - Dễ bị ảnh hưởng bởi tham số khởi tạo hoặc học rate không phù hợp.

1.8.2. Nhận xét

Về mặt thời gian:

- Use Sklearn và Use Math gần như tương đương về thời gian thực thi đối với tập dữ liệu trên.
- Use Gradient Descent chậm hơn một chút do tính chất lặp lại của thuật toán mà không được tối ưu về cách chọn *learning_rate* và *iterations* phù hợp ngay từ đầu.

Về độ mất mát:

- Kết quả MSE *(Mean Squared Error)* của cả ba phương pháp gần như tương đương nhau, chứng tỏ cả ba đều đạt được mô hình hồi quy tuyến tính tốt.
- Gradient Descent có thể không đạt được độ chính xác cao như hai phương pháp còn lại nếu số lần lặp hoặc learning rate không được tối ưu.

Dễ triển khai:

- Use Sklearn là đơn giản nhất vì chỉ cần gọi thư viện mà không cần tính toán nhiều.
- Use Math đòi hỏi kiến thức về đại số tuyến tính (nhân ma trận, nghịch đảo ma trận).
- Gradient Descent phức tạp hơn do cần điều chỉnh nhiều tham số và viết vòng lặp tối ưu.

Ưu điểm:

- Use Sklearn thích hợp cho các ứng dụng thực tế, nơi thời gian triển khai quan trọng.
- Use Math minh bạch và dễ kiểm chứng, phù hợp cho mục đích học tập hoặc khi không muốn phụ thuộc vào thư viện.
- Gradient Descent rất linh hoạt và mạnh mẽ trong việc giải quyết các bài toán lớn hoặc không tuyến tính.

Nhược điểm:

- Use Sklearn phụ thuộc vào thư viện, khó kiểm soát chi tiết các bước bên trong.

- Use Math không mở rộng tốt cho tập dữ liệu lớn do phải tính nghịch đảo ma trận.
- Gradient Descent dễ gặp vấn đề hội tụ và yêu cầu nhiều tinh chỉnh.

CHƯƠNG II: HỒI QUY LOGISTIC

2.1. TỔNG QUAN VỀ HỒI QUY LOGISTIC

Hồi quy Logistic là một mô hình thống kê được sử dụng để phân loại nhị phân, tức dự đoán một đối tượng thuộc vào một trong hai nhóm. Các mô hình phân loại đều tìm cách xác định đường biên phân chia tốt nhất các nhóm dữ liệu. Trong hồi qui Logistic chúng ta cũng tìm kiếm một đường biên phân chia như vậy để giải quyết bài toán phân loại nhị phân giữa hai nhóm 0 và 1.

2.1.1. Ứng dụng của hồi quy logistic:

Hồi quy logistic phục vụ một số mục đích chính trong phân tích thống kê, phân loại và phân tích dự đoán:

- Phân loại và phân tích dự đoán: Hồi quy logistic đơn giản hóa phép toán để đo lường tác động của nhiều biến số (ví dụ: độ tuổi, giới tính, vị trí đặt quảng cáo) với một kết quả nhất định (ví dụ: nhấn vào hoặc bỏ qua). Các mô hình thu được có thể phân tách cẩn thận hiệu quả tương đối của các biện pháp can thiệp khác nhau đối với các nhóm người khác nhau, chẳng hạn như trẻ/già hoặc nam/nữ.
- Dự đoán kết quả nhị phân: Hồi quy logistic là giải pháp lý tưởng để phân tích các tình huống có biến phụ thuộc nhị phân, dự đoán các kết quả có thể xảy ra như có hoặc không dựa trên dữ liệu trước đó. Hiệu quả của nó trong vấn đề này làm cho nó trở thành một yếu tố quan trọng trong các lĩnh vực như tiếp thị, tài chính và khoa học dữ liệu.
- Học máy: Các mô hình logistic cũng có thể chuyển đổi các luồng dữ liệu thô để tạo ra các tính năng cho các loại kỹ thuật AI và máy học khác. Trên thực tế, hồi quy logistic là một trong những thuật toán được sử dụng phổ biến trong học máy cho các bài toán phân loại nhị phân, là các bài toán với hai giá trị lớp, bao gồm các dự đoán có hoặc không và A hoặc B.
- Ước lượng xác suất: Hồi quy logistic cũng có thể ước tính xác suất của các sự kiện, bao gồm xác định mối quan hệ giữa các đặc điểm và xác suất của kết quả. Nghĩa là, nó có thể được sử dụng để phân loại bằng cách tạo ra một mô hình tương quan số giờ học với khả năng học sinh đậu hoặc trượt. Mặt khác, mô hình tương tự có thể được sử dụng để dự

đoán liệu một học sinh cụ thể sẽ đạt hay trượt khi số giờ học được cung cấp dưới dạng một đặc tính và biến cho câu trả lời có hai giá trị: đạt và trượt.

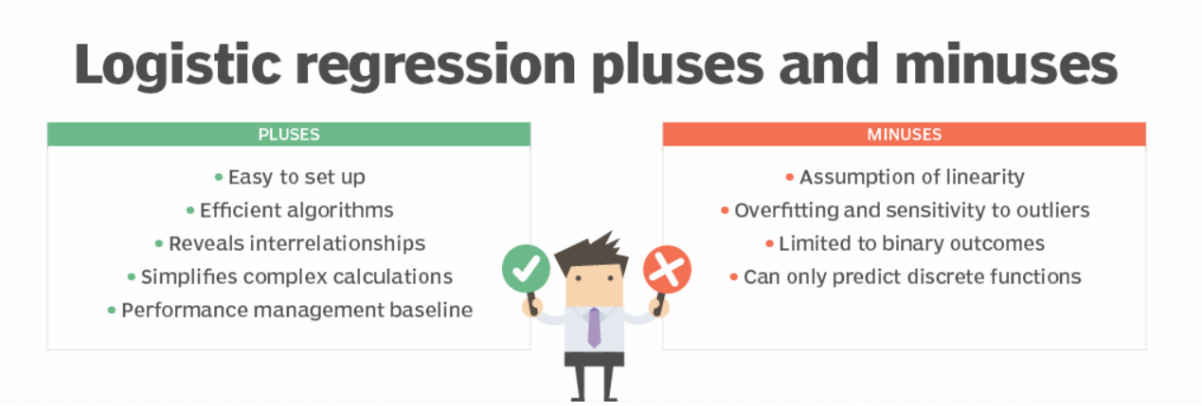
2.1.2. Các kiểu hồi quy logistic:

Hồi quy logistic nhị phân: Được sử dụng cho các vấn đề phân loại nhị phân chỉ có hai kết quả có thể xảy ra. Biến phụ thuộc chỉ có thể có hai giá trị, chẳng hạn như có và không hoặc 0 và 1. Mặc dù hàm logistic tính toán một phạm vi giá trị từ 0 đến 1, mô hình hồi quy nhị phân sẽ làm tròn câu trả lời đến các giá trị gần nhất. Nếu xác suất ước tính lớn hơn 50% (hoặc 0,5), thì mô hình dự đoán rằng phiên bản đó thuộc về lớp đó (đầu ra được gắn nhãn là 1). Mặc dù xác suất nhỏ hơn 50% nhưng mô hình dự đoán rằng phiên bản đó không thuộc lớp đó (đầu ra được gắn nhãn là 0).

Hồi quy logistic Softmax hoặc đa thức: Có thể phân tích các vấn đề có nhiều kết quả có thể xảy ra miễn là số lượng kết quả là hữu hạn. Ví dụ: nó có thể dự đoán giá nhà sẽ tăng 25%, 50%, 75% hay 100% dựa trên dữ liệu dân số, nhưng nó không thể dự đoán giá trị chính xác của một ngôi nhà. Hồi quy logistic này hoạt động bằng cách ánh xạ các giá trị kết quả tới các giá trị khác nhau trong khoảng từ 0 đến 1. Vì hàm logistic có thể trả về một phạm vi dữ liệu liên tục, như 0,1, 0,11, 0,12, v.v., hồi quy softmax cũng nhóm đầu ra thành các giá trị gần nhất có thể .

Hồi quy logistic thứ tự: Hồi quy logistic thông thường, hay mô hình logit có thứ tự, là một loại hồi quy đa thức đặc biệt cho các bài toán trong đó các số biểu thị thứ hạng thay vì giá trị thực. Ví dụ: bạn sẽ sử dụng hồi quy thứ tự để dự đoán câu trả lời cho câu hỏi khảo sát yêu cầu khách hàng xếp hạng dịch vụ của bạn là kém, trung bình, tốt hoặc xuất sắc dựa trên một giá trị bằng số, chẳng hạn như số lượng mặt hàng họ mua từ bạn trong suốt thời gian qua.

2.1.3. Ưu và nhược điểm của hồi quy logistic



Ưu điểm:

- **Dễ dàng cài đặt:** Ưu điểm chính của hồi quy logistic là đây là một mô hình đơn giản, dễ thiết lập và đào tạo dễ dàng hơn nhiều so với các mô hình học máy khác như mạng thần kinh và ứng dụng AI.
- **Thuật toán hiệu quả:** Một ưu điểm khác là đây là một trong những thuật toán hiệu quả nhất khi các kết quả hoặc sự khác biệt khác nhau được biểu thị bằng dữ liệu có thể phân tách tuyến tính. Điều này có nghĩa là bạn có thể vẽ một đường thẳng ngăn cách các kết quả của phép tính hồi quy logistic.
- **Tiết lộ mối quan hệ tương tác giữa các biến:** Một trong những điểm hấp dẫn nhất của hồi quy logistic đối với các nhà thống kê là nó có thể giúp tiết lộ mối quan hệ qua lại giữa các biến khác nhau và ảnh hưởng của chúng đến kết quả. Điều này có thể nhanh chóng xác định khi nào hai biến số có mối tương quan tích cực hay tiêu cực, chẳng hạn như phát hiện được trích dẫn ở trên rằng nghiên cứu nhiều hơn có xu hướng tương quan với kết quả kiểm tra cao hơn. Nhưng điều quan trọng cần lưu ý là cần phải có các kỹ thuật khác như AI nhân quả để tạo bước nhảy vọt từ mối tương quan sang mối quan hệ nhân quả.
- **Biến các phép tính phức tạp thành các bài toán đơn giản:** Hồi quy logistic biến các phép tính phức tạp xung quanh xác suất thành một bài toán số học đơn giản. Bản thân việc tính toán rất phức tạp nhưng các phương pháp và ứng dụng thống kê hiện đại sẽ tự động hóa phần lớn các phép tính. Điều này giúp đơn giản hóa đáng kể việc phân tích tác động của nhiều biến số và giảm thiểu tác động của các yếu tố gây nhiễu. Kết quả là, các nhà thống kê có thể nhanh chóng lập mô hình và khám phá sự đóng góp của nhiều yếu tố khác

nhau vào một kết quả nhất định. Hồi quy logistic chuyển đổi xác suất tương đối của bất kỳ nhóm con nào thành số logarit, được gọi là hệ số hồi quy, có thể được cộng hoặc trừ để đạt được kết quả mong muốn. Các hệ số hồi quy đơn giản hơn này cũng có thể đơn giản hóa các thuật toán học máy và khoa học dữ liệu khác.

- **Cơ sở để quản lý hiệu suất:** Hồi quy logistic thường được sử dụng làm cơ sở để đo lường hiệu suất do thiết lập nhanh chóng và dễ dàng.

Nhược điểm:

- **Giả định về tính tuyến tính:** Vì hồi quy logistic giả định mối quan hệ tuyến tính giữa một biến phụ thuộc và các biến độc lập nên khả năng ứng dụng của nó trong một số trường hợp nhất định có thể bị hạn chế.
- **Trang bị quá mức và nhạy cảm với các ngoại lệ:** Hồi quy logistic rất nhạy cảm với các ngoại lệ. Nếu số lượng quan sát nhỏ hơn số lượng đặc điểm thì không nên sử dụng hồi quy logistic; nếu không nó có thể dẫn đến trang bị quá mức. Kỹ thuật chính quy hóa L1 và L2 có thể được áp dụng để giúp giảm tình trạng trang bị quá mức.
- **Giới hạn ở kết quả nhị phân:** Hồi quy logistic được giới hạn trong việc lập mô hình phân loại nhị phân và kết quả và có thể không phù hợp với các kịch bản có kết quả không nhị phân nếu không có sửa đổi như hồi quy logistic thứ tự.
- **Chỉ có thể dự đoán các chức năng rời rạc:** Hồi quy logistic được thiết kế riêng để dự đoán các hàm rời rạc, hạn chế việc sử dụng nó đối với các biến phụ thuộc trong một tập hợp số rời rạc. Hạn chế này đặt ra những thách thức cho việc dự đoán dữ liệu liên tục.

2.1.4. So sánh hồi quy logistic và hồi quy tuyến tính

Hồi quy tuyến tính	Hồi quy logistic
Cung cấp một đầu ra không đổi.	Cung cấp đầu ra liên tục.
Kết quả hoặc biến phụ thuộc là biến nhị phân và chỉ có hai giá trị có thể.	Kết quả là liên tục, có nghĩa là nó có thể có bất kỳ một trong vô số giá trị có thể có.
Được sử dụng khi biến phản hồi mang tính phân loại, chẳng hạn như có hoặc không, đúng hoặc sai và đạt hoặc không đạt.	Được sử dụng khi thay vì biến phân loại, biến phản hồi là liên tục, chẳng hạn như giờ, chiều cao và cân nặng.

Chỉ cho phép các giá trị hoặc danh mục cụ thể	
Được sử dụng cho các nhiệm vụ phân loại như dự đoán liệu một email có phải là thư rác hay không bằng cách nghiên cứu các biến và tính năng dự đoán của nó.	Được sử dụng cho các nhiệm vụ như dự đoán doanh số bán hàng trong tương lai dựa trên dữ liệu lịch sử.

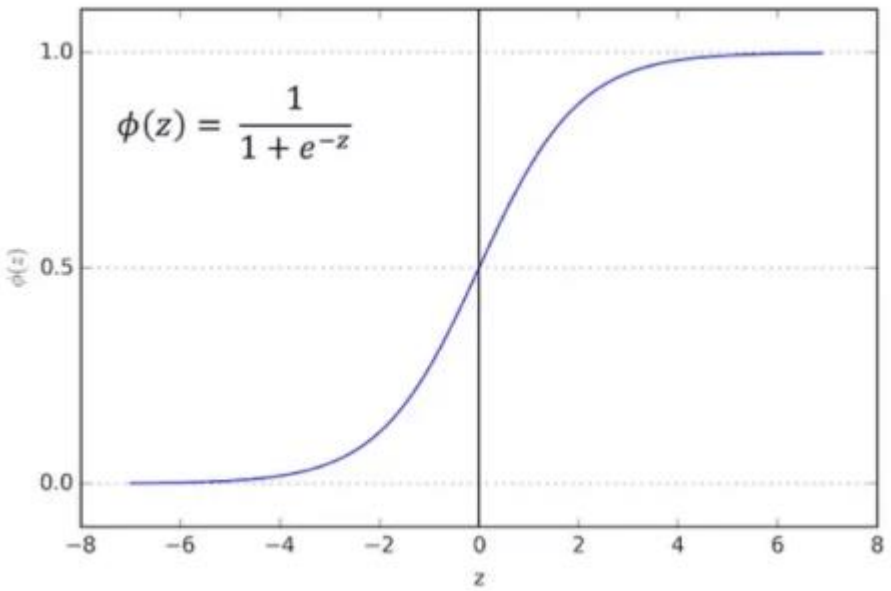
2.2. MÔ HÌNH HỒI QUY LOGISTIC

Vì hồi quy Logistic được dùng để giải quyết phân loại nhị giá nên để xây dựng mô hình hồi quy Logistic ta cần 1 hàm mà ở đó thỏa các điều kiện như:

- Liên tục nhận các giá trị thực và bị chặn trong khoản (0,1).
- Nếu chọn 1 điểm là ngưỡng (ở giữa) thì các điểm càng xa ngưỡng về trái sẽ có giá trị càng gần với 0, và ngược lại các điểm càng xa ngưỡng về phải sẽ có giá trị gần với 1.
- Có đạo hàm tại mọi điểm, để dễ dàng sử dụng Gradient Descent cho việc tối ưu hệ số.

Trong số các hàm số có 3 tính chất trên hàm *sigmoid* thường được sử dụng nhiều nhất.

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$



Ảnh 5: Đồ thị hàm sigmoid

Ngoài 3 tính chất trên hàm *sigmoid* còn có 1 điểm lợi khác là đạo hàm của nó khá đơn giản:

$$\sigma'(s) = \sigma(s) * (1 - \sigma(s))$$

Với mô hình hồi quy tuyến tính là $\hat{y_i} = w_0 + w_1 * x_i$ thì với mô hình logistic sẽ là:

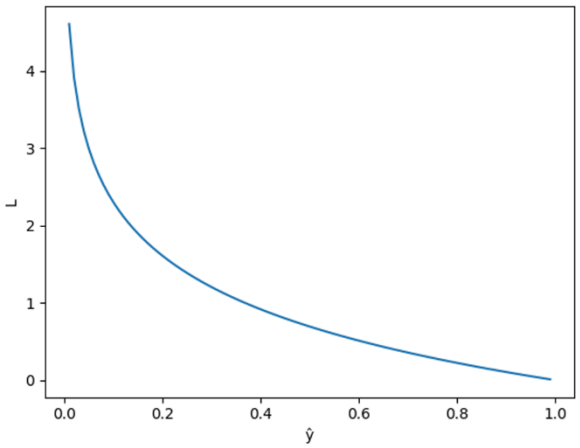
$$\hat{y_i} = \sigma(w_0 + w_1 * x_i) = \frac{1}{1 + e^{w_0 + w_1 * x_i}}$$

2.3. LOSS FUNCTION

Sau khi đã có mô hình cơ bản của hồi quy Logistic chúng ta cũng cần 1 hàm để đánh giá độ tốt của model. Với mỗi điểm $(x^{(i)}, y_i)$ gọi hàm loss $L = -(y_i * log\ log\ (\hat{y_i}) + (1 - y_i) * log\ (1 - \hat{y_i}))$ loss function này có tên gọi là *binary_crossentropy*. *(Lưu ý mặc định trong machine learning hoặc các lĩnh vực IT nói chung khi viết log sẽ được hiểu là ln nên $logx' = \frac{1}{x}$)*

Đánh giá Loss function

Với $y_i = 1 \Rightarrow L = -log\hat{y_i}$



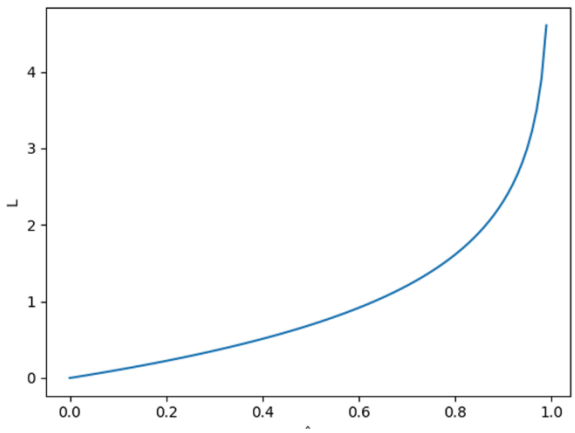
Nhận xét:

– Hàm L giảm dần từ 0 đến 1.

– Khi model dự đoán $\hat{y_i}$ gần 1, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0.

– Khi model dự đoán $\hat{y_i}$ gần 0, tức giá trị dự đoán ngược lại với giá trị thật y_i thì L rất lớn.

Với $y_i = 0 \Rightarrow L = -log\ (1 - \hat{y_i})$



Nhận xét:

– Hàm L tăng dần từ 0 đến 1.

– Khi model dự đoán $\hat{y_i}$ gần 0, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0.

– Khi model dự đoán $\hat{y_i}$ gần 1, tức giá trị dự đoán ngược lại với giá trị thật y_i thì L rất lớn.

Hàm L nhỏ khi giá trị model dự đoán gần với giá trị thật và rất lớn khi model dự đoán sai, hay nói cách khác L càng nhỏ thì model dự đoán càng gần với giá trị thật.

⇒ Bài toán tìm model trở thành tìm giá trị nhỏ nhất của L.

Hàm loss function trên toàn bộ dữ liệu:

$$J = -\frac{1}{N} * \sum_{i=1}^{N} (y_i * log\ log\ (\hat{y_i})\ + (1 - y_i) * log\ (1 - \hat{y_i}))$$

2.4. ÁP DỤNG GRADIENT DESCENT

Để áp dụng thuật toán gradient descent tối ưu loss function ta cần tính đạo hàm của loss function theo từng biến w.

Với mỗi điểm $(x^{(i)}, y_i)$, gọi hàm loss function $L = -(y_i * log\ log\ (\hat{y_i})\ + (1 - y_i) * log\ (1 - \hat{y_i}))$ trong đó $\hat{y_i} = \sigma(w_0 + w_1 * x_i) = \frac{1}{1+e^{w_0+w_1*x_i}}$, còn y_i là giá trị thật của dữ liệu.

Áp dụng Chain rule trong tính đạo hàm ta tính các đạo hàm sau:

$\frac{dL}{d\hat{y_i}} = \frac{\hat{y_i} - y_i}{\hat{y_i} * (1 - \hat{y_i})}$ $\frac{d\hat{y_i}}{dw_0} = \hat{y_i} * (1 - \hat{y_i})$,

$\frac{d\hat{y_i}}{dw_1} = x_1^{(i)} * \hat{y_i} * (1 - \hat{y_i})$ $\frac{d\hat{y_i}}{dw_2} = x_2^{(i)} * \hat{y_i} * (1 - \hat{y_i})$,

Suy ra đạo hàm từng biến w tại 1 điểm $(x^{(i)}, y_i)$ là:

$\frac{dL}{dw_0} = \hat{y_i} - y_i$ $\frac{dL}{dw_1} = x_1^{(i)} * (\hat{y_i} - y_i)$ $\frac{dL}{dw_2} = x_2^{(i)} * (\hat{y_i} - y_i)$

Đạo hàm từng biến w trên toàn bộ dữ liệu:

$\frac{dL}{dw_0} = \frac{1}{N} * \sum_{i=1}^{N} \hat{y_i} - y_i$

$\frac{dL}{dw_1} = \frac{1}{N} * \sum_{i=1}^{N} x_1^{(i)} * (\hat{y_i} - y_i)$ $\frac{dL}{dw_2} = \frac{1}{N} * \sum_{i=1}^{N} x_2^{(i)} * (\hat{y_i} - y_i)$

Biểu diễn bài toán dưới dạng ma trận:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \dots & \dots & \dots \\ 1 & x_1^{(3)} & x_2^{(3)} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

$$\hat{y} = \sigma(X * w)$$

$$\frac{dL}{dw_i} = \frac{1}{N} * X_i^T * (\hat{y_i} - y_i)$$

2.5. ÁP DỤNG HỒI QUY LOGISTIC ĐỂ TẠO 1 MÔ HÌNH PHÂN LOẠI

2.5.1. Dataset

PatientID	Gene1	Gene2	Gene3	Gene4	Gene5	5-year survival
1	92.0826	443.3735	350.94664	11.1876	77.926	0
2	97.1228	29.215618	2.579007	301.8684	171.9968	0
3	7.73995	42.368421	39.907121	9.2879	104.2632	1
…	…	…	…	…	…	…
86	192.027	30.881527	24.188658	59.5172	100.2386	0
87	220.0724	14.933961	13.052915	38.3612	125.2776	0

Download full dataset tại: https://github.com/NT02IT/BasicMachineLearning/blob/main/datasets/logistic-regression/logistic-regression.csv

Với dataset trên ta có:

- Các biến độc lập (X) là: Gene1 đến Gene5 *(Các gene của bệnh nhân)*
- Biến phụ thuộc (Y) là: 5-year survival *(Tình trạng sống sót của bệnh nhân sau 5 năm)*
- Số lượng mẫu (N): 87

2.5.2. Mẫu cần dự đoán

PatientID	Gene1	Gene2	Gene3	Gene4	Gene5
1	23.1	45.6	16.3	77.9	250.5

2	508.4	31.9	16.2	24.3	45.5
3	86.1	59	18.5	59.3	483.8
4	8	67.2	51.2	191.4	80.2
5	55.5	97.5	8.8	28.2	72.2

2.5.3. Xây dựng mô hình bằng hàm excel

Dữ liệu đầu vào:

PatientID	Gene1	Gene2	Gene3	Gene4	Gene5	5-year survival
1	92.0826	443.3735	350.9466	11.1876	77.926	0
2	97.1228	29.21562	2.579007	301.8684	171.9968	0
3	7.73995	42.36842	39.90712	9.2879	104.2632	1
4	60.2125	25.63164	2.420307	14.1677	94.6316	1
5	385.4766	20.32964	5.831751	35.4298	71.0588	0
6	476.644	12.48745	4.406125	50.444	104.7838	0

Đầu tiên, chúng ta sẽ tạo trọng số bằng cách sử dụng hàm LINEST và INDEX:

$$= @INDEX(LINEST(X1, X2),1, D1)$$

Trong đó $\boldsymbol{LINEST(X1, X2)}$:

- LINEST là hàm dùng để thực hiện hồi quy tuyến tính, tìm hệ số của phương trình tuyến tính dưới dạng y=mX+by = mX + by=mX+b (trong đó m là hệ số dốc và b là hằng số).
- X1 là phạm vi giá trị của biến phụ thuộc (giá trị cần dự đoán).
- X2 là phạm vi giá trị của biến độc lập (biến đầu vào để dự đoán).

Kết quả của $LINEST$ là một mảng giá trị chứa:

- Hệ số dốc m: vị trí đầu tiên trong mảng (hàng 1, cột 1).
- Intercept b: vị trí tiếp theo trong mảng.

Và $\boldsymbol{INDEX(\ldots, 1, D1)}$:

- INDEX trích xuất giá trị từ mảng kết quả LINEST, dựa vào tham số hàng và cột.
- Hàng 1 được chỉ định trong INDEX, tức là lấy dòng đầu tiên của kết quả LINEST (chứa các hệ số).
- Cột D1: Giá trị của ô D1 sẽ xác định cột nào được trích xuất từ dòng đầu tiên.

5	4	3	2	1	6
m1	m2	m3	m4	m5	b
-0.000316464	0.00301	-0.00513	-0.00051	-0.00074	0.535574025

Sau đó, chúng ta tính giá trị hồi quy bằng cách sử dụng công thức hồi quy tuyến tính. Có được giá trị hồi quy, chúng ta có thể sử dụng hàm sigmoid để chuyển giá trị ấy thành xác xuất.

m1x1+m2x2+...+b	P(1)
-0.581355664	0.358620715
-0.534364235	0.369499576
0.188276772	0.546930641

Chúng ta cần phân loại được điểm dữ liệu đó bằng cách kiểm tra xem giá trị P(1) có lớn hơn hay nhỏ hơn 0.5. Sau khi có được cột outcome, chúng ta sẽ cần phải kiểm tra xem nhóm dự đoán có đúng với nhóm điểm đầu vào không, nếu đúng để số 0, nếu sai để số 1.

Outcome	Difference
0	0
0	0
1	0

Tiếp theo, chúng ta tính likelihood bằng cách nếu nhóm điểm đầu vào bằng 1 thì lấy xác xuất P(1), nếu bằng 0 thì lấy 1 trừ xác xuất P(1). Với Ln(likelihood), nếu likelihood bằng 0, lấy giá trị -1000000, còn lại lấy log của likelihood.

Likelihood	**Ln(likelihood)**
0.641379285	-0.444134289
0.630500424	-0.461241451
0.546930641	-0.603433283

Sau đó, tính tổng toàn bộ cột Ln(likelihood) và tổng số lượng nhóm dự đoán khác nhóm đầu vào (total Diff).

Tổng Ln(likelihood):	-28.72108933

total Diff =	13

Nếu tổng của Ln(likelihood) quá nhỏ và total Diff quá lớn chúng ta có thể sử dụng solver hoặc gradient descent để tìm được trọng số tối ưu.

2.5.4. Xây dựng mô hình bằng code Python

2.5.4.1. Sử dụng Gradient Descent để tối ưu trọng số

Bước 1: Tạo class *UseGradientDescent.py* với phương thức khởi tạo _init_ và các phương thức *training(), testing(), predict(), getModelInfo(), _sigmoid(), _compute_cost(), _gradient_descent().*

- Phương thức khởi tạo đọc dataset training sau đó chia dữ liệu thành 2 phần 1 nữa dùng để huấn luyện mô hình, nữa còn lại dùng để test.
- Phương thức *training()* gọi đến hàm *_gradient_descent()* để tối ưu các trọng số.
- Phương thức *testing()* dự đoán cho nữa dữ liệu test sau đó so sánh kết quả để xem độ chính xác của mô hình.
- Phương thức *predict()* dùng để dự đoán và xuất ra kết quả dự đoán.
- Phương thức *getModelInfo()* dùng để in ra màn hình console các thông tin của model.
- Phương thức *_sigmoid()* định nghĩa hàm sigmoid.
- Phương thức *_compute_cost()* tính toán chi phí giữa y dự đoán và y thực tế.
- Phương thức *_gradient_descent()* thuật toán gradient descent để điều chỉnh trọng số qua các lần lặp.

Bước 2: Trong hàm khởi tạo viết code đọc dữ liệu từ URL Dataset sau đó chia dữ liệu thành 2 tập train và test với độ lớn của tập train được tính bằng $train\_size * 100\%$

```
def __init__(self, datasetURL, train_size=0.5):
    csv_handler = CSVHandler(datasetURL)
    dataframe = csv_handler.read_csv()

    dataframe, label_encoders = Normalization.encode_dataframe(dataframe)

    X = dataframe.iloc[:, :-1]    # Chọn tất cả các hàng và cột trừ cột cuối cùng
    y = dataframe.iloc[:, -1]     # Chọn cột cuối cùng
```

```
    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(dataframe) * train_size)
    self.X_train = X.iloc[:split_index]
    self.y_train = y.iloc[:split_index]
    self.X_test = X.iloc[split_index:]
    self.y_test = y.iloc[split_index:]
```

Bước 3: Trong hàm *_sigmoid()* viết code để nhập vào z sẽ trả về $\sigma(z)$

```
def _sigmoid(self, z):
    # Giới hạn ở -709 và 709 để tránh khi e mũ quá lớn hoặc quá nhỏ sẽ gây lỗi tràn số
    z = np.clip(z, -709, 709)
    return 1 / (1 + np.exp(-z))
```

Bước 4: Trong hàm *_compute_cost()* viết code nhận input là giá trị *y* (y thực tế) và *y_hat* (y dự đoán) và trả về chi phí giữa *y* và *y_hat.*

```
def _compute_cost(self, y, y_hat):
    # Tính toán chi phí (cost) sử dụng hàm binary cross-entropy.
    m = self.ytrain.shape[0]  # Số lượng mẫu
    cost = -1/m * np.sum(y * np.log(y_hat + 1e-10) + (1 - y) * np.log(1 - y_hat + 1e-10))
    return cost
```

1e-10: Một số rất nhỏ (epsilon) được thêm vào để tránh lỗi khi tính log của 0.

Bước 5: Trong hàm *_gradient_descent()* viết code nhập vào dataframe *X* và *y* cùng với *learning_rate* (tốc độ học) và *iterations* (số lần điều chỉnh w / *learning_rate*) để áp dụng thuật toán gradient descent điều chỉnh w.

```
def _gradient_descent(self, X, y, learning_rate=1e-5, iterations=1500):
    m, n = X.shape  # m: số lượng mẫu, n: số lượng đặc trưng
    # Khởi tạo trọng số
    self.weights = np.zeros(n)
    self.bias = 0

    count_patience = 0
    self.cost_values = []
    for i in range(iterations):
        linear_model = np.dot(X, self.weights) + self.bias
        y_hat = self._sigmoid(linear_model)

        # Tính toán chi phí giữa y dự đoán và y thực tế
        cost = self._compute_cost(y, y_hat)

        # Tính toán gradient
```

```
    dw = (1/m) * np.dot(X.T, (y_hat - y))  # Gradient cho trọng số
    db = (1/m) * np.sum(y_hat - y)  # Gradient cho bias

    # Cập nhật trọng số và bias
    self.weights -= learning_rate * dw
    self.bias -= learning_rate * db

    self.cost_values.append(cost)
    threshold = 10  # Ngưỡng thay đổi loss
    if i > 0 and abs(self.cost_values[-1] - self.cost_values[-2]) < threshold:
      count_patience += 1
      if count_patience >= 50:
        print(f"Dừng tại vòng lặp {i} vì loss không thay đổi đáng kể")
        break
    else:
      count_patience = 0
```

Bước 6: Viết hàm *training()* sử dụng hàm *_gradient_descent()* để tối ưu trọng số mô hình.

```
def training(self):
  self._gradient_descent(self.Xtrain, self.ytrain)
```

Bước 7: Viết hàm *testing()* thực hiện dự đoán trên tập dữ liệu test sau đó trả về độ chính xác của mô hình bằng cách tính tỉ lệ giữa số kết quả dự đoán đúng trên tổng mẫu thử.

```
def testing(self):
  # Dự đoán trên tập testing
  linear_model = np.dot(self.Xtest, self.weights) + self.bias
  y_hat = self._sigmoid(linear_model)
  predictions = [1 if i > 0.5 else 0 for i in y_hat]  # Ngưỡng 0.5

  # Tính độ chính xác so với kết quả thực tế
  matches = self.ytest == predictions
  accuracy = matches.mean() * 100  # tính trung bình và chuyển sang %
  print(f"Độ chính xác: {accuracy}%")
```

Bước 8: Viết hàm *predict()* để dự đoán cho tập dự đoán

```
def predict(self, data_input):
  linear_model = np.dot(data_input, self.weights) + self.bias
  y_pred = self._sigmoid(linear_model)
  predictions = [1 if i > 0.5 else 0 for i in y_pred]  # Ngưỡng phân lớp 0.5
  return predictions
```

Bước 9: Vào file main.py tạo 1 đối tượng của class UseGradientDescent và chạy qua các hàm trên để xem kết quả.1.5.4.2. Sử dụng model LogisticRegression thuộc thư viện Sklearn.

Download source code tại:

https://github.com/NT02IT/BasicMachineLearning/blob/main/LogisticRegression/UseGradientDescent.py

2.5.4.2. Sử dụng mô hình LogisticRegression thuộc thư viện Sklearn

Bước 1: Tạo class *UseSklearn.py* với phương thức khởi tạo _init_ và các phương thức *training(), testing(), predict(), getModelInfo().*

- Phương thức khởi tạo đọc dataset training sau đó chia dữ liệu thành 2 phần 1 nữa dùng để huấn luyện mô hình, nữa còn lại dùng để test.
- Phương thức *training()* gọi đến hàm *fit()* của mô hình LinearRegression để huấn luyện mô hình.
- Phương thức *testing()* dự đoán cho nữa dữ liệu test sau đó dùng hàm *accuracy_score()* thuộc thư viện *sklearn* để tính độ chính xác của mô hình.
- Phương thức *predict()* dùng để dự đoán và xuất ra kết quả dự đoán.
- Phương thức *getModelInfo()* dùng để in ra màn hình console các thông tin của model.

Bước 2: Viết hàm khởi tạo tương tự như trên.

Bước 3: Trong phương thức *traning()* sử dụng hàm *fit()* để huấn luyện mô hình

```
def training(self):
  self.model.fit(self.Xtrain, self.ytrain)
```

Bước 4: Trong hàm predict() viết code để dựa đoán đầu ra cho dữ liệu đầu vào

```
def predict(self, data_input):
  return self.model.predict(data_input)
```

Bước 5: Trong hàm *testing()* viết code sử dụng model dự đoán cho tập test sau đó sử dụng hàm *accuracy_score()* thuộc thư viện sklearn để tính độ chính xác của mô hình.

```
def testing(self):
    # Dự đoán trên tập testing
    y_pred = self.model.predict(self.Xtest)

    # Tính độ chính xác so với kết quả thực tế
    accuracy = accuracy_score(self.ytest, y_pred) * 100
    print(f"Độ chính xác: {accuracy}%")
```

Bước 6: Trong hàm viết code để xuất các thông tin của model ra màn hình console

```
def getModelInfo(self):
    print("Intercept (hệ số tự do):", self.model.intercept_[0])
    print("Coefficients (hệ số của các biến độc lập):", self.model.coef_[0])
```

Bước 7: Vào file main.py tạo 1 đối tượng của class UseSklearn và chạy qua các hàm trên để xem kết quả.

Download source code tại:

https://github.com/NT02IT/BasicMachineLearning/blob/main/LogisticRegression/UseSklearn.py

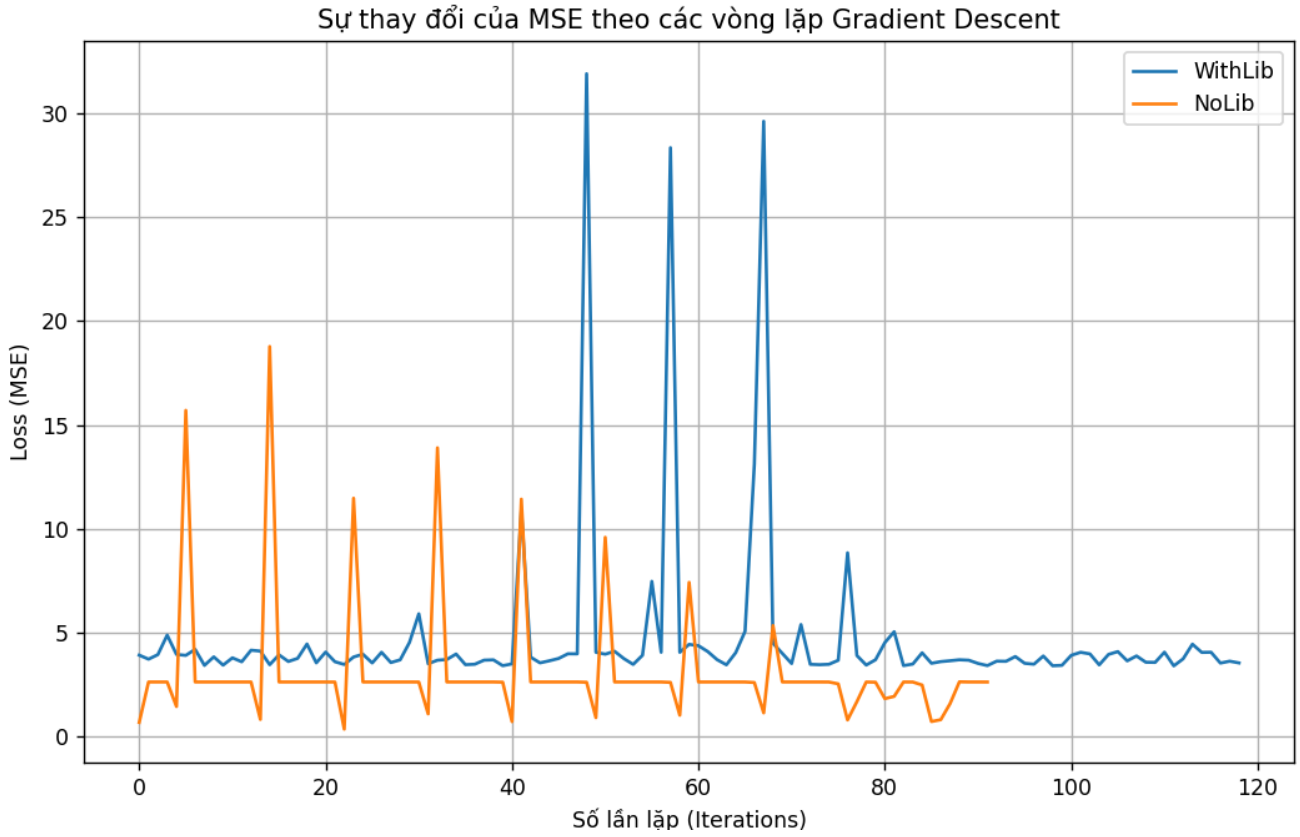
2.8. SO SÁNH VỚI THƯ VIỆN SKLEARN

2.8.1. Tổng quan

Thực hiện 2 thuật toán cùng trên 1 dataset gồm 20 thuộc tính (21 cột) và 41100 dòng dữ liệu, so sánh được thực hiện trên các điều kiện môi trường như nhau:

- Thực hiện trên cùng 1 máy tính, cùng môi trường python
- Dùng 50% số dòng (20594 dòng) để huấn luyện và 50% còn lại để test.
- Thời gian được so sánh thông qua thời gian thực hiện hàm *train()*
- Sử dụng hàm Loss MSE để so sánh độ sai lệch giữa *y_pred* và *y_true*
- Ngưỡng thay đổi Loss *(threshold)* không đáng kể: 10
- *count_patience* – số lần độ chệnh lệch Loss với lần trước đó nhỏ hơn *threshold*: 50

Biểu đồ so sánh sự thay đổi của MSE theo mỗi lần lặp tối ưu Gradient Descent giữa 2 phương pháp sử dụng thư viện Sklearn và sử dụng Gradient Descent không thông qua thư viện:



Dựa trên biểu đồ ta thấy:

- Logistic Regression với thư viện dừng tại vòng lặp 118 vì loss không thay đổi đáng kể
- Logistic Regression với Gradient Descent dừng tại vòng lặp 91 nhưng kết quả vẫn khá tốt khi so với khi sử dụng Sklearn, tuy nhiên vẫn còn biến động lớn ở giai đoạn gần cuối ⇒ vẫn có thể nới rộng khoảng so sánh độ thay đổi của các lần tối ưu *(count_patience)* để nâng cao độ chính xác của mô hình, nhưng khi làm như vậy buộc phải hi sinh về mặt thời gian.

Bảng so sánh hiệu suất model theo các tiêu chí:

Tiêu Chí	Use Gradient Descent	Use Sklearn
Thời gian chạy	1.075971s	0.413877s
Độ chính xác	88.98%	90.5%

2.8.2. Nhận xét

Thời gian chạy: SK Model thực hiện nhanh hơn rất nhiều so với Gradient Descent về thời gian chạy, SK Model mất 0.413877 giây so với 1.075971 giây của Gradient Descent. Sự chênh lệch này là rất lớn, cho thấy SK Model hiệu quả hơn nhiều về mặt tính toán trên tập dữ liệu lớn. Đối với Gradient Descent thì cần phải tối ưu thuật toán hơn về mặt thời gian để có thể phù hợp với những dữ liệu lớn.

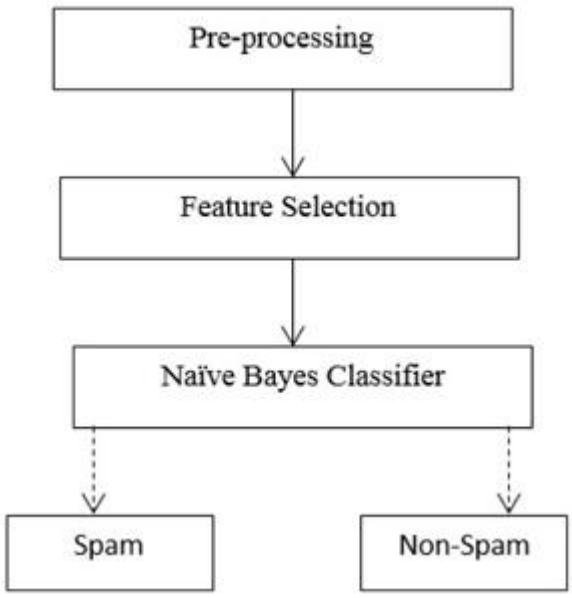
Độ chính xác: Kết quả cho thấy độ chính xác của Sklearn chỉ nhỉnh hơn Gradient Descent hơn 1%. Tuy nhiên nếu chọn *learning_rate* và số lần lặp không phù hợp, Gradient Descent có thể không cho ra kết quả chính xác.

CHƯƠNG III: MÔ HÌNH NAÏVE BAYES

3.1. ĐỊNH NGHĨA

Naïve Bayes được sử dụng cho các nhiệm vụ phân loại. Thuật toán này dựa trên định lý Bayes, cho rằng điều kiện xảy ra trước đó có thể liên quan đến một sự kiện sẽ ảnh hưởng đến khả năng xảy ra của sự kiện đó. Với Naïve Bayes, phương pháp hoạt động dựa trên giả định "naïve " *(giả định đơn giản)* rằng mỗi đặc trưng trong một tập dữ liệu là không liên quan đến các đặc trưng khác, mặc dù điều này có thể không hoàn toàn đúng, nhưng sự đơn giản hóa này giúp cho việc tính toán trở nên dễ dàng hơn. Các xác suất của từng đặc trưng theo từng lớp được học bởi thuật toán thông qua quá trình huấn luyện trên dữ liệu có nhãn. Sau đó, khi được cung cấp dữ liệu mới không có nhãn, thuật toán tính xác suất của mỗi lớp dựa trên các đặc trưng đã quan sát và chọn lớp có xác suất cao nhất làm dự đoán.

Ví dụ, giả sử ta muốn tạo một bộ phân loại có thể quyết định liệu một email có phải là thư rác *(spam)* hay không. Ta có thể trích xuất các thuộc tính như sự hiện diện của các từ hoặc cụm từ nhất định, địa chỉ email của người gửi và dòng chủ đề của email từ một tập dữ liệu các email đã được phân loại là thư rác hoặc không phải thư rác. Sau đó, các đặc trưng này sẽ được sử dụng bởi thuật toán Naïve Bayes để xác định liệu một email mới có khả năng là thư rác hay không.



Ảnh 8: Quá trình phân loại email spam

3.2. ĐỊNH LÝ BAYES

Định lý Bayes được sử dụng để tính xác suất có điều kiện. Xác suất có điều kiện được định nghĩa là khả năng xảy ra một sự kiện hoặc kết quả, dựa trên sự xuất hiện của một sự kiện hoặc kết quả trước đó.

$$P(B) = \frac{P(A).P(A)}{P(B)}$$

Trong đó:

- P(A|B): Xác suất của A xảy ra với điều kiện B đã xảy ra..
- P(A): Xác suất của A xảy ra (tiên nghiệm).
- P(B|A): Xác suất của B xảy ra với điều kiện A đã xảy ra.
- P(B): Xác suất của B xảy ra (tiên nghiệm).

Nói một cách đơn giản hơn, Định lý Bayes là một cách tìm xác suất khi chúng ta biết một số xác suất khác.

3.3. SỬ DỤNG NAÏVE BAYES ĐỂ PHÂN LOẠI

Đây là một ví dụ để hiểu cách hoạt động của Naïve Bayes. Hãy xem xét tập dữ liệu thời tiết này, nơi ta đang phân loại xem một ngày cụ thể có phù hợp để chơi golf hay không, dựa trên các đặc điểm của ngày đó.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes

Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Các cột thể hiện các tính năng và các hàng thể hiện các mục riêng lẻ cho mỗi ngày. Nếu lấy hàng đầu tiên, chúng ta có thể quan sát thấy khi trời mưa, nhiệt độ nóng, độ ẩm cao và không có gió thì ngày đó không thích hợp để chơi gôn. Vì vậy, chúng ta có thể đưa ra hai giả định sau:

- Temp "Hot" không liên quan gì đến Humidity "High" hay "Normal", khả năng "Rainy" không ảnh hưởng đến Windy. Do đó, các đặc trưng được coi là độc lập.
- Chỉ biết Temp và Humidity chúng ta không thể dự đoán kết quả chính xác. Không có thuộc tính nào là không liên quan và được cho là có đóng góp như nhau cho kết quả.

Phương trình định lý Bayes cho tập dữ liệu thời tiết này:

$$P(X) = \frac{P(y).P(y)}{P(X)}$$

- y : biến mục tiêu/lớp (cho biết có chơi golf hay không)
- X : các thông số/tính năng (Outlook, Temp, Humidity, Windy, Play Golf) dao động từ $x_1, x_2, \dots, x_n$. Ở đây, ta có 4 tính năng x_1, x_2, x_3, x_4.

Bằng cách thay thế $x_1, x_2, \dots, x_n$ cho X và khai triển bằng quy tắc dây chuyền, ta thu được xác suất hậu nghiệm là:

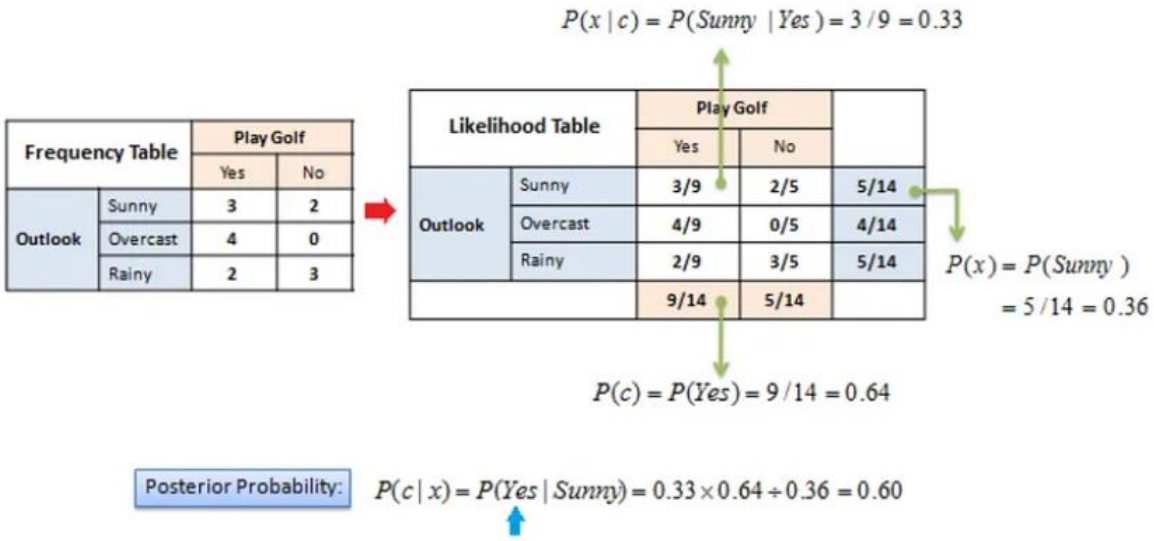
$$P(x_1, \dots, x_n) = \frac{P(y).P(y) \dots P(y_n)\, P(y)}{P(x_1).P(x_2) \dots P(x_n)}$$

Bây giờ, ta có thể lấy các giá trị cho từng lớp bằng cách xem tập dữ liệu và thay thế chúng vào phương trình trên. Xác suất sau có thể được tính bằng cách trước tiên xây dựng bảng tần số cho

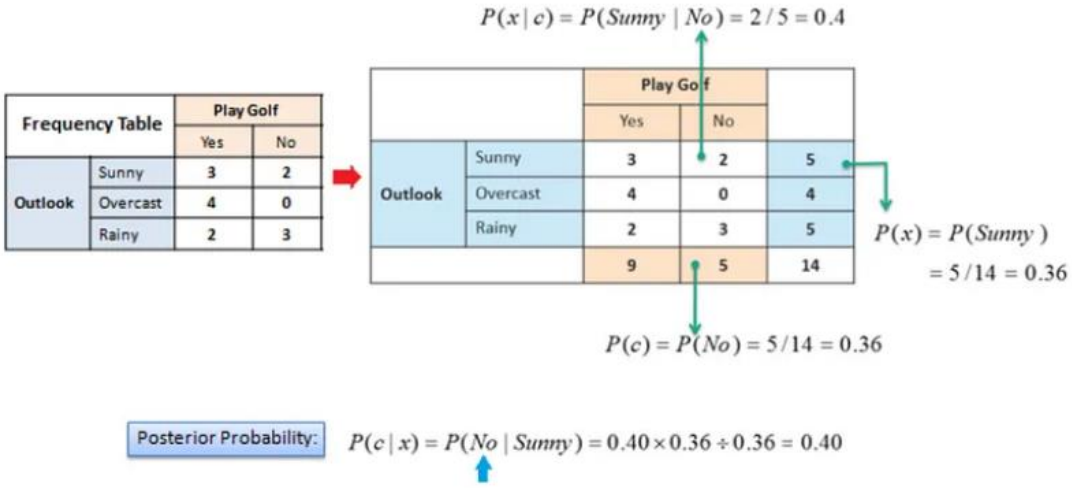
từng thuộc tính đối với mục tiêu. Sau đó, chuyển đổi bảng tần số thành bảng khả năng và cuối cùng sử dụng phương trình Naive Bayesian để tính xác suất hậu nghiệm cho mỗi lớp. Lớp có xác suất hậu nghiệm cao nhất là kết quả của dự đoán.

Tính toán xác suất và dự đoán biến mục tiêu

Từ tập dữ liệu, ta đã xây dựng bảng tần số cho tính năng "Outlook", sau đó ta lấy bảng khả năng như hiển thị bên dưới, sau đó tính xác suất sau của lớp là "Yes", với tính năng "Outlook" là "Sunny".



Sau đó, ta tính toán xác suất sau của lớp là "No" với tính năng "Outlook" là "Sunny"



⇒ Ta có xác suất về việc chơi golf cao hơn xác suất không chơi golf. Do đó, chúng ta có thể nói rằng khi trời “Sunny” sẽ chơi golf; Tương tự, ta có thể tính xác suất của "Overcast" và "Rainny".

⇒ Áp dụng tương tự, ta được bảng tần suất và khả năng xảy ra của cả bốn yếu tố dự đoán như sau:

Frequency Table

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	3/9	2/5
	Overcast	4/9	0/5
	Rainy	2/9	3/5

		Play Golf	
		Yes	No
Humidity	High	3/9	4/5
	Normal	6/9	1/5

		Play Golf	
		Yes	No
Temp.	Hot	2/9	2/5
	Mild	4/9	2/5
	Cool	3/9	1/5

		Play Golf	
		Yes	No
Windy	False	6/9	2/5
	True	3/9	3/5

Từ các bảng này, chúng ta có thể tính toán xác suất riêng cho từng lớp (Yes, No), cho các yếu tố dự đoán nhất định và tìm ra xác suất tối đa sẽ là biến mục tiêu.

$$P(x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Đối với tất cả các mục trong tập dữ liệu, mẫu số không thay đổi và giữ nguyên. Do đó, mẫu số đã bị loại bỏ và tỷ lệ được đưa vào phương trình trên.

Trong trường hợp này, lớp/biến mục tiêu *(y)* chỉ có hai kết quả là "Yes" hoặc "No". Nhưng có thể có trường hợp việc phân loại có thể có nhiều biến. Vì vậy, chúng ta cần tìm lớp y có xác suất lớn nhất. Sử dụng argmax vì đây là thao tác tìm đối số mang lại giá trị lớn nhất từ hàm đích.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Hãy lấy một ví dụ để thực hiện phương trình trên, ta hãy ghi lại các đặc điểm mới và tìm hiểu xem ngày cụ thể này có phù hợp để chơi golf hay không.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Cool	High	TRUE	?

Ta tính toán các xác suất hậu nghiệm riêng lẻ cho lớp "Yes" và "No" cho các yếu tố dự đoán *(tính năng)* khác nhau:

Yes:

$$P(X) = P(Yes) * P(Yes) * P(Yes) * P(Yes) * P(Yes) = \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.00529$$

No:

$$P(X) = P(No) * P(No) * P(No) * P(No) * P(No) = \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.02075$$

Ta có thể thấy rằng, đối với các tính năng/dự đoán nhất định, xác suất của lớp là "No" lớn hơn xác suất của lớp là "Yes". Do đó, trò chơi gôn sẽ được "No" trong ngày đặc biệt này.

3.4. PHÉP SỬA LỖI LAPLACE

Hãy lấy một ví dụ về phân loại văn bản trong đó nhiệm vụ là phân loại xem Review là “positive” hay “negative”. Xây dựng bảng khả năng dựa trên training dataset. Trong khi truy vấn Review,

chúng ta sử dụng các giá trị bảng khả năng, nhưng điều gì sẽ xảy ra nếu một từ trong Review không có trong tập training dataset?

$$Review = w1\ w2\ w3\ w'$$

Chúng ta có 4 từ trong chuỗi Review và giả sử chỉ có w1, w2 và w3 có trong training dataset. Vì vậy, chúng ta sẽ có khả năng cho những từ đó. Để tính toán xem Review là "positive" hay "negative", chúng ta so sánh $P(positive|review)$ và $P(negative|review)$.

$$P(review) = P(positive) * P(positive) * P(positive) * P(positive) * P(positive) *$$

Trong bảng khả năng, chúng ta có $P(positive), P(positive), P(positive)$ và $P(positive)$ nhưng không hề có $P(positive)$.

Nếu từ này không có trong training dataset, thì chúng ta không có khả năng của nó. Chúng ta nên làm gì?

- **Cách tiếp cận 1** – bỏ qua w'
  Bỏ qua (từ này) có nghĩa là chúng ta gán cho nó giá trị 1, tức là xác suất xuất hiện của w' trong đánh giá tích cực $P(positive)$ và đánh giá tiêu cực $P(negative)$ là 1. Cách tiếp cận này có vẻ không hợp lý về mặt logic.
- **Cách tiếp cận 2** – tính xác xuất của w' và vì không tìm thấy nên nó bằng 0
  Khi đó $P(positive) = 0$ và $P(negative) = 0$ Đây là vấn đề xác suất bằng 0. Vậy làm thế nào để đối phó với vấn đề này?

Laplace smoothing là một kỹ thuật làm mịn xử lý vấn đề xác suất bằng 0 trong Naïve Bayes. Sử dụng Laplace smoothing, chúng ta có thể biểu diễn $P(positive)$ là:

$$P(positive) = \frac{0 + \alpha}{N + \alpha * K}$$

Trong đó:

α : tham số làm mịn

N : tổng số Review "positive"

K : số tính năng/phân lớp trong tập dữ liệu

Nếu chúng ta chọn giá trị alpha khác 0, xác suất sẽ không còn bằng 0 ngay cả khi một từ không có trong training dataset.

Khi α tăng, xác suất khả năng di chuyển theo hướng phân bố đồng đều (0,5). Gần như trong mọi trường hợp, $\alpha = 1$ đang được sử dụng để loại bỏ vấn đề xác suất bằng 0.

3.5. CÁC LOẠI NAÏVE BAYES

Đa thức: Phân loại đa thức Naïve Bayes được sử dụng khi dữ liệu được phân phối đa thức. Nó chủ yếu được sử dụng cho các vấn đề phân loại tài liệu, tức là một tài liệu cụ thể thuộc về danh mục nào, chẳng hạn như Thể thao, Chính trị, Giáo dục, v.v. Trình phân loại sử dụng tần suất các từ trong tài liệu này làm yếu tố dự đoán/đặc điểm.

Bernoulli: Phân loại Bernoulli hoạt động tương tự như phân loại đa thức, nhưng các biến dự đoán là các biến Boolean độc lập. Chẳng hạn như một từ cụ thể có hiện diện hay không trong tài liệu.

Gaussian: Khi các yếu tố dự đoán lấy các giá trị liên tục thay vì rời rạc, thì mô hình giả định rằng các giá trị này được lấy mẫu từ phân bố Gaussian. Do cách các giá trị hiện diện trong tập dữ liệu thay đổi nên công thức tính xác suất có điều kiện thay đổi thành:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Trong đó μ là giá trị trung bình và σ là độ lệch chuẩn:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma = [\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2]^{0.5}$$

3.6. ƯU ĐIỂM VÀ NHƯỢC ĐIỂM CỦA NAÏVE BAYES

Ưu điểm:

- Dễ hiểu và đơn giản để sử dụng.
- Cần ít dữ liệu huấn luyện hơn.
- Xử lý tốt dữ liệu liên tục và dữ liệu rời rạc.
- Có khả năng mở rộng tốt khi có số lượng lớn các biến dự đoán và điểm dữ liệu.
- Có thể sử dụng để đưa ra dự đoán theo thời gian thực và có tốc độ nhanh.

- Bỏ qua với các đặc điểm không quan trọng.
- Dự đoán cho lớp của tập dữ liệu kiểm tra nhanh chóng và dễ dàng.
- Hoạt động tốt hơn so với các mô hình học máy khác như hồi quy logistic hoặc cây quyết định và cần ít dữ liệu huấn luyện hơn khi giả định độc lập là đúng.
- Hoạt động tốt với các biến đầu vào phân loại hơn là biến số. Với các biến số, nó giả định rằng chúng có phân phối chuẩn (đường cong hình chuông, đây là một giả định mạnh).

Nhược điểm:

- Naïve Bayes đưa ra giả định không phổ biến và không thực tế rằng tất cả các biến dự đoán (hoặc đặc trưng) đều độc lập. Điều này hạn chế tính ứng dụng của thuật toán trong các trường hợp sử dụng thực tế.
- Do một số ước lượng có thể không chính xác, ta không nên tin tưởng tuyệt đối vào các xác suất đầu ra của nó.
- Nếu một biến phân loại (trong tập dữ liệu kiểm tra) có một danh mục không xuất hiện trong tập dữ liệu huấn luyện, mô hình sẽ gán xác suất bằng 0 và không thể dự đoán được bất cứ điều gì. Điều này thường được gọi là Tần số Bằng Không. Chúng ta có thể sử dụng phương pháp làm mịn để khắc phục vấn đề này, trong đó, ước lượng Laplace là một trong những kỹ thuật làm mịn đơn giản nhất.

Naïve Bayes là một thuật toán phân loại xác suất, tạo ra dự đoán bằng cách ước lượng khả năng của mỗi lớp dựa trên dữ liệu đầu vào. Thuật toán này đưa ra giả định (đôi khi không chính xác) rằng các đặc trưng là độc lập với nhau, mặc dù sự đơn giản hóa này giúp tính toán chính xác hơn. Hệ thống học các xác suất của mỗi đặc trưng đối với từng lớp bằng cách huấn luyện trên dữ liệu có nhãn. Sau đó, khi được cung cấp dữ liệu mới không có nhãn, hệ thống sẽ xác định xác suất của mỗi lớp dựa trên các đặc trưng quan sát được và chọn lớp có xác suất cao nhất làm dự đoán.

3.7. SO SÁNH MÔ HÌNH NAÏVE BAYES VÀ MÔ HÌNH HỒI QUY LOGISTIC

Mô hình Naïve Bayes	Mô hình hồi quy logistic

Dựa trên định lý Bayes, phương pháp phân loại này xác định đặc trưng quan trọng nhất trong phân loại bằng cách tính toán xác suất của mỗi đặc trưng dựa trên các thuộc tính được cung cấp. Vì vậy, phương pháp Bayes này được gọi là "Naive" (Đơn Giản).	Xem xét khả năng của một mẫu trong một lớp và phân loại mẫu đó một cách tuyến tính dựa trên xác suất. Tìm đường biên giữa hai hoặc nhiều lớp và các mẫu của chúng để phân chia các lớp dựa trên hành vi là bước quan trọng nhất trong quá trình này.
Mô hình sinh này nhắm đến mục tiêu B với đặc trưng A để định lý P(B|A) có thể được sử dụng nhằm tính xác suất giữa hai yếu tố này. Điều này làm rõ liệu A có xảy ra mỗi khi B xảy ra hay không, giúp phân loại trở nên đơn giản hơn.	Mô hình phân biệt này tính toán xác suất trực tiếp bằng cách ánh xạ từ A sang B hoặc từ B sang A để có thể xác định liệu B có xảy ra trong một khoảng thời gian cụ thể do A gây ra hay không.
Vì mỗi đặc trưng được coi là độc lập, việc phân loại được thực hiện theo cách sinh tạo. Nếu có bất kỳ đặc trưng nào có mối liên kết, quá trình phân loại sẽ không diễn ra như mong đợi.	Các đặc trưng được chia theo cách tuyến tính, vì vậy ngay cả khi có sự tương quan giữa các đặc trưng, hồi quy logistic vẫn hỗ trợ phân tích dữ liệu tốt nhờ vào phân loại tuyến tính và mang lại kết quả tốt hơn so với Naive Bayes.
Ta có thể thực hiện phân loại tốt hơn dựa trên số lượng đặc trưng góp phần tạo ra xác suất tốt cho dữ liệu, ngay cả trước khi phân tích dữ liệu, khi tổng lượng dữ liệu được xem xét là nhỏ hoặc kích thước mẫu ít. Điều này hỗ trợ gián tiếp trong việc dự báo thị trường, giúp các nhà phân tích nhận diện đặc trưng nổi bật.	Ít dữ liệu không thuận lợi cho hồi quy logistic vì mô hình kết quả với các đặc trưng đã cho sẽ trở nên rộng hơn. Các kỹ thuật hồi quy sẽ giúp giảm hiện tượng overfitting, nhưng kết quả thu được sẽ không như mong đợi, và phân tích sẽ không giúp ích nhiều trong việc hiểu rõ dữ liệu.
Naive Bayes có *phương sai thấp hơn* và *độ lệch cao hơn*. Kết quả được nghiên cứu để hiểu cách thức sản xuất dữ liệu, giúp dự đoán dễ dàng hơn với ít biến và ít thông tin hơn. Khi chỉ có một số bộ dữ liệu huấn luyện nhỏ, Naive Bayes cung cấp một giải pháp nhanh chóng hơn.	Hồi quy logistic có *độ lệch thấp* và *phương sai cao*. Với các biến phân loại và liên tục, xác suất được dự đoán theo một cách hàm gián tiếp, dẫn đến một tập hợp các kết quả phân loại. Hồi quy logistic đa lớp được sử dụng trong phân tích dữ liệu khi có nhiều lớp.

3.8. ÁP DỤNG NAÏVE BAYES TẠO MÔ HÌNH PHÂN LOẠI

3.8.1. Dataset

SampleID	Size	Number	Thickness	Lung Cancer
1	ex-large	scale-2	grey	stage-3
2	median	scale-3	grey	stage-1
3	small	scale-1	grey	negative
…	…	…	…	…
1977	median	scale-4	dark	stage-1
1978	large	scale-3	dark	stage-1

Download full dataset tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/datasets/naive-bayes/naive-bayes.csv

Với dataset trên ta có:

Các biến độc lập (X) là:

- Size *(Kích thước)*: Kích thước của mẫu được chia thành các mức độ, từ kích thước rất lớn *(ex-large)*, lớn *(large)*, trung bình *(median)*, nhỏ *(small)*, và không có kích thước *(none)*.
- Number *(Số lượng)*: Số lượng của mẫu được biểu diễn qua các thang đo từ 1 đến 5, từ scale-1 *(ít nhất)* đến scale-5 *(nhiều nhất)*.
- Thickness *(Độ dày)*: Độ dày của mẫu chia thành các mức độ, từ light *(mỏng nhất)*, grey *(trung bình)*, đến dark *(dày nhất)*.

Biến phụ thuộc (Y) là: Lung Cancer *(Ung thư phổi)*: Tình trạng ung thư phổi của mẫu, với các giá trị gồm:

- negative: Không có ung thư phổi.
- stage-1: Giai đoạn 1 của ung thư phổi.
- stage-2: Giai đoạn 2 của ung thư phổi.
- stage-3: Giai đoạn 3 của ung thư phổi.

Số lượng mẫu (N): 1978

3.8.2. Mẫu cần dự đoán

SampleID	Hotspot-size	Spot-number	Thickness
1	none	scale-4	light
2	large	scale-1	light
3	small	scale-2	grey
4	median	scale-5	dark
5	none	scale-1	dark
6	ex-large	scale-3	light
7	none	scale-5	dark
8	small	scale-1	grey
9	median	scale-4	light
10	small	scale-2	grey

3.8.3. Xây dựng mô hình bằng hàm excel

Đầu tiên, chúng ta phải tính số lượng dữ liệu đầu vào (Sample size) bằng hàm COUNT để đếm số dòng dữ liệu. Sau khi tìm được số lượng dữ liệu đầu vào, ta sẽ phải tìm số lượng dữ liệu đầu vào của mỗi phân loại Lung cancer với công thức Excel: =COUNTIF(X1, X2)

Trong đó:

- X1 là vùng có ô dữ liệu cần đếm
- X2 là điều kiện để đếm ô dữ liệu đấy

Sample size	1978	Lung Cancer			
		negative	stage-1	stage-2	stage-3
	count	491	480	502	505

Tiếp theo, tìm xác xuất của mỗi loại Lung Cancer bằng cách lấy số lượng dữ liệu đầu vào của mỗi phân loại mục tiêu chia cho số lượng toàn bộ dữ liệu:

Sample size	1978	Lung Cancer			
		negative	stage-1	stage-2	stage-3
	count	491	480	502	505
	probability	0.248230536	0.24266936	0.253791709	0.255308392

Bây giờ chúng ta có thể bắt đầu tính xác xuất có điều kiện của từng đặc trưng trong từng phân lớp (Lung Cancer). Ví dụ: Tìm xác xuất xuất hiện loại ex-large của đặc trưng size với loại negative của mục tiêu Lung Cancer P(Size = ex-large | $C_{negative}$)

$$= COUNTIFS(X1, Z1, X2, Z2)/B1$$

Trong đó:

- X1 là vùng dữ liệu cột Lung Cancer
- Z1 là điều kiện dữ liệu ô thuộc phân loại để đếm ô Lung Cancer
- X2 là vùng dữ liệu cột đặc trưng
- Z1 là điều kiện dữ liệu ô thuộc phân loại để đếm ô cột đặc trưng
- B1 là số lượng dữ liệu thuộc phân loại mục tiêu

Hàm COUNTIF giúp đếm được các dữ liệu thỏa hai điều kiện trên.

Sample size	1978	Lung Cancer			
		negative	stage-1	stage-2	stage-3
	count	491	480	502	505
	probability	0.248230536	0.24266936	0.253791709	0.255308392
Size	ex-large	0.187372709	0.21666667	0.201195219	0.205940594
	large	0.181262729	0.21875	0.187250996	0.221782178
	median	0.236252546	0.19583333	0.167330677	0.201980198
	small	0.187372709	0.19583333	0.215139442	0.192079208
	none	0.207739308	0.17291667	0.229083665	0.178217822
Number	scale-1	0.203665988	0.20625	0.211155378	0.213861386
	scale-2	0.199592668	0.20416667	0.187250996	0.207920792
	scale-3	0.211812627	0.20625	0.203187251	0.198019802
	scale-4	0.211812627	0.2	0.199203187	0.186138614
	scale-5	0.17311609	0.18333333	0.199203187	0.194059406
Thickness	light	0.356415479	0.36041667	0.338645418	0.324752475
	grey	0.364562118	0.2875	0.348605578	0.330693069
	dark	0.279022403	0.35208333	0.312749004	0.344554455

Sau khi có được các xác xuất cần thiết, ta có thể dự đoán được dữ liệu đầu vào thuộc loại Lung Cancer nào.

Scoring Data				
SampleID	Hotspot-size	Spot-number	Thickness	Lung Cancer
1	none	scale-4	light	
2	large	scale-1	light	
3	small	scale-2	grey	
4	median	scale-5	dark	
5	none	scale-1	dark	
6	ex-large	scale-3	light	
7	none	scale-5	dark	
8	small	scale-1	grey	
9	median	scale-4	light	
10	small	scale-2	grey	

Đầu tiên, chúng ta cần phải tính xác xuất xuất hiện của mỗi loại Lung Cancer với dữ liệu đầu vào. Công thức: VB

$$P(X|Cy) = P(X1|Cy) * P(X2|Cy) * P(X3|Cy) * P(Cy)$$

Trong đó:

- y là một loại Lung Cancer
- X là thuộc tính đầu vào
- $P(X_z|C)$ là xác xuất xuất hiện của một loại Z trong thuộc tính X với kết quả C

$$= SUMIF(X1, Z1, B1) * SUMIF(X2, Z2, B2) * SUMIF(X3, Z3, B3) * C1$$

Trong đó:

- X1, X2, X3: Đây là các vùng dữ liệu trong các cột có chứa đặc trưng bạn muốn kiểm tra.
- Z1, Z2, Z3: Đây là các điều kiện tương ứng với từng đặc trưng. Mỗi điều kiện xác định những giá trị nào trong cột đặc trưng sẽ được tính tổng.
- B1, B2, B3: Đây là các vùng chứa giá trị cần cộng lại khi điều kiện được thỏa mãn.
- C1: Đây là xác suất của một phân loại mục tiêu, giúp điều chỉnh tổng có điều kiện theo xác suất này.

Sau khi áp dụng công thức cho các thông tin đầu vào, chúng ta sẽ dùng hàm MAX để tìm ra xác xuất nào lớn nhất:

P(t|n)	P(t|1)	P(t|2)	P(t|3)	P(max)
0.003892981	0.00302473	0.003922049	0.002750459	0.003922049
0.00326617	0.003946043	0.0033982	0.003932571	0.003946043
0.003384369	0.002789486	0.00356415	0.003371852	0.00356415
0.00283275	0.003067528	0.002645728	0.003447994	0.003447994
0.002930431	0.003047132	0.003839455	0.003352791	0.003839455
0.003511316	0.003908462	0.003513474	0.003381179	0.003908462
0.002490866	0.002708562	0.003622128	0.003042347	0.003622128
0.003453437	0.002817951	0.004019148	0.00346819	0.004019148
0.004427312	0.003425598	0.002864801	0.003117187	0.004427312
0.003384369	0.002789486	0.00356415	0.003371852	0.00356415

Với dữ liệu đã tìm được ở trên, chúng ta sẽ lấy được xác xuất lớn nhất của mỗi dòng và tương ứng với nó lấy được phân loại kết quả nào lớn nhất thì chúng ta điền kết quả đó vào phần Lung Cancer

Scoring Data				
SampleID	**Hotspot-size**	**Spot-number**	**Thickness**	**Lung Cancer**
1	none	scale-4	light	stage-2
2	large	scale-1	light	stage-1
3	small	scale-2	grey	stage-2
4	median	scale-5	dark	stage-3
5	none	scale-1	dark	stage-2
6	ex-large	scale-3	light	stage-1
7	none	scale-5	dark	stage-2
8	small	scale-1	grey	stage-2
9	median	scale-4	light	negative
10	small	scale-2	grey	stage-2

3.8.4. Xây dựng mô hình bằng code Python

3.8.4.1. Không sử dụng thư viện

Bước 1: Tạo class *UseNaive.py* với phương thức khởi tạo _init_ và các phương thức *training(), testing(), predict(), calculate_class_prior_probabilities(), calculate_conditional_probabilities().*

- Phương thức khởi tạo đọc dataset training sau đó chia dữ liệu thành 2 phần 1 nữa dùng để huấn luyện mô hình, nữa còn lại dùng để test. Ngoài ra hàm khởi tạo còn khởi tạo biến để lưu xác suất của các phân lớp và các xác suất có điều kiện.
- Phương thức *calculate_class_prior_probabilities()* dùng để tính xác suất của từng phân lớp trong y.
- Phương thức *calculate_conditional_probabilities()* dùng để tính xác suất có điều kiện của các đặc trưng ở từng phân lớp.
- Phương thức *training()* gọi lần lượt 2 hàm *calculate_class_prior_probabilities()* và *calculate_conditional_probabilities()* để lấy các biến xác suất lưu vào biến đã khởi tạo ở hàm *_init_()*.
- Phương thức *predict()* dùng để dự đoán phân lớp cho dữ liệu dự đoán.
- Phương thức *test()* dùng để dự đoán phân lớp cho tập dữ liệu test sau đó đánh giá độ chính xác của mô hình.

Bước 2: Trong phương thức khởi tạo viết code để lấy dữ liệu từ file csv lưu vào dataframe.

```
def __init__(self, datasetURL, train_size=0.5):
    csv_handler = CSVHandler(datasetURL)
    dataframe = csv_handler.read_csv()
```

Bước 3: Trong phương thức khởi tạo tiếp tục viết code để chuẩn hóa dữ liệu từ chữ sang số để mô hình có thể huấn luyện (Lớp Normalization sẽ được viết ở chương IV Data Mining).

```
    dataframe, label_encoders = Normalization.encode_dataframe(dataframe)
```

Bước 4: Trong phương thức khởi tạo tiếp tục viết code để chia dữ liệu thành 2 tập dùng để training và testing.

```
    X = dataframe.iloc[:, :-1]   # Chọn tất cả các hàng và cột trừ cột cuối cùng
    y = dataframe.iloc[:, -1]    # Chọn cột cuối cùng

    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(dataframe) * train_size)
    self.X_train = X.iloc[:split_index]
    self.y_train = y.iloc[:split_index]
```

```
    self.X_test = X.iloc[split_index:]
    self.y_test = y.iloc[split_index:]
```

Bước 5: Khởi tạo các biến để lưu trữ xác suất của các phân lớp và các xác suất có điều kiện của các đặc trưng ở từng phân lớp.

```
    self.class_priors = defaultdict(float)
    self.conditional_probs = defaultdict(lambda: defaultdict(lambda: defaultdict(float)))
```

conditional_probs là 1 mảng 3 chiều với dạng như sau:

conditional_probs[feature][class_label][value] trong đó *feature* chính là cột $X_{feature}$, *class_label* chính là phân lớp y_i, *value* chính là giá trị của 1 đặc trưng của cột $X_{feature}$

Bước 6: Viết hàm *calculate_class_prior_probabilities()* với tham số đầu vào là *ytrain* dùng để tính xác suất của các phân lớp trong training dataset.

```
def calculate_class_prior_probabilities(self, y):
    # Tính xác suất của từng phân lớp trong y (sử dụng sửa lỗi Laplace)
    n_classes = len(y.unique())
    total_samples = len(y)

    for class_label in y.unique():
      class_count = sum(y == class_label)
      self.class_priors[class_label] = (class_count + 1) / (total_samples + n_classes)
```

Bước 7: Viết hàm *calculate_conditional_probabilities()* nhận tham số đầu vào là *Xtrain* và *ytrain* dùng để tính xác suất có điều kiện cho từng đặc trưng ở từng phân lớp.

$$P = (X_{feature} = value | y = class\_label)$$

```
def calculate_conditional_probabilities(self, X, y):
    for feature in X.columns:
      unique_values = X[feature].nunique()  # Số lượng giá trị khác nhau cho mỗi đặc trưng
      for class_label in y.unique():
        # Chọn ra các hàng có nhãn phân lớp class_label
        rows_in_class = X[y == class_label]
        for value in range(unique_values):
          count_value_in_class = sum(rows_in_class[feature] == value)
          # Áp dụng sửa lỗi Laplace cho xác suất điều kiện
          self.conditional_probs[feature][class_label][value] = (count_value_in_class + 1) /
(len(rows_in_class) + unique_values)
```

Bước 8: Viết hàm *training()* gọi lần lượt đến 2 hàm *calculate_class_prior_probabilities()* và *calculate_conditional_probabilities()* để lấy các xác suất được training bởi 2 hàm này.

```
def training(self):
    self.calculate_class_prior_probabilities(self.ytrain)
    self.calculate_conditional_probabilities(self.Xtrain, self.ytrain)
```

Bước 9: Trong hàm *predict()* viết code đọc dataset từ file csv và lưu vào dataframe.

```
    path='Dataset\predict\\naive-bayes-nolib.csv'
    csv_handler = CSVHandler(path)
    predictDataframe = csv_handler.read_csv()
```

Bước 10: Trong hàm *predict()* viết code dự đoán phân lớp cho từng mẫu

```
    predictions = []
    for _, row in input_data.iterrows():
      best_class, best_prob = None, -1
      for class_label in self.class_priors:
        # Tính toán xác suất cho phân lớp hiện tại
        prob = self.class_priors[class_label]
        for feature, value in zip(input_data.columns, row):
          prob *= self.conditional_probs[feature][class_label].get(value, 1e-9)
          # 1e-9 để tránh xác suất 0
        # Lưu lại phân lớp có xác suất cao nhất
        if prob > best_prob:
          best_class, best_prob = class_label, prob
      predictions.append(best_class)
y_pred = pd.DataFrame(predictions)
return y_pred
```

Bước 12: Gỉa sử đã có file *ValidateNoLib.py* (được viết ở chương VII). Trong hàm *testing()* viết code để đánh giá mô hình và hiển thị các thông tin như: Cỡ mẫu test, Các phân lớp, Ma trận nhầm lẫn, Độ chính xác.

```
    predictions_df = pd.DataFrame(predictions, columns=['Lung Cancer'])
    validateNoLib = ValidateNoLib(self.ytest, predictions_df)
    print("Số lượng mẫu test:", validateNoLib.getSampleSize())
    print("Các phân lớp:", validateNoLib.getSampleClasses())
    print("Ma trận nhầm lẫn:\n", validateNoLib.confusionMatrix())
    print("Độ chính xác:", validateNoLib.accuracy())
```

Bước 13: Vào file *main.py* gọi các hàm trên và xem kết quả.

Download source code tại:

https://github.com/NT02IT/BasicMachineLearning/blob/main/NaiveBayes/UseNaive.py

3.8.4.2. Sử dụng mô hình Naïve Bayes thuộc thư viện sklearn

Bước 1: Tạo class *UseSklearn.py* với phương thức khởi tạo _init_ và các phương thức *training(), testing(), predict()*

- Phương thức khởi tạo đọc dataset training sau đó chia dữ liệu thành 2 phần 1 nữa dùng để huấn luyện mô hình, nữa còn lại dùng để test.
- Phương thức *training()* gọi hàm *fit()* của model GaussianNB thuộc thư viện sklearn để huấn luyện mô hình.
- Phương thức *predict()* dùng để dự đoán phân lớp cho dữ liệu dự đoán.
- Phương thức *test()* dùng để dự đoán phân lớp cho tập dữ liệu test sau đó đánh giá độ chính xác của mô hình.

Bước 2: Trong phương thức khởi tạo viết code tương tự như class *UseNaive.py* và bổ sung code khởi tạo model *self.model=GaussianNB()*

Bước 3: Trong phương thức *training()* gọi đến hàm *fit()* để huấn luyện mô hình: *self.model.fit(self.Xtrain, self.ytrain)*

Bước 4: Trong hàm *predict(input_data)* viết code để dự đoán đầu ra cho dữ liệu đầu vào:

return self.model.predict(input_data);

Bước 5: Trong hàm *testing()* viết code dự đoán phân lớp cho tập testing và in ra kết quả test.

```
def testing(self):
  # Dự đoán trên tập testing
  y_pred = self.model.predict(self.X_test)
```

Bước 6: Giả sử đã có file *ValidateNoLib.py* (được viết ở chương VII). Trong hàm *testing()* tiếp tục viết code để đánh giá mô hình và hiển thị các thông tin như: Cỡ mẫu test, Các phân lớp, Ma trận nhầm lẫn, Độ chính xác.

```
  validateNoLib = ValidateNoLib(self.ytest, y_pred)
  print("Số lượng mẫu test:", validateNoLib.getSampleSize())
  print("Các phân lớp:", validateNoLib.getSampleClasses())
  print("Ma trận nhầm lẫn:\n", validateNoLib.confusionMatrix())
  print("Độ chính xác:", validateNoLib.accuracy())
```

Download source code tại:

https://github.com/NT02IT/BasicMachineLearning/blob/main/NaiveBayes/UseSklearn.py

3.8. SO SÁNH VỚI THƯ VIỆN SKLEARN

3.8.1. Tổng quan

Thực hiện 2 thuật toán cùng trên 1 dataset gồm 13 thuộc tính (14 cột) và 45,221 dòng dữ liệu, so sánh được thực hiện trên các điều kiện môi trường như nhau:

- Thực hiện trên cùng 1 máy tính, cùng môi trường python
- Dùng 50% số dòng (22611 dòng) để huấn luyện và 50% còn lại để test.
- Thời gian được so sánh thông qua thời gian thực hiện hàm *train()*

Tiêu Chí	Use Naïve	Use Sklearn
Thời gian chạy	1.315s	0.218s
Độ chính xác	81,49%	79.74%

Ma trận nhầm lẫn:

Use Sklearn	Actual (<=50K)	Actual (>50K)
Predict (<=50K)	16158	844
Predict (>50K)	3736	1873

No Lib	Actual (<=50K)	Actual (>50K)
Predict (<=50K)	14327	2675
Predict (>50K)	1510	4099

3.8.2. Nhận xét

Thời gian chạy: Sklearn thực hiện nhanh gấp 6 lần so với sử dụng xác suất Naïve về thời gian chạy, Sklearn mất 0.218 giây so với 1.315 giây của xác suất Naïve. Sự chênh lệch này là rất lớn, cho thấy Sklearn hiệu quả hơn nhiều về mặt tính toán trên tập dữ liệu lớn. Đối với xác suất Naïve

cần được tối ưu lại thuật toán về mặt thời gian để phù hợp với dữ liệu lớn, cũng như áp dụng vào thực tiễn. ⇒ Vì Sklearn chủ động dùng thuật toán để tối ưu cho việc tính xác suất gần đúng để có lợi hơn về mặt thời gian.

Độ chính xác: Sklearn và xác suất Naïve thể hiện độ chính xác cao, sự dụng tốt cho dữ liệu lớn và khả thi trong việc áp dụng thuật toán vào thực tế.

CHƯƠNG IV: DATA MINING

4.1. ĐỊNH NGHĨA KHAI THÁC DỮ LIỆU *(DATA MINING)*

Khai thác dữ liệu *(Data Mining)* là quá trình phân tích một lượng lớn dữ liệu để tìm kiếm các mẫu *(patterns),* xu hướng *(trends)* và thông tin hữu ích. Đây là một bước trong quy trình khám phá tri thức từ cơ sở dữ liệu *(Knowledge Discovery in Databases - KDD)*, giúp doanh nghiệp và tổ chức đưa ra quyết định dựa trên dữ liệu.

Dữ liệu dùng để khai thác có thể được thu thập từ nhiều nguồn khác nhau, tùy thuộc vào mục đích và lĩnh vực của phân tích:

- Dữ liệu nội bộ doanh nghiệp: Cơ sở dữ liệu khách hàng, Dữ liệu bán hàng, Dữ liệu vận hành,…
- Dữ liệu web và mạng xã hội: Dữ liệu từ website, Dữ liệu mạng xã hội,…
- Dữ liệu từ các cảm biến và thiết bị IoT (Internet of Things): Cảm biến công nghiệp, Thiết bị IoT trong nhà, Dữ liệu từ xe thông minh,…
- Dữ liệu công khai và mở (Open Data): Dữ liệu chính phủ, Dữ liệu giáo dục và y tế,…
- Dữ liệu nghiên cứu và khoa học: Dữ liệu từ các dự án nghiên cứu, Dữ liệu gen và sinh học,…
- Dữ liệu giao dịch tài chính: Dữ liệu thị trường chứng khoán, Dữ liệu thanh toán và ngân hàng,…
- Dữ liệu từ khảo sát và bảng câu hỏi: Khảo sát khách hàng và người dùng, Khảo sát thị trường,…
- Dữ liệu từ các trang thu thập và tổng hợp dữ liệu: Dữ liệu thu thập từ web *(web scraping),* Dữ liệu tổng hợp

4.2. QUY TRÌNH KHAI THÁC DỮ LIỆU

Quy trình tiêu chuẩn liên ngành đối với khai thác dữ liệu *(Cross-Industry Standard Process for Data Mining, CRISP-DM)* là một hướng dẫn tuyệt vời để bắt đầu quy trình khai thác dữ liệu. CRISP-DM vừa là phương pháp luận, vừa là mô hình quy trình trung lập với ngành, công cụ và ứng dụng.

Với vai trò phương pháp luận, CRISP-DM mô tả các giai đoạn điển hình trong một dự án khai thác dữ liệu, phác thảo những nhiệm vụ liên quan trong mỗi giai đoạn và giải thích mối quan hệ giữa những nhiệm vụ này.

Thông qua các giai đoạn CRISP-DM linh hoạt, đội ngũ dữ liệu có thể di chuyển qua lại giữa các giai đoạn nếu cần. Ngoài ra, công nghệ phần mềm có thể thực hiện hoặc hỗ trợ một số nhiệm vụ này.

Giai đoạn 1. Hiểu biết về doanh nghiệp

Nhà khoa học dữ liệu hoặc người khai thác dữ liệu bắt đầu bằng cách xác định các mục tiêu và phạm vi dự án. Họ hợp tác với các bên liên quan của doanh nghiệp để xác định một số thông tin nhất định.

- Vấn đề cần giải quyết
- Ràng buộc hoặc giới hạn của dự án
- Tác động kinh doanh của các giải pháp tiềm năng

Sau đó, họ sử dụng thông tin này để xác định mục tiêu khai thác dữ liệu cũng như nhận định tài nguyên cần có để khai phá kiến thức.

Giai đoạn 2. Hiểu biết về dữ liệu

Khi đã nắm được vấn đề kinh doanh, các nhà khoa học dữ liệu bắt đầu phân tích sơ bộ dữ liệu. Họ thu thập các tập dữ liệu từ nhiều nguồn khác nhau, lấy được quyền truy cập và chuẩn bị báo cáo mô tả dữ liệu. Báo cáo này bao gồm các loại dữ liệu, số lượng cũng như yêu cầu về phần cứng và phần mềm để xử lý dữ liệu. Sau khi được doanh nghiệp phê duyệt kế hoạch, các nhà khoa học dữ liệu bắt đầu khám phá và xác minh dữ liệu. Họ thao tác dữ liệu bằng các kỹ thuật thống kê cơ bản, đánh giá chất lượng dữ liệu và chọn tập dữ liệu cuối cùng cho giai đoạn tiếp theo.

Giai đoạn 3. Chuẩn bị dữ liệu

Người khai thác dữ liệu dành nhiều thời gian nhất cho giai đoạn này do phần mềm khai thác dữ liệu yêu cầu dữ liệu chất lượng cao. Các quy trình kinh doanh thu thập và lưu trữ dữ liệu vì nhiều

lý do khác ngoài việc khai thác và người khai thác dữ liệu phải tinh chỉnh dữ liệu trước khi sử dụng để lập mô hình. Chuẩn bị dữ liệu bao gồm các quy trình sau đây.

- Làm sạch dữ liệu: xử lý dữ liệu bị thiếu, lỗi dữ liệu, giá trị mặc định và hiệu chỉnh dữ liệu.
- Tích hợp dữ liệu: kết hợp hai tập dữ liệu riêng biệt để có được tập dữ liệu đích cuối cùng.
- Định dạng dữ liệu: chuyển đổi loại dữ liệu hoặc cấu hình dữ liệu cho công nghệ khai thác cụ thể đang được sử dụng.

Giai đoạn 4. Lập mô hình dữ liệu

Người khai thác dữ liệu nhập dữ liệu đã chuẩn bị vào phần mềm khai thác dữ liệu và nghiên cứu kết quả. Để làm điều này, họ có thể chọn trong số nhiều kỹ thuật và công cụ khai thác dữ liệu. Họ cũng phải viết các bài kiểm thử để đánh giá chất lượng của kết quả khai thác dữ liệu. Để lập mô hình dữ liệu, các nhà khoa học dữ liệu có thể:

- Đào tạo mô hình máy học (ML) trên các tập dữ liệu nhỏ hơn bằng kết quả đã biết.
- Sử dụng mô hình để phân tích thêm các tập dữ liệu chưa biết.
- Điều chỉnh và cấu hình lại phần mềm khai thác dữ liệu cho đến khi kết quả thỏa mãn yêu cầu.

Giai đoạn 5. Đánh giá

Sau khi tạo mô hình, người khai thác dữ liệu bắt đầu đo lường mô hình so với mục tiêu kinh doanh ban đầu. Sau đó, họ chia sẻ kết quả với các chuyên viên phân tích nghiệp vụ và thu thập phản hồi. Mô hình này có thể giải đáp cặn kẽ câu hỏi ban đầu hoặc hiển thị các kiểu mẫu mới chưa biết trước đây. Người khai thác dữ liệu có thể thay đổi mô hình, điều chỉnh mục tiêu kinh doanh hoặc xem xét sửa đổi dữ liệu tùy thuộc vào phản hồi của doanh nghiệp. Đánh giá, phản hồi và sửa đổi liên tục là một phần của quy trình khai phá kiến thức.

Giai đoạn 6. Triển khai

Trong quá trình triển khai, những bên liên quan khác sử dụng mô hình làm việc để tạo ra nghiệp vụ thông minh. Nhà khoa học dữ liệu lên kế hoạch cho quy trình triển khai, bao gồm việc truyền đạt cho những cá nhân khác về chức năng của mô hình, liên tục giám sát và duy trì ứng dụng

khai thác dữ liệu. Các chuyên viên phân tích nghiệp vụ sử dụng ứng dụng này để tạo báo cáo quản lý, chia sẻ kết quả với khách hàng và cải tiến quy trình kinh doanh.

4.3. CÁC KỸ THUẬT DATA MINING PHỔ BIẾN

4.3.1. Khai thác quy tắc liên kết

Khai thác quy tắc liên kết là quy trình tìm kiếm mối quan hệ giữa hai tập dữ liệu khác nhau, dường như không liên quan đến nhau. Câu lệnh if-then sẽ cho biết xác suất của mối quan hệ giữa hai điểm dữ liệu. Nhà khoa học dữ liệu đo lường độ chính xác của kết quả bằng các tiêu chí hỗ trợ và độ tin cậy. Các tiêu chí hỗ trợ đo lường tần suất xuất hiện của những phần tử liên quan trong tập dữ liệu, trong khi đó các tiêu chí độ tin cậy cho biết số lần câu lệnh if-then được thực hiện chính xác.

Ví dụ: khi khách hàng mua một mặt hàng, họ cũng thường mua mặt hàng thứ hai có liên quan. Các nhà bán lẻ có thể sử dụng quy trình khai thác liên kết cho dữ liệu mua hàng trước đây để xác định mối quan tâm của khách hàng mới. Họ sử dụng kết quả khai thác dữ liệu để điền dữ liệu vào mục đề xuất của các cửa hàng trực tuyến.

4.3.2. Phân loại

Phân loại là một kỹ thuật khai thác dữ liệu phức tạp đào tạo thuật toán ML để sắp xếp dữ liệu thành các danh mục riêng biệt. Kỹ thuật này sử dụng các phương pháp thống kê như cây quyết định và thuật toán láng giềng gần nhất để xác định danh mục. Trong tất cả những phương pháp này, thuật toán được lập trình trước bằng các mục phân loại dữ liệu đã biết để đoán loại phần tử dữ liệu mới.

Ví dụ: các nhà phân tích có thể đào tạo phần mềm khai thác dữ liệu bằng hình ảnh quả táo và quả xoài được gắn nhãn. Sau đó, phần mềm có thể dự đoán hình ảnh mới là quả táo, xoài hay loại trái cây khác với độ chính xác nhất định.

4.3.3. Phân cụm

Phân cụm là quá trình nhóm nhiều điểm dữ liệu lại với nhau dựa trên những điểm tương đồng của chúng. Phân cụm khác với phân loại do không thể phân biệt dữ liệu theo danh mục cụ thể nhưng có thể tìm thấy kiểu mẫu trong các điểm tương đồng của chúng. Khai thác dữ liệu tạo ra

một tập hợp cụm, trong đó mỗi tập hợp sẽ khác biệt với các nhóm khác nhưng đối tượng trong mỗi cụm sẽ phần nào đó có điểm tương đồng.

Ví dụ: phân tích cụm có thể giúp nghiên cứu thị trường khi làm việc với dữ liệu đa biến từ các cuộc khảo sát. Các nhà nghiên cứu thị trường sử dụng phân tích cụm để chia người tiêu dùng thành nhiều phân khúc thị trường và hiểu rõ hơn về mối quan hệ giữa các nhóm khác nhau.

4.3.4. Phân tích trình tự và đường xu hướng

Phần mềm khai thác dữ liệu cũng có thể tìm kiếm các kiểu mẫu mà trong đó một tập sự kiện hoặc giá trị cụ thể sẽ tạo ra nhiều sự kiện hoặc giá trị sau này. Phần mềm này có thể nhận ra một số thay đổi trong dữ liệu, xảy ra theo khoảng thời gian đều đặn hoặc thường xuyên dao động theo thời gian của các điểm dữ liệu.

Ví dụ: một doanh nghiệp có thể sử dụng phân tích đường xu hướng để phát hiện doanh số bán hàng của một số sản phẩm nhất định tăng đột biến ngay trước kỳ nghỉ lễ hoặc lưu ý thấy thời tiết càng ấm, số người truy cập trang web của họ càng tăng.

4.4. LÀM SẠCH DỮ LIỆU

Làm sạch dữ liệu *(Data Cleaning)* là quá trình xử lý và chuẩn bị dữ liệu nhằm loại bỏ hoặc chỉnh sửa các giá trị lỗi *(errors)*, dữ liệu thiếu *(missing values),* trùng lặp *(duplicates)* hoặc không phù hợp để nâng cao độ chính xác và chất lượng của dữ liệu. Đây là bước quan trọng trong xử lý dữ liệu, giúp đảm bảo rằng dữ liệu đầu vào cho phân tích hoặc mô hình học máy sẽ không bị sai lệch và kết quả thu được sẽ chính xác hơn.

Tại sao làm sạch dữ liệu lại quan trọng?

Dữ liệu thô *(raw data)* thường chứa nhiều lỗi do cách nhập liệu, các thiết bị đo lường, hoặc từ quá trình thu thập dữ liệu. Nếu dữ liệu không được làm sạch:

- Các mô hình phân tích và dự đoán có thể đưa ra kết quả sai.
- Phân tích và báo cáo có thể bị lệch hoặc không chính xác.
- Các quyết định dựa trên dữ liệu có thể không đáng tin cậy.

Các bước chính trong quá trình làm sạch dữ liệu

1. Xác định và xử lý dữ liệu thiếu *(Missing Data)*:

   - Loại bỏ các hàng hoặc cột có quá nhiều giá trị thiếu, đặc biệt khi dữ liệu thiếu là một phần lớn của tập dữ liệu.
   - Điền giá trị thay thế: Điền giá trị trung bình, trung vị, hoặc giá trị phổ biến nhất (mode) của cột đó hoặc dự đoán giá trị dựa trên các cột khác.
   - Điền giá trị bằng các thuật toán nâng cao: Sử dụng các kỹ thuật như hồi quy hoặc mô hình học máy để dự đoán giá trị thiếu.

2. Xử lý dữ liệu trùng lặp *(Duplicates)*:

   Dữ liệu trùng lặp là các bản ghi xuất hiện nhiều lần và có thể làm lệch kết quả phân tích. Việc xóa các bản ghi trùng lặp sẽ làm cho dữ liệu chính xác hơn.

3. Sửa lỗi cú pháp và định dạng:

   - Đảm bảo các cột có định dạng chính xác, ví dụ: định dạng ngày giờ, số, hoặc chuỗi.
   - Sửa lỗi chính tả và cách viết không nhất quán trong dữ liệu văn bản.

4. Phát hiện và xử lý các ngoại lệ *(Outliers)*:

   - Các giá trị ngoại lệ có thể gây ra sai lệch trong phân tích, đặc biệt trong các mô hình học máy.
   - Phân tích ngoại lệ để quyết định có cần loại bỏ, sửa đổi hay giữ lại *(trong trường hợp giá trị đó có ý nghĩa đặc biệt).*

5. Chuyển đổi dữ liệu không phù hợp *(Inconsistent Data)*:

   - Ví dụ: Các đơn vị đo lường khác nhau hoặc cách viết khác nhau cho cùng một giá trị *(như "Nam" và "M" cho giới tính nam).*
   - Chuyển đổi dữ liệu về cùng một định dạng và chuẩn hóa các đơn vị đo lường.

6. Loại bỏ hoặc gắn nhãn dữ liệu không liên quan *(Irrelevant Data)*:

   Loại bỏ các cột hoặc dòng không cần thiết cho phân tích, giúp dữ liệu tập trung hơn và quá trình xử lý nhanh hơn.

4.5. TÍCH HỢP DỮ LIỆU

Tích hợp dữ liệu *(Data Integration)* là quá trình kết hợp dữ liệu từ nhiều nguồn khác nhau để tạo ra một tập dữ liệu duy nhất, nhất quán và có cấu trúc, nhằm phục vụ cho việc phân tích, báo cáo, hoặc sử dụng trong các ứng dụng cụ thể. Đây là bước quan trọng trong quản lý dữ liệu, đặc biệt là khi dữ liệu được lưu trữ rải rác ở nhiều hệ thống hoặc định dạng khác nhau.

Mục tiêu của tích hợp dữ liệu

- Tạo ra cái nhìn tổng quan: Tổng hợp dữ liệu từ nhiều nguồn giúp tạo ra một cái nhìn đầy đủ và nhất quán.
- Giảm thiểu lỗi và trùng lặp: Đảm bảo dữ liệu hợp nhất có độ chính xác cao, nhất quán và không trùng lặp.
- Tăng cường hiệu quả phân tích: Dữ liệu tích hợp có thể cung cấp nhiều góc nhìn, giúp các nhà phân tích dễ dàng tìm ra xu hướng hoặc mối quan hệ giữa các biến.
- Đơn giản hóa quá trình truy cập dữ liệu: Người dùng chỉ cần truy cập một nguồn duy nhất mà không cần phải lấy từ nhiều nguồn khác nhau.

Các bước chính trong quá trình tích hợp dữ liệu

- Thu thập và xác định nguồn dữ liệu: Xác định các nguồn dữ liệu cần tích hợp, như các cơ sở dữ liệu nội bộ, kho dữ liệu từ bên thứ ba, hay dữ liệu từ các ứng dụng và thiết bị IoT.
- Tiền xử lý và làm sạch dữ liệu: Làm sạch và chuẩn hóa dữ liệu trước khi tích hợp, bao gồm việc xử lý dữ liệu thiếu, dữ liệu không nhất quán, hoặc các định dạng khác nhau.
- Chuyển đổi dữ liệu *(Data Transformation)*: Chuyển đổi dữ liệu từ các nguồn về cùng một định dạng hoặc cấu trúc, thường thông qua các công cụ ETL *(Extract, Transform, Load - Trích xuất, Chuyển đổi, Tải).*
- Đồng bộ và hợp nhất dữ liệu: Liên kết các bảng hoặc các trường *(fields)* có cùng nội dung từ các nguồn khác nhau, loại bỏ trùng lặp và đảm bảo tính nhất quán.
- Lưu trữ dữ liệu tích hợp: Lưu trữ dữ liệu đã tích hợp vào một kho dữ liệu *(data warehouse)*, hồ dữ liệu *(data lake)*, hoặc một cơ sở dữ liệu tập trung để dễ dàng truy cập.

Các phương pháp tích hợp dữ liệu

- ETL *(Extract, Transform, Load)*: Dữ liệu được trích xuất từ nhiều nguồn, chuyển đổi thành định dạng thống nhất và tải vào một kho dữ liệu hoặc cơ sở dữ liệu chung.
- ELT *(Extract, Load, Transform)*: Dữ liệu được trích xuất và tải vào nơi lưu trữ trước, sau đó mới thực hiện các chuyển đổi. Phương pháp này phù hợp với kho dữ liệu đám mây có khả năng xử lý lớn.
- Tích hợp dữ liệu ảo *(Data Virtualization)*: Cho phép người dùng truy cập và truy vấn dữ liệu từ nhiều nguồn mà không cần phải hợp nhất vật lý dữ liệu vào một nơi duy nhất. Dữ liệu ảo hóa giúp truy cập theo thời gian thực nhưng đòi hỏi công nghệ phù hợp để xử lý.
- Tích hợp ngang hàng *(Data Federation)*: Cho phép người dùng truy cập và truy vấn dữ liệu từ các nguồn khác nhau thông qua một giao diện chung. Không chuyển đổi dữ liệu vào cùng một kho mà thực hiện truy vấn tại nguồn và trả về kết quả.

Thách thức trong tích hợp dữ liệu

- Định dạng và cấu trúc khác nhau: Dữ liệu từ các nguồn khác nhau thường có định dạng khác nhau, gây khó khăn khi hợp nhất.
- Vấn đề về trùng lặp và dữ liệu không nhất quán: Dữ liệu có thể bị trùng lặp hoặc không khớp, dẫn đến sai lệch trong phân tích.
- Tính bảo mật và quyền riêng tư: Việc chia sẻ và tích hợp dữ liệu có thể gây rủi ro cho quyền riêng tư và bảo mật, đòi hỏi phải có các biện pháp bảo mật phù hợp.
- Khả năng mở rộng: Cần phải thiết kế hệ thống tích hợp sao cho có thể mở rộng dễ dàng khi có thêm các nguồn dữ liệu mới.

4.6. CHUẨN HÓA DỮ LIỆU

Chuẩn hóa dữ liệu *(Data Normalization)* là quá trình chuyển đổi dữ liệu thành một định dạng nhất quán và chuẩn mực, thường là để chuẩn bị dữ liệu cho các phân tích, mô hình học máy, hoặc quản lý cơ sở dữ liệu. Mục tiêu của chuẩn hóa là để đảm bảo dữ liệu không bị trùng lặp, loại bỏ lỗi và sắp xếp thông tin để sử dụng dễ dàng, từ đó tăng tính chính xác và hiệu quả của các phân tích hoặc mô hình.

Tại sao cần chuẩn hóa dữ liệu?

Chuẩn hóa dữ liệu giúp:

- Giảm thiểu độ lệch dữ liệu: Các giá trị từ các nguồn khác nhau được đưa về cùng một thang đo hoặc chuẩn, tránh các vấn đề do đơn vị hoặc phạm vi khác nhau.
- Cải thiện hiệu quả và hiệu suất của mô hình: Các mô hình học máy hoạt động tốt hơn khi các đặc trưng có cùng thang đo.
- Dễ dàng so sánh và phân tích dữ liệu: Các giá trị chuẩn hóa sẽ giúp quá trình phân tích trở nên đơn giản và trực quan hơn.
- Đảm bảo tính nhất quán trong toàn bộ dữ liệu, đặc biệt khi dữ liệu đến từ nhiều nguồn khác nhau.

4.6.1. Chuẩn hóa Min-Max

Chuẩn hóa Min-Max là một phương pháp chuyển đổi dữ liệu sao cho các giá trị của dữ liệu được thu hẹp về một khoảng xác định, thường là từ 0 đến 1, hoặc từ -1 đến 1. Đây là một kỹ thuật phổ biến trong tiền xử lý dữ liệu khi làm việc với các thuật toán học máy và các phân tích thống kê.

Với phương pháp Min-Max, giá trị mới $X_{normalized}$ của một giá trị dữ liệu Xđược tính theo công thức:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} * (newmax - newmin) + newmin$$

Trong đó:

X : là giá trị ban đầu của dữ liệu

X_{min}, X_{max} : lần lượt là giá trị nhỏ nhất và giá trị lớn nhất của dữ liệu trong 1 đặc trưng (feature) cụ thể.

Lợi ích và hạn chế của chuẩn hóa Min-Max

- Lợi ích: Đơn giản, dễ hiểu, và có thể giữ được mối quan hệ tỷ lệ giữa các giá trị ban đầu.
- Hạn chế: Phụ thuộc vào giá trị min và max, nên có thể bị ảnh hưởng bởi các giá trị ngoại lệ *(outliers)* và cần tính lại nếu dữ liệu mới có giá trị min hoặc max thay đổi.

4.6.2. Chuẩn hóa Z-score

Chuẩn hóa Z-score *(hay Standardization)* là một kỹ thuật chuẩn hóa dữ liệu bằng cách chuyển đổi dữ liệu về một phân phối có giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1. Phương pháp này giúp làm cho dữ liệu có dạng chuẩn và loại bỏ các đơn vị đo khác nhau, giúp việc phân tích và sử dụng các mô hình học máy chính xác và hiệu quả hơn.

Với chuẩn hóa Z-score, giá trị chuẩn hóa Z của một giá trị dữ liệu X được tính theo công thức:

$$Z = \frac{X - \mu}{\sigma}$$

Trong đó:

X : là giá trị ban đầu của dữ liệu

μ : giá trị trung bình của đặc trưng *(feature)* đang cần chuẩn hóa

σ : độ lệch chuẩn của đặc trưng đó

Kết quả của phép chuẩn hóa Z-score sẽ tạo ra các giá trị có phân phối chuẩn *(normal distribution)* với trung bình là 0 và độ lệch chuẩn là 1. Các giá trị Z-score sẽ nằm quanh khoảng từ -3 đến 3 trong hầu hết các trường hợp *(tùy thuộc vào phân phối của dữ liệu gốc)*.

Khi nào nên sử dụng chuẩn hóa Z-score?

- Dữ liệu có phân phối chuẩn hoặc gần chuẩn: Khi dữ liệu gốc đã có phân phối gần với phân phối chuẩn *(normal distribution)*, Z-score có thể là lựa chọn tối ưu.
- Dữ liệu chứa nhiều đặc trưng có phạm vi giá trị khác nhau: Chuẩn hóa Z-score đưa tất cả các đặc trưng về cùng một thang đo *(trung bình 0, độ lệch chuẩn 1)*, giúp cân bằng ảnh hưởng của chúng trong mô hình.
- Các thuật toán yêu cầu dữ liệu chuẩn hóa: Một số thuật toán như Hồi quy tuyến tính, Hồi quy logistic, K-means clustering, và Principal Component Analysis *(PCA)* yêu cầu hoặc hoạt động hiệu quả hơn khi dữ liệu có dạng chuẩn hóa.

Lợi ích và hạn chế của chuẩn hóa Z-score

Lợi ích:

- Đưa tất cả các đặc trưng về cùng một thang đo mà không thay đổi sự phân phối tương đối của dữ liệu.
- Hữu ích trong các bài toán phân tích dữ liệu thống kê hoặc các thuật toán học máy yêu cầu dữ liệu có dạng chuẩn.

Hạn chế:

- Nhạy cảm với ngoại lệ (outliers), vì các giá trị ngoại lệ có thể làm thay đổi trung bình và độ lệch chuẩn, ảnh hưởng đến kết quả chuẩn hóa.
- Có thể khó áp dụng cho dữ liệu có phân phối quá lệch, vì Z-score chỉ thực sự hữu ích khi dữ liệu gần với phân phối chuẩn.

4.6.3. Chuẩn hóa theo tỉ lệ

Chuẩn hóa theo tỷ lệ (Ratio Scaling) là một phương pháp chuẩn hóa dữ liệu, trong đó các giá trị của một đặc trưng được chia cho một giá trị cơ sở hoặc tỷ lệ chuẩn để đưa chúng về một phạm vi cụ thể hoặc thang đo chung. Khác với các phương pháp chuẩn hóa khác như Min-Max hay Z-score, chuẩn hóa theo tỷ lệ chủ yếu tập trung vào việc thay đổi tỷ lệ giữa các giá trị dữ liệu, đồng thời vẫn giữ lại mối quan hệ tỷ lệ giữa các điểm dữ liệu.

Các loại chuẩn hóa theo tỷ lệ

(1) Chuẩn hóa bằng tỷ lệ tối đa (Max Ratio Scaling): Trong phương pháp này, giá trị của một đặc trưng được chia cho giá trị tối đa của đặc trưng đó trong dữ liệu. Đây là một dạng chuẩn hóa theo tỷ lệ với mục đích đưa dữ liệu về phạm vi từ 0 đến 1.

$$X_{scaled} = \frac{X}{X_{max}}$$

Trong đó:

X : là giá trị gốc của dữ liệu

X_{max} : là giá trị tối đa trong đặc trưng đó

(2) Chuẩn hóa theo tỉ lệ trung bình: Dữ liệu được chia cho giá trị trung bình của đặc trưng. Phương pháp này giúp đưa các giá trị dữ liệu về một tỷ lệ so với trung bình.

$$X_{scaled} = \frac{X}{X_{\mu}}$$

(3) Chuẩn hóa theo tỉ lệ với tổng cộng: Dữ liệu có thể được chia cho tổng của tất cả các giá trị trong một đặc trưng. Phương pháp này thường được áp dụng khi muốn các giá trị của dữ liệu trở thành tỷ lệ phần trăm của tổng các giá trị.

$$X_{scaled} = \frac{X}{\sum_{x=1}^{n} X_i}$$

Khi nào nên sử dụng chuẩn hóa theo tỷ lệ?

- Dữ liệu có các giá trị thuộc cùng một thang đo nhưng có tỷ lệ chênh lệch: Chuẩn hóa theo tỷ lệ giúp điều chỉnh các đặc trưng có tỷ lệ khác nhau nhưng vẫn duy trì mối quan hệ giữa các đặc trưng.
- Dữ liệu không có đơn vị đo đặc biệt: Phương pháp này thường áp dụng cho các dữ liệu không có đơn vị đo cố định, hoặc khi các đặc trưng có tỷ lệ hợp lý mà không bị ảnh hưởng bởi giá trị tối đa hoặc trung bình.
- Khi dữ liệu có giá trị ngoại lệ *(outliers)*: Tương tự như chuẩn hóa Min-Max, chuẩn hóa theo tỷ lệ có thể bị ảnh hưởng bởi giá trị ngoại lệ, nhưng các ngoại lệ sẽ ít ảnh hưởng nếu chuẩn hóa được thực hiện với tỷ lệ trung bình hoặc tổng.

Lợi ích và hạn chế của chuẩn hóa theo tỷ lệ

Lợi ích:

- Giữ lại tỷ lệ tương đối giữa các giá trị dữ liệu, làm cho chúng dễ dàng so sánh với nhau.
- Đơn giản và dễ hiểu.
- Thích hợp khi các giá trị có đơn vị đo giống nhau và cần điều chỉnh theo một chuẩn hoặc tỷ lệ chung.

Hạn chế:

- Dễ bị ảnh hưởng bởi các giá trị ngoại lệ (outliers), đặc biệt nếu tỷ lệ chuẩn hóa được tính dựa trên giá trị tối đa hoặc tổng.

- Không thay đổi mối quan hệ giữa các dữ liệu khi có sự chênh lệch lớn trong tỷ lệ giữa các đặc trưng.

4.7. RỜI RẠC HÓA DỮ LIỆU

Rời rạc hóa dữ liệu *(Discretization)* là quá trình chuyển đổi các giá trị liên tục *(continuous)* thành các giá trị rời rạc *(discrete)*. Mục tiêu của quá trình này là chia các giá trị liên tục thành các khoảng hoặc lớp *(bins)*, giúp các mô hình học máy xử lý dễ dàng hơn, đặc biệt khi làm việc với dữ liệu phân loại hoặc khi các thuật toán yêu cầu đầu vào rời rạc.

Mục đích của rời rạc hóa

- Giảm độ phức tạp của dữ liệu: Rời rạc hóa giúp giảm số lượng các giá trị độc đáo trong dữ liệu, làm cho dữ liệu trở nên dễ dàng xử lý hơn, đặc biệt khi dữ liệu có quá nhiều giá trị.
- Chuyển đổi dữ liệu liên tục thành dữ liệu phân loại: Dữ liệu liên tục có thể khó sử dụng trong một số thuật toán học máy *(như cây quyết định)*. Rời rạc hóa giúp chuyển đổi nó thành dạng phân loại, phù hợp với những mô hình yêu cầu đầu vào phân loại.
- Giúp cải thiện hiệu suất mô hình: Trong một số trường hợp, rời rạc hóa có thể giúp cải thiện hiệu suất của mô hình bằng cách giảm sự phức tạp và làm giảm hiện tượng quá khớp *(overfitting)*.

Các phương pháp rời rạc hóa

(1) Rời rạc hóa theo ngưỡng cố định *(Equal-width binning)*:

- Dữ liệu được chia thành các khoảng có độ rộng bằng nhau.
- Ví dụ: Chia giá trị từ 0 đến 100 thành 5 khoảng, mỗi khoảng có độ rộng là 20 (0-20, 21-40, ...).

(2) Rời rạc hóa theo số lượng phần tử (Equal-frequency binning):

- Dữ liệu được chia thành các khoảng sao cho mỗi khoảng có số lượng phần tử (dữ liệu) tương đối đều nhau.
- Ví dụ: Nếu có 1000 giá trị và chia thành 4 khoảng, mỗi khoảng sẽ chứa 250 giá trị.

(3) Rời rạc hóa theo phương pháp k-means clustering:

- Dữ liệu được chia thành các nhóm *(clusters)* bằng cách sử dụng thuật toán k-means, rồi gán nhãn cho các điểm dữ liệu theo nhóm đó.
- Phương pháp này dựa vào kỹ thuật phân nhóm dữ liệu tự nhiên.

(4) Rời rạc hóa bằng thuật toán tối ưu *(Optimal binning)*: Phương pháp này tìm cách tối ưu hóa các khoảng sao cho chúng tốt nhất để phân loại các đối tượng. Có thể sử dụng các thuật toán như phân tích thông tin để tìm cách phân chia dữ liệu sao cho chúng dễ phân biệt nhất.

(5) Rời rạc hóa thủ công: Dữ liệu có thể được chia thành các khoảng hoặc lớp theo cách thủ công, dựa vào sự hiểu biết của người phân tích về dữ liệu và mục tiêu phân tích.

Khi nào sử dụng rời rạc hóa?

- Khi dữ liệu có độ phân giải cao hoặc có nhiều giá trị liên tục: Rời rạc hóa giúp giảm sự phức tạp của dữ liệu và làm cho việc xử lý trở nên dễ dàng hơn.
- Khi mô hình yêu cầu dữ liệu phân loại: Các thuật toán như cây quyết định hoặc Naive Bayes đôi khi yêu cầu dữ liệu đầu vào phải là dạng phân loại, thay vì dữ liệu liên tục.
- Khi cần tối ưu hóa các mô hình học máy: Việc rời rạc hóa có thể giúp giảm bớt các vấn đề quá khớp (overfitting) trong các mô hình học máy.

Lợi ích của rời rạc hóa

- Giảm độ phức tạp: Chuyển dữ liệu liên tục thành dữ liệu rời rạc giúp giảm số lượng các giá trị cần phải xử lý.
- Cải thiện tốc độ xử lý: Việc sử dụng các giá trị rời rạc có thể làm cho các thuật toán học máy hoạt động nhanh hơn, nhất là với các thuật toán nhạy cảm với dữ liệu liên tục như KNN hay SVM.
- Giảm hiện tượng overfitting: Khi dữ liệu có quá nhiều chi tiết, các mô hình học máy có thể bị overfit. Rời rạc hóa giúp làm giảm chi tiết và chỉ tập trung vào các mẫu chính, giúp mô hình tổng quát hơn.

Hạn chế của rời rạc hóa

- Mất thông tin: Việc rời rạc hóa có thể làm mất một số thông tin quan trọng từ dữ liệu gốc, vì các giá trị gần nhau có thể bị gộp chung vào cùng một nhóm.

– Khó khăn trong việc xác định số nhóm (bins): Việc chọn số nhóm (bins) thích hợp có thể khó khăn, và việc chọn sai có thể ảnh hưởng xấu đến hiệu suất mô hình.

4.7. THỰC HÀNH CHUẨN HÓA DỮ LIỆU THEO MIN-MAX NORMALIZATION

4.7.1. Dataset

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	Target
b	30.83	0	u	g	w	v	1.25	t	t	1	f	g	202	0	+
a	58.67	4.46	u	g	q	h	3.04	t	t	6	f	g	43	560	+
a	24.5	0.5	u	g	q	h	1.5	t	f	0	f	g	280	824	+
b	27.83	1.54	u	g	w	v	3.75	t	t	5	t	g	100	3	+
b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120	0	+
b	32.08	4	u	g	m	v	2.5	t	f	0	t	g	360	0	+
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	280	750	-
b	35	3.375	u	g	c	h	8.29	f	f	0	t	g	0	0	-

Download full dataset tại: https://github.com/NT02IT/BasicMachineLearning/blob/main/datasets/min-max-normalization/min-max-normalization.csv

Dataset Japanese Credit Screening được thu thập và cung cấp bởi Chiharu Sano vào năm 1992, có sẵn trên UCI Machine Learning Repository. Dữ liệu này là một bài toán phân loại, được sử dụng để dự đoán khả năng thanh toán của khách hàng trong lĩnh vực tín dụng tại Nhật Bản. Dữ liệu chứa thông tin liên quan đến các đặc trưng của khách hàng, bao gồm các yếu tố như độ tuổi, nghề nghiệp, và thông tin tín dụng.

Mô tả các đặc trưng của dataset

A1 đến A15: Là các đặc trưng mô tả các yếu tố liên quan đến khách hàng, bao gồm các giá trị phân loại và số liệu. Các giá trị trong các cột này có thể là chữ cái hoặc số và biểu thị thông tin về độ tuổi, loại nghề nghiệp, mức độ tín dụng, v.v.

Target (Cột cuối cùng): Biến mục tiêu (target) được sử dụng để phân loại khách hàng thành hai nhóm:

- "+": Đại diện cho khách hàng có khả năng thanh toán tín dụng tốt.
- "-": Đại diện cho khách hàng có khả năng thanh toán tín dụng kém.

Số lượng mẫu *(instances)*: Dataset chứa dữ liệu của 690 khách hàng.

Số lượng đặc trưng *(features)*: Có 15 đặc trưng *(A1 đến A15)* mô tả thông tin khách hàng.

Dạng dữ liệu: Các đặc trưng có sự kết hợp giữa các giá trị số và phân loại, giúp phản ánh đặc điểm tín dụng đa dạng của khách hàng.

4.7.2. Chuẩn hóa dữ liệu bằng Excel

Dữ liệu đầu vào:

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	Target
b	30.83	0	u	g	w	v	1.25	t	t	1	f	g	202	0	+
a	58.67	4.46	u	g	q	h	3.04	t	t	6	f	g	43	560	+
a	24.5	0.5	u	g	q	h	1.5	t	f	0	f	g	280	824	+
b	27.83	1.54	u	g	w	v	3.75	t	t	5	t	g	100	3	+
b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120	0	+
b	32.08	4	u	g	m	v	2.5	t	f	0	t	g	360	0	+
b	33.17	1.04	u	g	r	h	6.5	t	f	0	t	g	164	31285	+
a	22.92	11.585	u	g	cc	v	0.04	t	f	0	f	g	80	1349	+
b	54.42	0.5	y	p	k	h	3.96	t	f	0	f	g	180	314	+
b	42.5	4.915	y	p	w	v	3.165	t	f	0	t	g	52	1442	+

Đầu tiên ta cần tính phương sai của thuộc tính có dữ liệu là số tại dòng Variance với công thức

$$= IFERROR(VAR.P(X), "")$$

Trong đó: X là vùng dữ liệu cần tính phương sai

Dòng Blank Cells gồm các ô chứa số lượng ô trống dữ liệu

Dòng Missing Data gồm các ô chứa số lượng ô có dữ liệu là ?

23.61391			30164.17	27105828		**Variance**
0	0	0	0	0	0	**Blank Cells**
0	0	0	13	0	0	**Missing Data**

Sau đó, ta sẽ tạo 1 trang mới để chuẩn hóa dữ liệu với phương pháp bình thường hóa dữ liệu có công thức: y = (x - min) / (max - min)

Trong đó:

- y là biến sau khi bình thường hóa dữ liệu
- x là biến trước khi bình thường hóa dữ liệu
- max, min là biến lớn nhất, nhỏ nhất trong dữ liệu đầu vào

Công thức excel:

$$= IFERROR((crx!X - MIN(crx!X1))/(MAX(crx!X1) - MIN(crx!X1)), crx!X)$$

Trong đó:

- crx: trang tính
- X: biến dữ liệu đầu vào trước khi được bình thường hóa
- X1: vùng dữ liệu đầu vào

Hàm IFERROR được sử dụng để phòng tránh trường hợp sai kiểu dữ liệu. Nếu có sai kiểu thì sẽ trả lại biến đầu vào

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	Target
b	25.68421053	0	u	g	w	v	4.385964912	t	t	1.492537313	f	g	10.1	0	+
a	67.54887218	15.92857143	u	g	q	h	10.66666667	t	t	8.955223881	f	g	2.15	0.56	+
a	16.16541353	1.785714286	u	g	q	h	5.263157895	t	f	0	f	g	14	0.824	+
b	21.17293233	5.5	u	g	w	v	13.15789474	t	t	7.462686567	t	g	5	0.003	+
b	9.654135338	20.08928571	u	g	w	v	6	t	f	0	f	s	6	0	+
b	27.56390977	14.28571429	u	g	m	v	8.771929825	t	f	0	t	g	18	0	+

Sau cùng ta tính phương sai của dữ liệu sau khi bình thường hóa:

23.61391			30164.17	27105828		**Variance**

4.7.3. Chuẩn hóa dữ liệu bằng Python

Bước 1: Tạo class *Normalization.py* với các phương thức static *minMaxNormalizationWlib(), minMaxNormalizationNolib(), encode_dataframe(), decode_dataframe().*

Bước 2: Trong phương thức *minMaxNormalizationWlib()* viết code để chuẩn hóa dataframe nhập vào và chuẩn hóa các cột là số về khoảng mới sử dụng MinMaxScaler thuộc thư viện Sklearn.

```
@staticmethod
def minMaxNormalizationWlib(df, new_min, new_max):
    df_normalized = df.copy()
    scaler = MinMaxScaler(feature_range=(new_min, new_max))
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:
            df_normalized[col] = scaler.fit_transform(df[[col]])
    return df_normalized
```

Bước 3: Trong phương thức *minMaxNormalizationNolib()* viết code để chuẩn hóa dataframe nhập vào và chuẩn hóa các cột là số về khoảng mới sử dụng MinMaxScaler thuộc thư viện Sklearn.

```
  @staticmethod
  def minMaxNormalizationNolib(df, new_min, new_max):
    df_normalized = df.copy()

    for col in df_normalized.columns:
      if df_normalized[col].dtype in ['int64', 'float64']:
        min_val = df_normalized[col].min()
        max_val = df_normalized[col].max()

        if max_val - min_val == 0:
          df_normalized[col] = new_min
        else:
          df_normalized[col] = (df_normalized[col] - min_val) / (max_val - min_val) * (new_max -
new_min) + new_min

    return df_normalized
```

Bước 4: Trong phương thức *encode_dataframe()* viết code để đổi các cột là chữ trong dataframe được nhập vào thành số

```
  @staticmethod
  def encode_dataframe(df):
    label_encoders = {} # Từ điển lưu trữ LabelEncoder cho mỗi cột

    for col in df.columns:
      if df[col].dtype == 'object':  # Kiểm tra nếu cột là kiểu chuỗi
        encoder = LabelEncoder()
        df[col] = encoder.fit_transform(df[col])  # Mã hóa cột chuỗi thành số
        label_encoders[col] = encoder  # Lưu trữ encoder cho cột vào từ điển

    return df, label_encoders
```

Bước 5: Trong phương thức *decode_dataframe()* viết code để trả lại trạng thái của 1 dataframe về trạng thái trước khi được số hóa bằng cách truyền vào dataframe đó cùng với map các nhãn ban đầu.

```
  @staticmethod
  def decode_dataframe(df, label_encoders):
    for col in df.columns:
      if col in label_encoders:  # Kiểm tra nếu cột đã có LabelEncoder
        encoder = label_encoders[col]
```

```
        df[col] = encoder.inverse_transform(df[col])  # Giải mã các giá trị số thành chuỗi
    return df
```

Download source code tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/Normalization/Normalization.py

CHƯƠNG V: NEURAL NETWORK

5.1. TỔNG QUAN VỀ NEURAL NETWORK

5.1.1. Neural Network là gì?

Con chó có thể phân biệt được người thân trong gia đình và người lạ, hay đứa trẻ có thể phân biệt được các con vật. Những việc tưởng chừng như rất đơn giản nhưng lại cực kì khó để thực hiện bằng máy tính. Vậy sự khác biệt nằm ở đâu? Câu trả lời nằm ở cấu trúc bộ não với lượng lớn các neuron thần kinh liên kết với nhau. Liệu máy tính có thể mô phỏng lại cấu trúc bộ não để giải các bài toán trên ???

Neural là tính từ của neuron (nơ-ron), network chỉ cấu trúc, cách các neuron đó liên kết với nhau, nên Neural Network (NN) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

Neural network là một loại trí tuệ nhân tạo cố gắng mô phỏng cách hoạt động của não người. Thay vì sử dụng mô hình số, trong đó tất cả các phép tính chỉ xử lý các số 0 và 1, Neural Network hoạt động bằng cách tạo ra các kết nối giữa các yếu tố xử lý, tương đương với các neuron trong máy tính. Cấu trúc và trọng số của các kết nối quyết định đầu ra.

Neural Network đặc biệt hiệu quả trong việc dự đoán các sự kiện khi chúng có một cơ sở dữ liệu lớn chứa các ví dụ trước đó để tham khảo. Nói chính xác, một Neural Network ngụ ý một máy tính không kỹ thuật số, nhưng các Neural Network có thể được mô phỏng trên các máy tính kỹ thuật số.

Neural Network là một tập hợp các thuật toán, mô phỏng sơ bộ theo não người, được thiết kế để nhận diện các mẫu. Chúng diễn giải dữ liệu cảm giác thông qua một dạng nhận thức máy, gán nhãn hoặc phân nhóm các dữ liệu thô. Các mẫu mà chúng nhận diện là các giá trị số, chứa trong các vector, mà tất cả dữ liệu thực tế, dù là hình ảnh, âm thanh, văn bản hay chuỗi thời gian, đều phải được chuyển đổi thành.

5.1.2. Mạng thần kinh nhân tạo (ANN - Artificial Neutral Networks)

Mô hình tính toán dựa trên cấu trúc và chức năng của các mạng thần kinh sinh học được gọi là mạng thần kinh nhân tạo (ANN). Bởi vì một mạng neuron thay đổi - hay "học" theo một cách nào đó - dựa trên đầu vào và đầu ra, thông tin đi qua mạng sẽ thay đổi cấu trúc của ANN.

ANNs là công cụ mô hình hóa dữ liệu phi tuyến tính, được sử dụng để mô tả các tương tác phức tạp giữa đầu vào và đầu ra hoặc để khám phá các mẫu. Thiết kế mạng neuron là một tên gọi khác của ANN. Với khả năng xuất sắc trong việc suy luận ý nghĩa từ dữ liệu phức tạp hoặc không chính xác, mạng neuron có thể được sử dụng để khám phá các mẫu và xu hướng mà con người hoặc các hệ thống máy tính khác không thể phát hiện. Một mạng neuron đã qua đào tạo có thể được coi là một "chuyên gia" trong lĩnh vực thông tin mà nó đã được huấn luyện để đánh giá.

5.1.3. Hoạt động của các neuron

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là thành phần quan trọng nhất của não. Đầu chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron lại liên kết với tầm 10.000 nơ-ron khác.

Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác. ⇒ Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không. Tương tự các hoạt động của hàm sigmoid.

Tuy nhiên NN chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình đấy đi giải quyết các bài toán chúng ta cần.

5.2. MÔ HÌNH NEURAL NETWORK

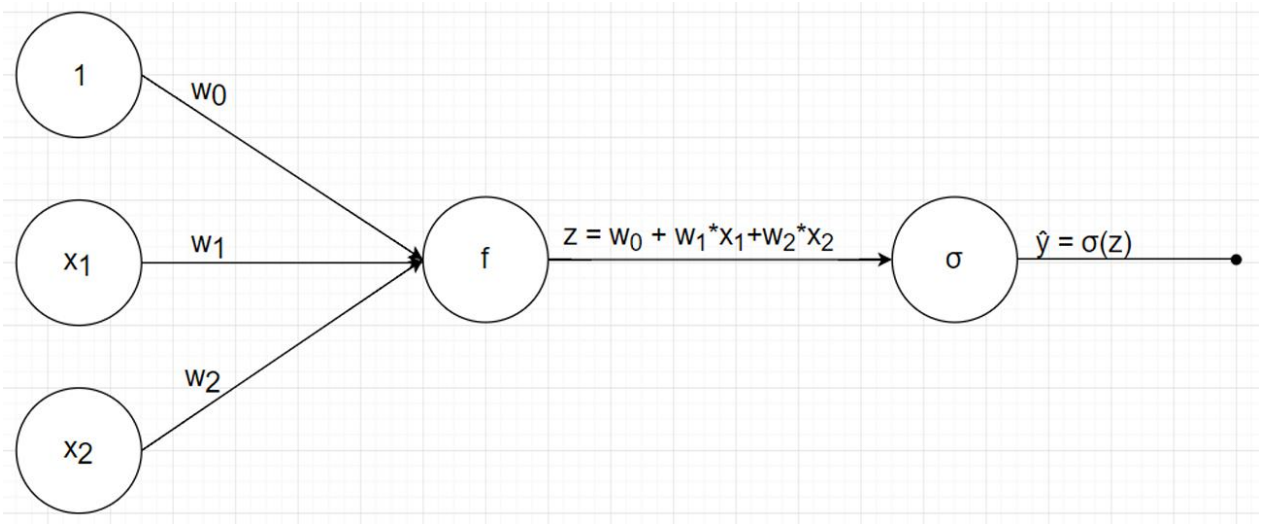
5.2.1. Logistic Regression

Logistic regression là một mô hình neural network đơn giản.

Logistic model: $\widehat{y_i} = \sigma(w_0 + w_1 * x_1 + w_2 * x_1)$

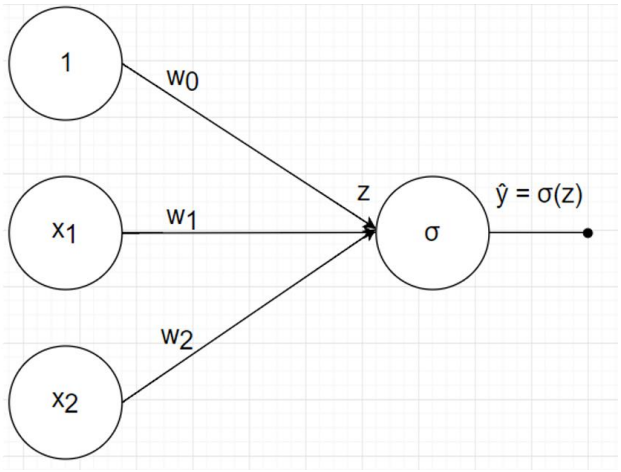
Có 2 bước chính để thực hiện:

- Tính tổng linear: $z = w_0 + w_1 * x_1 + w_2 * x_1$
- Áp dụng sigmoid function: $\widehat{y_i} = \sigma(z)$



Ảnh 9: Mô hình Neural Network đơn giản với Logistic Regression (1)

Để biểu diễn gọn lại ta sẽ gộp hai bước trên thành một trên biểu đồ như sau:



Ảnh 10: Mô hình Neutral Network đơn giản với Logistic Regression (2)

Hệ số w_0 được gọi là bias. Để ý từ những bài trước đến giờ dữ liệu khi tính toán luôn được thêm 1 để tính hệ số bias w_0 . Tại sao lại cần hệ số bias? Quay lại với bài Linear Regression, phương trình đường thẳng sẽ thế nào nếu bỏ w_0, phương trình giờ có dạng: $y = w_1 x$, sẽ luôn đi qua gốc tọa độ và nó không tổng quát hóa phương trình đường thẳng nên có thể không tìm được phương trình mong muốn. ⇒ Việc thêm bias *(hệ số tự do)* là rất quan trọng.

Hàm sigmoid ở đây được gọi là **activation function**.

5.2.2. Mô hình tổng quát

Có ba lớp *(layer)* trong kiến trúc Mạng Neuron Nhân tạo:

- **Input layer:** Lớp đầu vào của mạng neuron là tập hợp các neuron đầu vào nhân tạo. Chúng truyền thông tin từ các lớp neuron ban đầu vào hệ thống để xử lý. Lớp đầu vào của mạng neuron bắt đầu quy trình làm việc.
- **Hidden layer:** Lớp ẩn trong mạng neuron nhân tạo bao gồm các lớp đầu vào và đầu ra, và đầu vào cũng như đầu ra của các neuron nhân tạo được xác định trọng số dựa trên số lượng đầu vào.
- **Output layer:** Lớp neuron cuối cùng trong mạng neuron nhân tạo là lớp đầu ra, cung cấp các đầu ra cụ thể trong chương trình. Vì là các nút "thực hiện" cuối cùng của mạng, các neuron trong lớp đầu ra có thể được xây dựng và xử lý theo cách khác nhau.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer-1 *(không tính input layer).* Các hình tròn được gọi là node.

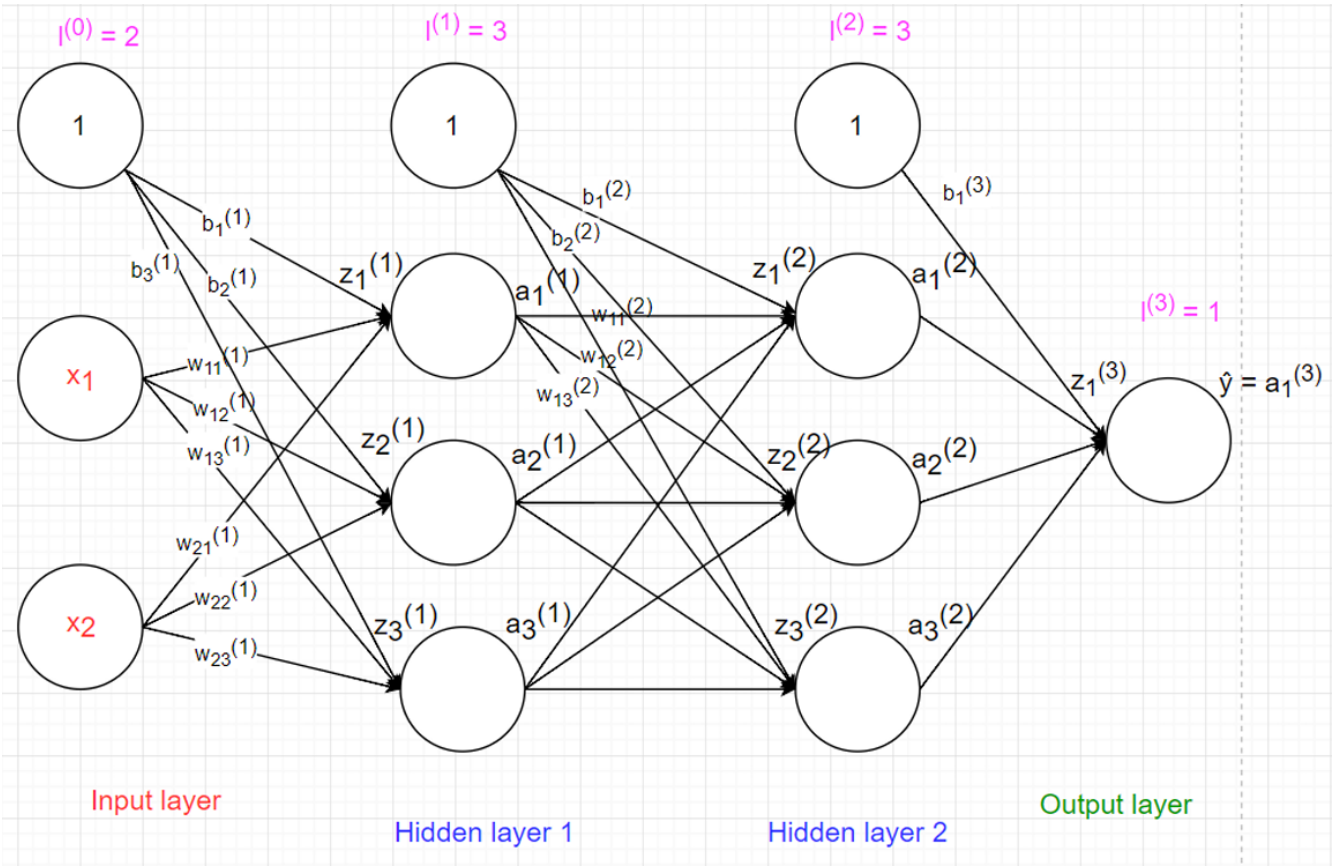
Ví dụ như ở hình trên có 1 input layer, 2 hidden layer và 1 output layer. Số lượng layer của mô hình là 3 layer.

Mỗi node trong hidden layer và output layer :

- Liên kết với tất cả các node ở layer trước đó với các hệ số *w* riêng.
- Mỗi node có 1 hệ số bias b riêng.
- Diễn ra 2 bước: tính tổng linear và áp dụng activation function.

Biểu diễn toán học:

- Số node trong hidden layer thứ i là $l^{(i)}$.
- Ma trận $W^{(k)}$ kích thước $l^{(k-1)} * l^{(k)}$, là ma trận hệ số giữa layer *(k-1)* và layer *k*, trong đó $W_{ij}^{(k)}$ là hệ số kết nối từ node thứ *i* của layer *k-1* đến node thứ *j* của layer *k*.
- Vector $b^{(k)}$ kích thước $l^{(k)} * 1$, là hệ số bias của các node trong layer *k*, trong đó $b_i^{(k)}$ là bias thứ *i* trong layer *k*.
- Với node thứ i trong layer *l* có bias $b_i^{(k)}$ thực hiện 2 bước:
    - Tính tổng linear: $z_i^{(l)} = \sum_{j=1}^{l^{(l-1)}} (a_j^{(l-1)} * w_{ji}^{(l)} + b_i^{(l)})$ là tổng tất cả các node trong layer trước nhân với hệ số *w* tương ứng, rồi cộng với bias *b*.
    - Áp dụng activation function: $a_i^{(l)} = \sigma(z_i^{(l)})$
- Vector $z^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị các node trong layer *k* sau bước tính tổng linear.
- Vector $a^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị các node trong layer *k* sau khi áp dụng activation function.



Ảnh 11: Ví dụ về mô hình Neural Network

Môhình neural network trên gồm 3 layer. Input layer có 2 node ($l^{(0)} = 2$), hidden layer 1 có 3 node, hidden layer 2 có 3 node và output layer có 1 node.

Do mỗi node trong hidden layer và output layer đều có bias nên trong input layer và hidden layer cần thêm node 1 để tính bias (nhưng không tính vào tổng số node layer có).

Tại node thứ 2 ở layer 1, ta có:

- $z_2^{(1)} = x_1 * w_{12}^{(1)} + x_2 w_{22}^{(1)} + b_2^{(1)}$
- $a_2^{(1)} = \sigma(z_2^{(1)})$

Hay ở node thứ 3 layer 2, ta có:

- $z_3^{(2)} = a_1 * w_{13}^{(2)} + a_2 w_{23}^{(2)} + a_3 w_{33}^{(2)} + b_3^{(2)}$
- $a_3^{(2)} = \sigma(z_3^{(2)})$

5.3. LAN TRUYỀN XUÔI (FORWARD PROPAGATION)

Lan truyền xuôi là quá trình đi từ lớp đầu vào qua các lớp ẩn đến lớp đầu ra của mạng, để tính toán kết quả dự đoán.

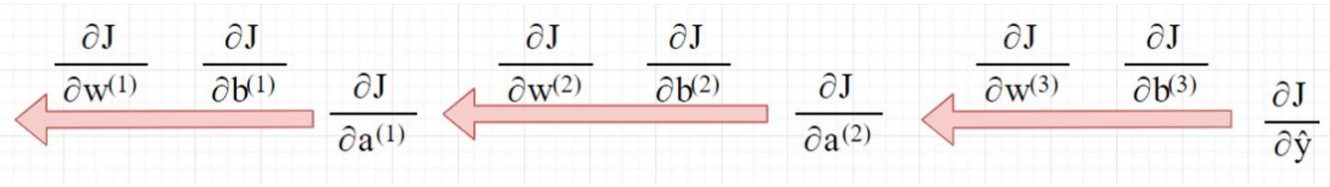
Ảnh 12: Quá trình lan truyền xuôi

Quy trình lan truyền xuôi:

- Bước 1: Giá trị đầu vào X được đưa vào mạng tại lớp đầu vào.
- Bước 2: Ở mỗi lớp, tính toán các đầu ra của neuron dựa trên trọng số W và bias b của chúng. Với mỗi lớp, công thức là: $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$
- Bước 3: Áp dụng hàm kích hoạt (activation function) như ReLU, sigmoid, hay tanh để tạo đầu ra (output) cho lớp hiện tại: $a^{(l)} = activation(z^{(l)})$
- Bước 4: Lặp lại các bước trên cho đến khi đạt đến lớp cuối (lớp đầu ra), từ đó lấy ra đầu ra cuối cùng của mạng $\hat{y}$, là dự đoán của mạng.

5.4. LAN TRUYỀN NGƯỢC (BACKWARD PROPAGATION)

Lan truyền ngược là quá trình tính toán đạo hàm của hàm mất mát theo các tham số của mạng, nhằm điều chỉnh trọng số và bias để giảm lỗi dự đoán.



Ảnh 13: Quá trình lan truyền ngược

Quy trình lan truyền ngược:

- Bước 1: Tính toán hàm mất mát $L(\hat{y}, y)$ (ví dụ, Mean Squared Error cho hồi quy, Cross-Entropy cho phân loại) giữa đầu ra dự đoán $\hat{y}$ và giá trị thực y.

- Bước 2: Bắt đầu từ lớp cuối cùng, tính đạo hàm của hàm mất mát theo đầu ra của lớp, rồi theo trọng số và bias của lớp đó. Dựa vào quy tắc chuỗi trong tính đạo hàm, quá trình này tính toán từ lớp cuối đến lớp đầu.
- Bước 3: Lặp lại việc tính đạo hàm và cập nhật tham số cho tất cả các lớp theo thứ tự ngược từ lớp đầu ra về lớp đầu vào.

5.5. LUẬT HỌC PERCEPTRON

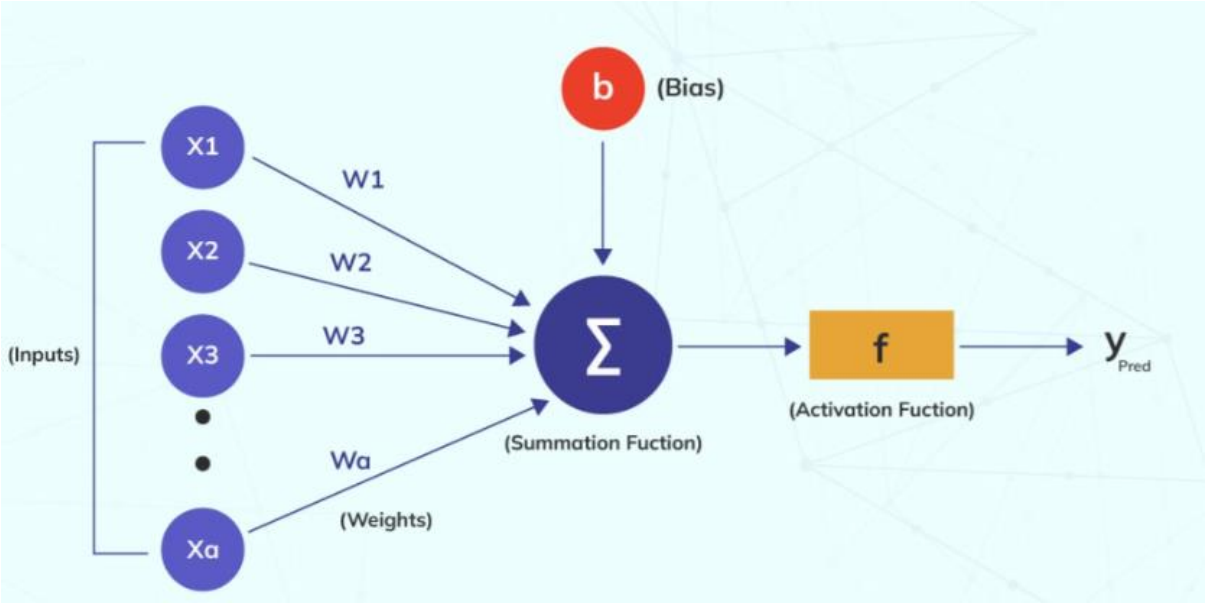
5.5.1. Giới thiệu

Perceptron được Frank Rosenblatt giới thiệu vào năm 1957. Ông đã đề xuất một quy tắc học Perceptron dựa trên neuron MCP ban đầu. Perceptron là một thuật toán học có giám sát các bộ phân loại nhị phân. Thuật toán này cho phép các neuron học và xử lý từng phần tử trong tập huấn luyện cùng một lúc.

Một thuật toán dựa trên máy được sử dụng để học có giám sát các tác vụ sắp xếp nhị phân khác nhau được gọi là Perceptron. Hơn nữa, Perceptron còn có vai trò thiết yếu như một neuron nhân tạo hoặc liên kết thần kinh trong việc phát hiện một số tính toán dữ liệu đầu vào nhất định trong kinh doanh thông minh. Mô hình perceptron cũng được phân loại là một trong những loại mạng thần kinh nhân tạo tốt nhất và cụ thể nhất. Là một thuật toán học có giám sát của các bộ phân loại nhị phân, chúng ta cũng có thể coi đây là mạng thần kinh một lớp với bốn tham số chính: giá trị đầu vào, trọng số và độ lệch, tổng ròng và hàm kích hoạt.

Perceptron và Neuron: Perceptron là một mô hình toán học của neuron sinh học. Nó tạo ra đầu ra nhị phân từ các giá trị đầu vào trong khi xem xét các trọng số và giá trị ngưỡng. Mặc dù được tạo ra để bắt chước hoạt động của các neuron sinh học, nhưng mô hình perceptron sau đó đã được thay thế bằng các mô hình tiên tiến hơn như mạng lan truyền ngược để huấn luyện mạng neuron nhân tạo. Perceptron sử dụng chức năng kích hoạt giòn để cung cấp đầu ra dương hoặc âm dựa trên một giá trị cụ thể. Một neuron, còn được gọi là nút trong mạng neuron nhân tạo lan truyền ngược tạo ra các giá trị được phân loại từ 0 đến 1. Đó là sự khái quát hóa ý tưởng của perceptron vì neuron cũng thêm các đầu vào có trọng số. Tuy nhiên, nó không tạo ra đầu ra nhị phân mà là giá trị được phân loại dựa trên độ gần của đầu vào với giá trị mong muốn là 1. Các kết quả thiên

về các giá trị cực trị 0 hoặc 1 vì nút sử dụng hàm đầu ra sigmoid. Các giá trị được phân loại có thể được diễn giải để xác định xác suất của loại đầu vào.



Ảnh 14: Luật học Perceptrion

5.5.2. Các thành phần cơ bản của một perceptron

Lớp đầu vào: Lớp đầu vào bao gồm một hoặc nhiều neuron đầu vào, nhận tín hiệu đầu vào từ thế giới bên ngoài hoặc từ các lớp khác của mạng lưới thần kinh.

Trọng số: Mỗi neuron đầu vào được liên kết với một trọng số, đại diện cho cường độ kết nối giữa neuron đầu vào và neuron đầu ra.

Độ lệch: Một thuật ngữ sai lệch được thêm vào lớp đầu vào để cung cấp cho perceptron tính linh hoạt bổ sung trong việc mô hình hóa các mẫu phức tạp trong dữ liệu đầu vào.

Hàm kích hoạt: Hàm kích hoạt xác định đầu ra của perceptron dựa trên tổng trọng số của đầu vào và thuật ngữ sai lệch. Các hàm kích hoạt phổ biến được sử dụng trong perceptron bao gồm hàm bước, hàm sigmoid và hàm ReLU.

Đầu ra: Đầu ra của perceptron là một giá trị nhị phân duy nhất, 0 hoặc 1, cho biết loại hoặc danh mục mà dữ liệu đầu vào thuộc về.

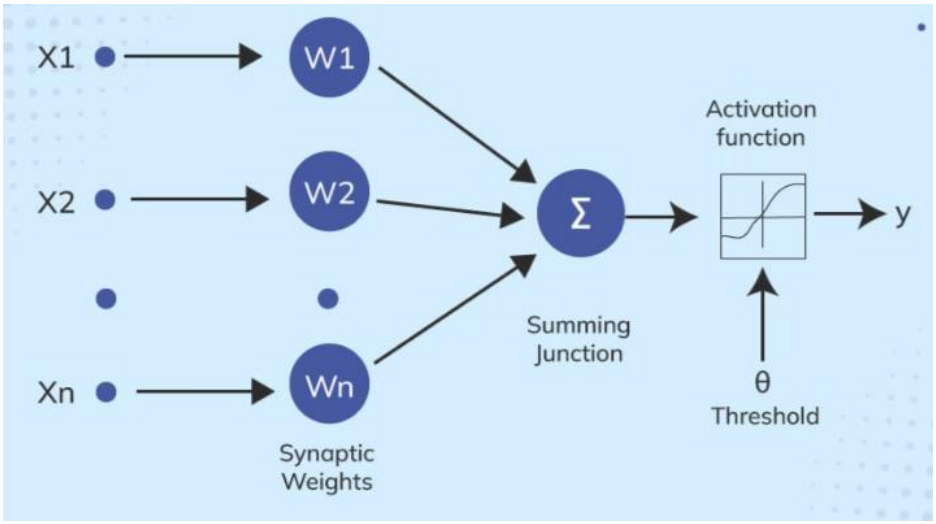
Thuật toán đào tạo: Perceptron thường được đào tạo bằng thuật toán học có giám sát, chẳng hạn như thuật toán học perceptron hoặc lan truyền ngược. Trong quá trình huấn luyện, trọng số

và độ lệch của perceptron được điều chỉnh để giảm thiểu sai số giữa đầu ra dự đoán và đầu ra thực sự đối với một tập hợp các ví dụ huấn luyện nhất định.

⇒ Nhìn chung, perceptron là một thuật toán đơn giản nhưng mạnh mẽ, có thể được sử dụng để thực hiện các nhiệm vụ phân loại nhị phân và đã mở đường cho các mạng thần kinh phức tạp hơn được sử dụng trong học sâu ngày nay.

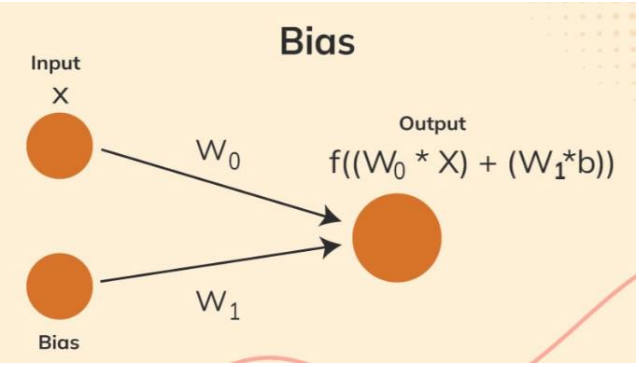
5.5.3. Tại sao cần có trọng số và độ lệch?

Trọng số và độ lệch là hai khía cạnh quan trọng của mô hình perceptron. Đây là những tham số có thể học được và khi mạng được huấn luyện, nó sẽ điều chỉnh cả hai tham số để đạt được giá trị mong muốn và đầu ra chính xác.



Ảnh 15: Tại sao chúng ta cần Trọng lượng và Độ lệch?

Trọng số được sử dụng để đo lường tầm quan trọng của từng tính năng trong việc dự đoán giá trị đầu ra. Các đặc điểm có giá trị gần bằng 0 được cho là có trọng số hoặc tầm quan trọng thấp hơn. Chúng ít quan trọng hơn trong quá trình dự đoán so với các đặc trưng có giá trị xa hơn 0 được gọi là trọng số có giá trị lớn hơn. Bên cạnh các đặc trưng có trọng số cao có khả năng dự đoán lớn hơn các đặc trưng có trọng số thấp, trọng số cũng có thể dương hoặc âm. Nếu trọng số của một đối tượng là dương thì nó có mối quan hệ trực tiếp với giá trị đích và nếu nó âm thì nó có mối quan hệ nghịch đảo với giá trị đích.

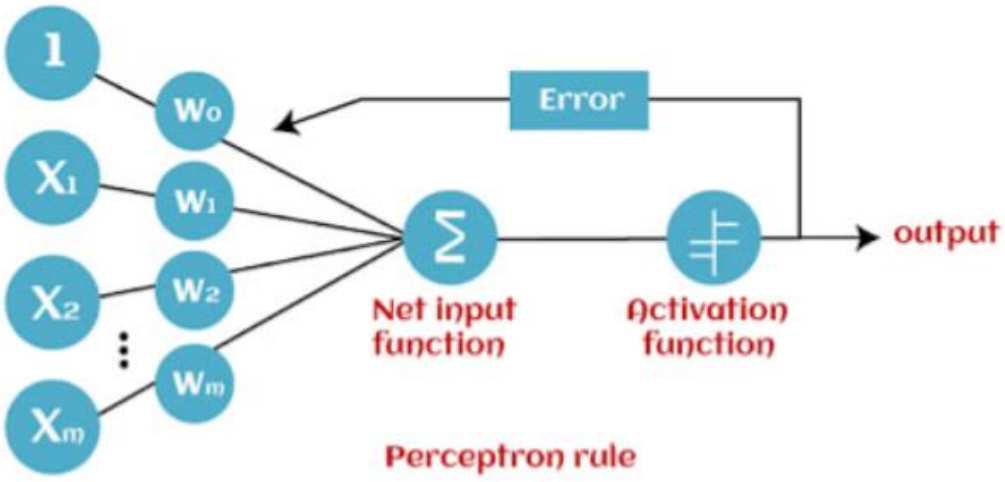


Ảnh 16: Bias

Ngược lại với trọng số trong mạng neuron làm tăng tốc độ kích hoạt chức năng kích hoạt, độ lệch làm trì hoãn việc kích hoạt chức năng kích hoạt. Nó hoạt động giống như một điểm chặn trong một phương trình tuyến tính. Nói một cách đơn giản, Bias là một hằng số được sử dụng để điều chỉnh đầu ra và giúp mô hình cung cấp đầu ra phù hợp nhất cho dữ liệu nhất định.

5.5.4. Các hoạt động của Perceptron

Perceptron được coi là liên kết thần kinh một lớp với 4 tham số chính. Mô hình perceptron bắt đầu bằng việc nhân tất cả các giá trị đầu vào và trọng số của chúng, sau đó cộng các giá trị này để tạo ra tổng có trọng số. Hơn nữa, tổng có trọng số này được áp dụng cho hàm kích hoạt *f* để thu được đầu ra mong muốn. Hàm kích hoạt này còn được gọi là hàm bước và được biểu thị bằng *f*.



Ảnh 17: Luật học Perceptron

Hàm bước hoặc hàm kích hoạt này rất quan trọng trong việc đảm bảo rằng đầu ra được ánh xạ giữa (0,1) hoặc (-1,1). Hãy lưu ý rằng trọng số của đầu vào biểu thị cường độ của nút. Tương tự, giá trị đầu vào cho khả năng dịch chuyển đường cong hàm kích hoạt lên hoặc xuống.

5.6. THỰC HÀNH NEURAL NETWORK THÔNG QUA EXCEL

Đầu tiên, chúng ta sẽ cần phải khởi tạo giá trị trọng số w và bias b

Neuron	w1	w2	w3	w4	w5	w6	w7	w8	w9	b
1	2.986611	0.918438	0.995289	0.292043	2.674862	2.503609	2.435193	2.821204	0.320002	0.731305
2	1.245525	2.813976	1.013842	0.206636	2.073316	0.488967	2.149017	3.857962	0.999991	2.312305
3	1.235171	2.795811	1.567176	0.623029	3.666206	1.377034	2.41288	1.816279	1.367636	3.195212

Tiếp theo, để tính được neuron weighted sum chúng ta cần sử dụng slope-intercept có công thức

$$= SUMPRODUCT(X, INDEX(Z, A, 0)) + INDEX(B, A, 1)$$

Trong đó:

SUMPRODUCT(X, INDEX(Z, A, 0)):

- X là một dãy dữ liệu cần nhân theo từng phần tử với dãy được tạo bởi INDEX.
- INDEX(Z, A, 0):
    - Z là một ma trận dữ liệu cần trích xuất một hàng cụ thể.
    - A: xác định hàng nào trong Z sẽ được chọn.
    - 0: Chỉ định rằng INDEX sẽ trả về toàn bộ hàng A trong Z.
- SUMPRODUCT(X, INDEX(Z, A, 0)): Tính tổng của tích từng phần tử tương ứng giữa hai mảng X và hàng A của Z.

INDEX(B, A, 1):

- B là một cột dữ liệu từ đó cần trích xuất một giá trị cụ thể.
- A: xác định hàng nào trong B sẽ được chọn.
- 1: Chỉ định rằng INDEX sẽ trả về phần tử tại cột đầu tiên của hàng A trong B (ở đây, vì B là một cột duy nhất nên chỉ có cột 1).
- INDEX(B, A, 1): Trả về giá trị tại hàng A của B.

x1	x2	x3	x4	x5	x6	x7	x8	x9
10	10	10	8	6	1	8	9	1
1	1	1	1	2	1	3	1	1
3	1	1	1	2	1	1	1	1
3	1	1	1	2	1	2	1	1

và mỗi hàng là 1 neuron cho ra:

neuron weighted sum		
1	2	3
115.8161891	120.5414606	124.5524834
24.22380257	23.53288465	28.5484007
25.32663909	21.72590223	26.1929822
27.76183176	23.87491879	28.60586235

Sau đó chúng ta cần phải tìm giá trị lớn nhất, nhỏ nhất trong cột:

MAX	152.8442976	147.1142486	163.6870916
MIN	16.67855532	17.16153556	20.05643455

Sử dụng min max normalize. Để biến đổi mỗi đặc điểm, giá trị tối thiểu của đặc điểm đó được chuyển thành 0, giá trị tối đa được chuyển thành 1, và mọi giá trị khác được chuyển thành một số thập phân giữa 0 và 1.

$$\frac{value - min}{max - min}$$

Công thức Excel:

$$= (B - Z)/(X - Z)$$

Trong đó:

- X là giá trị đầu vào mà công thức sẽ chuẩn hóa
- Z là giá trị nhỏ nhất của vùng dữ liệu chứa X
- B là giá trị lớn nhất của vùng dữ liệu chứa X

Sử dụng công thức trên sẽ cho ta các giá trị sau:

normalized		
1	2	3
0.728065901	0.79551956	0.727533042
0.055412229	0.049028211	0.059123632
0.06351145	0.035123289	0.042724497
0.081395484	0.051660201	0.059523698

Hàm kích hoạt sigmoid đề cập đến việc sử dụng hàm sigmoid như một hàm kích hoạt trong neural network. Các hàm kích hoạt giống như những công cụ ra quyết định trong học máy. Chúng giúp xác định xem liệu các neuron cụ thể trong neural network có nên được kích hoạt hay không dựa trên đầu vào mà chúng nhận được.

$$= 1/(1 + EXP(0 - X))$$

Trong đó hàm EXP là hàm trả về lũy thừa số với mũ là normalized neuron weight sum X

sigmoid transformed		
1	2	3
0.674380704	0.689015257	0.674263682
0.513849514	0.512254598	0.514776604
0.515872528	0.50877992	0.5106795
0.520337644	0.512912179	0.514876532

Để tính được output ta sẽ cần sử dụng slope-intercept với công thức:

$$= MMULT(X, Z) + B$$

Trong đó:

- X là một dòng giá trị sigmoid
- Z là một cột giá trị weight
- B là giá trị bias

Hàm MMULT thực hiện phép nhân ma trận giữa hàng X và cột Z, trả về tích vô hướng của hai vector (tức là một giá trị tổng).

Với giá trị đã tạo trước:

Wo	Bo
2.889909379	0
3.09532652	1.694666386
1.543226689	2.611312848

Cho ra kết quả:

Output
5.12216803
3.864990766
3.853759068
3.885930512

Tiếp theo ta cần phải tính được điểm chia cắt (cut-off point) để quyết định được nhóm sẽ cho ra dự đoán và trong bài toán này dự đoán chỉ có thể là 2 hoặc 4.

Đầu tiên, ta tìm trung bình giá trị output của mỗi nhóm với số hàng input có class là 2 và số hàng input có class là 4.

$$= AVERAGEIFS(X, Z, B)$$

Trong đó:

- X là phạm vi chứa các giá trị output cần tính trung bình, với các giá trị được tính trung bình khi thỏa mãn điều kiện.
- Z là phạm vi điều kiện mà AVERAGEIFS sẽ kiểm tra.
- B là giá trị phân loại điều kiện để lọc Z. Chỉ những ô trong X tương ứng với các ô trong Z có giá trị bằng B mới được dùng để tính trung bình.

$$= COUNTIFS(X, Z)$$

Trong đó:

- X là phạm vi chứa các giá trị phân loại cần đếm khi thỏa mãn điều kiện
- Z là giá trị phân loại mục tiêu mà các giá trị X cần phải thỏa

Ta tính điểm cut off bằng công thức tính trung bình có trọng số giữa hai nhóm.

$$= (X1 * Z1 + X2 * Z2)/(Z1 + Z2)$$

Trong đó:

- X1, X2 là trung bình giá trị output của mỗi nhóm
- Z1, Z2 là số lượng dữ liệu có phân loại

	2	4	cutoff
mean	3.930124377	4.827297433	4.258489716
count	317	183	

Ta có thể tính sai số bằng cách bình phương sau khi output trừ đi class. Chúng ta sẽ dự đoán bằng cách so sánh với điểm cut off để phân loại.

$$= IF(X <= Z, 2,4)$$

Trong đó:

- X là giá trị ouput
- Z là giá trị điểm cut off

Sau khi dự đoán xong, so sánh nhóm dự đoán với nhóm đầu vào nếu khác nhau để số 1, nếu không khác nhau để số 0.

Error	Predict	Diff
1.259261087	4	0
3.478190557	2	0
3.436422681	2	0
3.556733897	2	0

Nếu tổng của cột Error và Diff quá lớn chúng ta có thể sử dụng solver hoặc gradient descent để tìm được trọng số tối ưu.

CHƯƠNG VI: FREQUENT ITEMSET MINING

6.1. ĐỊNH NGHĨA

6.1.1. Frequent itemset là gì?

Các frequent pattern là các mẫu xuất hiện thường xuyên trong một tập dữ liệu. Một tập hợp mục thường xuyên là một tập hợp chứa một trong những mẫu này, đó là lý do tại sao khai thác mẫu thường xuyên thường được gọi là khai thác tập hợp mục thường xuyên *(frequent itemset mining)*.

Khai thác mẫu thường xuyên được giải thích dễ dàng nhất thông qua phân tích giỏ hàng, một ứng dụng phổ biến. Phân tích giỏ hàng cố gắng xác định các mối quan hệ, hoặc mẫu, giữa các sản phẩm mà một khách hàng cụ thể đã chọn và đặt vào giỏ hàng của họ *(dù là giỏ hàng thực tế hay ảo)*, và áp dụng các thước đo hỗ trợ và độ tin cậy để so sánh. Giá trị của việc này nằm ở tiếp thị chéo và phân tích hành vi khách hàng.

Sự tổng quát của phân tích giỏ hàng là khai thác mẫu thường xuyên, và thực chất nó khá giống với phân loại, ngoại trừ việc bất kỳ thuộc tính hoặc tổ hợp thuộc tính nào *(không chỉ lớp)* cũng có thể được dự đoán trong một mối liên kết. Vì mối liên kết không yêu cầu gán nhãn trước cho các lớp, nó là một hình thức học không giám sát.

6.1.2. Luật kết hợp

Nếu chúng ta nghĩ về tổng số mặt hàng có sẵn trong giỏ hàng của mình *(được bán tại cửa hàng thực tế, tại nhà bán lẻ trực tuyến)*, thì mỗi mặt hàng có thể được biểu thị bằng một biến Boolean, thể hiện liệu mặt hàng đó có nằm trong một "giỏ" nhất định hay không. Mỗi giỏ khi đó chỉ đơn giản là một vectơ Boolean, có thể khá dài tùy thuộc vào số lượng mặt hàng có sẵn. Khi đó, tập dữ liệu sẽ là ma trận kết quả của tất cả các vectơ giỏ có thể có.

Sau đó, tập hợp các vector Boolean này được phân tích để tìm mối liên kết, mẫu, mối tương quan hoặc bất kỳ tên nào mà ta muốn gọi những mối quan hệ này. Một trong những cách phổ biến nhất để biểu diễn các mẫu này là thông qua các luật kết hợp. Lấy ví dụ, ta có 1 giỏ hàng trong đó các sản phẩm bánh mì và bơ có mối liên hệ tin cậy 3/5 với sữa sản phẩm, trong khi sản phẩm bia có mối liên hệ tin cậy 1/2 với sản phẩm tã lót.

	Milk	Bread	Butter	Beer	Diapers
0	Yes	Yes	Yes	No	No
1	No	Yes	Yes	No	No
2	Yes	Yes	No	No	Yes
3	No	Yes	Yes	Yes	No
4	Yes	Yes	Yes	Yes	Yes
5	Yes	Yes	Yes	No	Yes

{Bread, Butter} → Milk (3/5 confidence)

Beer → Diapers (1/2 confidence)

Ảnh 18: Ví dụ về luật kết hợp (1)

Cách tiếp cận phổ biến nhất để tìm ra mối liên hệ giữa các mục là đếm các mục thường xuyên nhất. Trong hình bên dưới, ta chỉ quan tâm đến những mục xuất hiện nhiều hơn 2 lần, nghĩa là chúng ta gạch bỏ bất kỳ mục nào có tần suất ≤ 2.

Items	Freq.
Milk	4
Bread	6
Butter	5
[redacted]	
Diapers	3

Items	Freq.
Milk, Bread	4
Milk, Butter	3
Milk, Diapers	3
Bread, Butter	5
Bread, Diapers	3
[redacted]	

Items	Freq.
Milk, Bread, Butter	3
Milk, Bread, Diapers	3
[redacted]	
[redacted]	

Ảnh 19: Ví dụ về luật kết hợp (2)

Vậy tại sao ta lại quan tâm đến những sản phẩm xuất hiện thường xuyên nhất? Từ quan điểm kinh doanh, loại phân tích này sẽ giúp:

- Tăng tỷ lệ mua các sản phẩm được mua cùng nhau.
- Quảng bá chéo bằng cách sử dụng các mối quan hệ định hướng.
- Tối ưu hóa giá cả, đặc biệt trong các đợt khuyến mãi.
- Quản lý hàng tồn kho, chẳng hạn như dự trữ đúng số lượng sản phẩm phụ thuộc.
- Tinh chỉnh hoạt động tiếp thị, chẳng hạn như nhắm mục tiêu các phân khúc dựa trên sở thích của chúng.

6.1.3. Confidence và Support

Làm sao ta biết một quy tắc nhất định có thể ấn tượng hoặc sâu sắc đến mức nào? Đó là nơi mà confidence và support xuất hiện. Giả sử ta có mối liên hệ {bánh mì, bơ} ⇒ sữa có mức hỗ trợ *(support)* = 25% và độ tin cậy *(confidence)* = 60%.

Support là thước đo tần số tuyệt đối. Mức hỗ trợ 25% cho thấy bánh mì, bơ và sữa được mua cùng nhau trong 25% tổng số giao dịch.

Confidence là thước đo tần số tương quan. Độ tin cậy 60% cho thấy 60% người mua bánh mì và bơ cũng mua sữa.

Trong một ứng dụng nhất định, các quy tắc kết hợp thường được tạo trong giới hạn của một số ngưỡng tối thiểu được xác định trước cho cả độ tin cậy và độ hỗ trợ, đồng thời các quy tắc chỉ được coi là ấn tượng và sâu sắc nếu chúng đáp ứng các ngưỡng tối thiểu này.

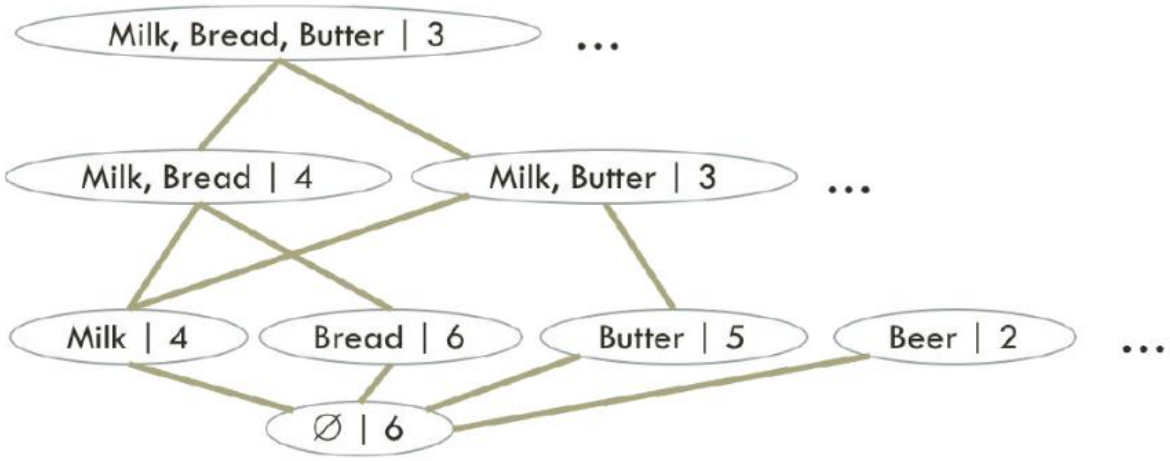
6.1.4. Biểu diễn bài toán tổng quát

Giả sử ta có một tập các mục I = {i1, i2, …, im}. Ta cũng có một tập hợp các giao dịch D trong đó mỗi giao dịch T là một tập hợp các mục sao cho T là tập con của I. Mỗi T có một mã định danh duy nhất liên quan được gọi là TID.

- X → Y là quy tắc kết hợp trong đó X và Y đều là tập con của I và không phải là tập rỗng.
- X → Y đúng với D với mức độ tin cậy c nếu ít nhất c% giao dịch trong D chứa X cũng chứa Y.
- X → Y có độ hỗ trợ s nếu s% giao dịch trong D chứa giao điểm của X và Y.

Cho một tập các giao dịch D, ta mong muốn tạo ra tất cả các luật kết hợp X → Y có độ hỗ trợ và độ tin cậy lớn hơn độ hỗ trợ tối thiểu và độ tin cậy tối thiểu do người dùng chỉ định tương ứng. Như đã định nghĩa ở trên, độ hỗ trợ, trong tập mục phổ biến, là số lượng giao dịch bao gồm 2 mục. Ta xác định mức hỗ trợ tối thiểu cần thiết để các quy tắc được coi là ấn tượng. Bắt đầu với các bộ vật phẩm đơn lẻ, ta xây dựng các bộ vật phẩm mới ở mỗi cấp độ nếu chúng tiếp tục đáp ứng mức hỗ trợ tối thiểu.

Một cách tiếp cận hữu ích để tìm các luật kết hợp này là sử dụng lattices. Hình dưới đây cho thấy một ví dụ về cách chúng ta xây dựng một mạng từ dưới lên, bắt đầu từ các tập hợp rỗng cho đến tập hợp có kích thước 3.



Ảnh 20: Mạng từ dưới lên tìm luật kết hợp

6.2. THUẬT TOÁN APRIORI

Apriori là một thuật toán để khai thác tập mục thường xuyên và học quy tắc kết hợp trên cơ sở dữ liệu giao dịch. Nó tiến hành bằng cách xác định các mục riêng lẻ thường xuyên trong cơ sở dữ liệu và mở rộng chúng thành các tập mục ngày càng lớn hơn miễn là các tập mục đó xuất hiện đủ thường xuyên trong cơ sở dữ liệu.

Apriori sử dụng cách tiếp cận "từ dưới lên", trong đó các tập hợp con thường xuyên được mở rộng từng mục một (một bước được gọi là tạo ứng cử viên) và các nhóm ứng viên được kiểm tra dựa trên dữ liệu. Thuật toán kết thúc khi không tìm thấy phần mở rộng thành công nào nữa.

Apriori sử dụng tìm kiếm theo chiều rộng và cấu trúc cây Hash để đếm các tập mục ứng cử viên một cách hiệu quả. Nó tạo ra các tập mục ứng cử viên có độ dài k từ các tập mục có độ dài (k - 1). Sau đó, nó sẽ lược bỏ các ứng cử viên có mẫu con không thường xuyên. Theo bổ đề đóng xuống, tập ứng cử viên chứa tất cả các tập mục phổ biến có độ dài k. Sau đó, nó quét cơ sở dữ liệu giao dịch để xác định các tập phổ biến trong số các ứng cử viên.

Pseudocode của thuật toán:

```
C_k: Candidate itemset of size k
L_k : frequent itemset of size k
L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) do begin
    C_{k+1} = candidates generated from L_k;
    for each transaction t in database do
        increment the count of all candidates in C_{k+1} are contained in t
    L_{k+1}= candidates in C_{k+1} with min_support
end
return ∪_k L_k;
```

Trong đó:

- *k-itemset* là một tập phần tử có kích thước *k* với các phần tử được sắp xếp theo từ điển.
- L_k là tập hợp *k-itemset* với độ hỗ trợ tối thiểu chứa các mục và số lượng.
- C_k là một tập *k-itemset* ứng cử viên, mỗi bộ chứa các mục và số đếm.

Cách hoạt động của thuật toán Apriori

Ta có bảng dữ liệu bên dưới và đặt độ hỗ trợ tối thiểu là 2.

Transaction	Items
1	Milk, Bread, Butter
2	Bread, Cheese
3	Bread, Jam
4	Milk, Bread, Cheese
5	Milk, Jam
6	Bread, Jam
7	Milk, Jam
8	Milk, Bread, Butter, Jam
9	Milk, Bread, Jam

Bước đầu tiên là tạo tất cả các mục 1 mặt hàng cũng như hỗ trợ tương ứng của họ (L_1), trong đó có 5 trong số đó.

Item	Support

{Milk}	6
{Bread}	7
{Butter}	2
{Cheese}	2
{Jam}	6

Vì mức hỗ trợ tối thiểu là 2 nên tất cả các mục ở đây đều đáp ứng yêu cầu. Điều này có nghĩa là $C_1 = L_2$. Tiếp theo, ta có thể tạo tất cả tập gồm 2 mục ứng viên (C_2). Gồm: {Milk, Bread}, {Milk, Butter}, {Milk, Cheese}, {Milk, Jam}, {Bread, Butter}, {Bread, Cheese}, {Bread, Jam}, {Butter, Cheese}, {Butter, Jam}, {Cheese, Jam}

Tuy nhiên, không phải tất cả các kết hợp kích thước 2 này đều đáp ứng yêu cầu hỗ trợ tối thiểu. Trong quá trình cắt tỉa, chúng ta có thể loại bỏ 4 tổ hợp, để lại 6 tập 2 mục (L_2).

Items	Support
{Milk, Bread}	4
{Milk, Butter}	2
{Milk, Jam}	2
{Bread, Butter}	2
{Bread, Cheese}	2
{Bread, Jam}	4

Chuyển sang cấp độ thứ ba, trong đó có 5 bộ 3 mục ứng viên (C_3) được tạo ra từ 6 bộ 2 mục (L_2).

Items	Support
{Milk, Bread, Butter}	2
{Milk, Bread, Jam}	1
{Milk, Butter, Jam}	1
{Bread, Butter, Cheese}	1
{Bread, Butter, Jam}	1

{Bread, Cheese, Jam}	0

Loại các items có độ support dưới 2, ta còn:

Items	Support
{Milk, Bread, Butter}	2

6.3. KHAI THÁC LUẬT KẾT HỢP

Nhớ rằng các luật kết hợp có dạng X → Y. Điều này gợi ý rằng nếu X xuất hiện trong một giao dịch, thì Y cũng có khả năng xuất hiện trong cùng giao dịch đó. Các thuật toán khai thác luật sử dụng các chỉ số về độ mạnh và/hoặc độ thú vị để đề xuất các quy luật có thể xảy ra, và các quy luật này được xác nhận thông qua độ hỗ trợ.

Độ tin cậy là một dấu hiệu cho thấy mức độ thường xuyên mà quy tắc này được cho là đúng. Giá trị tin cậy của một quy tắc, X→ Y, đối với một tập hợp các giao dịch D, là tỷ lệ các giao dịch chứa X và cũng chứa Y.

$$confidence(X \rightarrow Y) = p(X) = \frac{support(X \cup Y)}{support(X)}$$

Lift của quy tắc là tỷ lệ giữa mức hỗ trợ được quan sát so với mức hỗ trợ dự kiến nếu X và Y độc lập.

$$lift(X \rightarrow Y) = \frac{support(X \cup Y)}{support(X) \times support(Y)}$$

Để chứng minh điều đó, giả sử chúng ta có quy tắc kết hợp Sữa → Bánh mì.

Transaction	Items
1	Milk, Bread, Butter
2	Bread, Cheese
3	Bread, Jam
4	Milk, Bread, Cheese
5	Milk, Jam

6	Bread, Jam
7	Milk, Jam
8	Milk, Bread, Butter, Jam
9	Milk, Bread, Jam

Ta sẽ tính toán các giá trị sau:

$$support(Milk) = \frac{6}{9}; support(Bread) = \frac{7}{9};$$

$$support(Milk \cup Break) = \frac{4}{9}$$

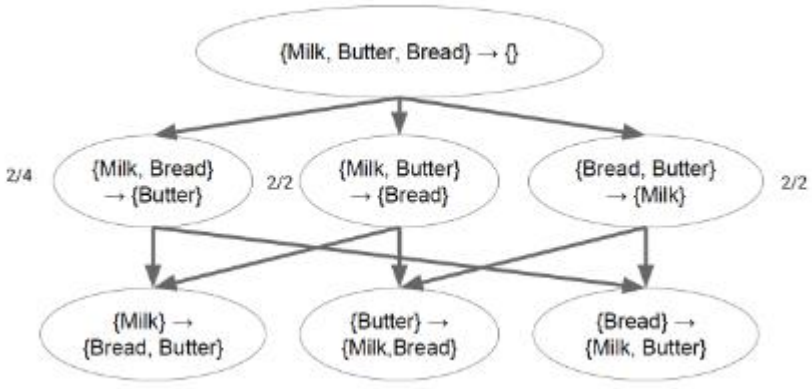
$$\Rightarrow confidence(Milk \rightarrow Bread) = \frac{4}{9} \div \frac{6}{9} = \frac{4}{6}$$

$$lift(Milk \rightarrow Bread) = \frac{4}{9} \div (\frac{6}{9} \times \frac{7}{9}) = \frac{6}{7}$$

6.4. LUẬT LATTICE

Đối với mỗi tập phổ biến có k > 1, ta xem xét tất cả các luật kết hợp có thể có.

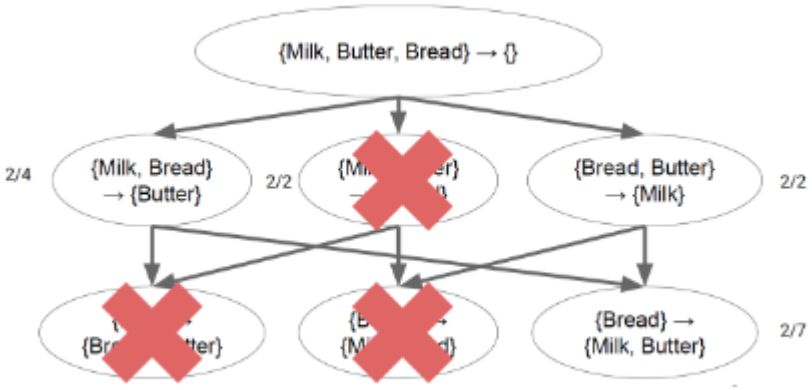
Ví dụ: Nếu quy tắc {Milk, Butter} → {Bread} không đáp ứng yêu cầu hỗ trợ tối thiểu thì 2 quy tắc kết hợp ở cấp độ bên dưới được kết nối với nó ({Milk} → {Bread, Butter} và {Butter} → {Milk, Bread}) cũng không thể bị loại bỏ.



Ảnh 21: Luật Lacttice (1)

Tập hợp mục thường xuyên tối đa

Nhìn vào bảng dưới đây, giả sử chúng ta có một tập hợp 3 mục {Milk, Bread, Butter} với độ hỗ trợ là 2. Điều này có nghĩa là chúng ta không cần xem xét bất kỳ tập hợp 2 mục và 1 mục nào chứa các sản phẩm này vì điều đó sẽ là dư thừa. Do đó, tôi có thể loại bỏ {Milk}, {Bread}, {Butter}, {Milk, Bread}, {Milk, Butter}, và {Bread, Butter} khỏi tính toán của mình.



Ảnh 22: Luật Lacttice (2)

6.5. CÀI ĐẶT THUẬT TOÁN APRIORI VỚI THƯ VIỆN SKLEARN

6.5.1. Dataset

InvoiceNo	Quantity	StockCode
1	12	s21
2	2	s2
2	1	s4
2	2	s8
2	3	s19
2	2	s25
2	2	s9
3	2	s4
3	2	s24

…	…	…
321	1	s10
321	1	s3

Download full dataset tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/datasets/apriori/apriori.csv

Dataset này là một tập dữ liệu bán lẻ trực tuyến, chứa thông tin về các hóa đơn và các mặt hàng trong các giao dịch thương mại. Dữ liệu đã được xử lý trước để phục vụ cho việc phân tích.

Với dataset trên ta có:

Các cột chính là:

- InvoiceNo: Số hóa đơn, đại diện cho một giao dịch hoặc một biên lai. Mỗi hóa đơn có thể chứa nhiều mặt hàng khác nhau, và các mặt hàng trong cùng một hóa đơn sẽ có cùng số InvoiceNo.
- Quantity: Số lượng của một mặt hàng nhất định được bán trong một hóa đơn cụ thể.
- StockCode: Mã sản phẩm, đại diện cho từng mặt hàng riêng biệt trong kho.

Tổng số mã sản phẩm: 25 mã khác nhau, đại diện cho các mặt hàng riêng lẻ.

Tổng số giao dịch: 1113 giao dịch, mỗi giao dịch có thể gồm một hoặc nhiều hóa đơn.

Tổng số hóa đơn: 321 hóa đơn, mỗi hóa đơn tương ứng với một biên lai duy nhất chứa thông tin về các mặt hàng và số lượng bán.

6.5.2. Xây dựng mô hình bằng code Python

Bước 1: Tạo class AprioriWLib.py với phương thức khởi tạo *_init_* và phương thức *training()* dùng để huấn luyện model và in ra itemset và các luật kết hợp.

Bước 2: Trong phương thức khởi tạo viết code đọc training data từ file csv và gán vào dataframe. Chuyển đổi dữ liệu vừa đọc được về dạng hóa đơn (1 hóa đơn sẽ gồm 1 hoặc nhiều mặt hàng và 1 mặt hàng được mua với số lượng 1 hoặc nhiều)

```
def __init__(self):
    csv_handler = CSVHandler('Dataset\\apriori.csv')
    dataframe = csv_handler.read_csv()
```

```
    # Chuyển đổi dữ liệu sang dạng giỏ hàng
    self.transactions = dataframe.groupby('InvoiceNo')['StockCode'].apply(list).tolist()
```

Bước 3: Trong phương thức *training()* thực hiện encode để chuẩn hóa các dữ liệu dạng chữ về thành số:

```
    encoder = TransactionEncoder()
    onehot = encoder.fit(self.transactions).transform(self.transactions)
    onehot_df = pd.DataFrame(onehot, columns=encoder.columns_)
```

Bước 4: Trong phương thức *training()* tiếp tục viết code sử dụng thuật toán apriori để lấy các *itemset* phổ biến với *min_support=0.05* và các quy tắc kết hợp nhận được từ tập itemset này.

```
    frequent_itemsets = apriori(onehot_df, min_support=0.05, use_colnames=True)

    # Tạo quy tắc kết hợp từ các tập hợp sản phẩm
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
```

Bước 5: Vào file *main.py* gọi các hàm trên và xem kết quả.

Download source code tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/Apriori/AprioriWLib.py

6.6. GIẢI THUẬT TOÁN BẰNG EXCEL

Dữ liệu đầu vào:

ReceiptNo	Quantity	ItemCode
1	2	I28
1	6	I46
1	6	I47
1	8	I48
1	6	I50
2	6	I20
2	6	I21
3	6	I26
3	6	I27
3	32	I44
4	6	I38
5	12	I24
5	24	I25
6	80	I8
7	6	I20
7	6	I21

Đầu tiên ta phải điếm số lần ItemCode xuất hiện trong một hóa đơn với công thức sau:

$$= COUNTIFS(X1, X2\ , X3, X4)$$

Trong đó:

- X1: vùng dữ liệu đầu vào ReceiptNo
- X2: ô có Receipt# tương ứng
- X3: vùng dữ liệu đầu vào ItemCode trong từng hóa đơn
- X4: ô có ItemCode để so sánh với ItemCode trong mỗi hóa đơn

Receipt#	I1	I2	I3	I4	I5	I6	I7	I8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0

Trước đó, ta sẽ tính số lần ItemCode xuất hiện trong dãy hóa đơn input.

Tiếp theo ta cần phải tính số lần xuất hiện của tập các hạng mục (itemset) có tập k-hạng mục (k-itemset) là 2 trong các hóa đơn input. Với công thức:

$$= COUNTIFS(INDEX(X1,0, X2),1, X3,1)$$

Trong đó:

- X1: vùng dữ liệu của số lần các ItemCode xuất hiện trong các hóa đơn
- X2: ô chứa vị trí của cột ItemCode đang xét qua
- X3: vùng dữ liệu của số lần ItemCode xuất hiện trong các hóa đơn đang xét qua

Công thức trên để đếm các hàng có giá trị là 1 ở hai cột đang xét

		I1	I2	I3	I4	I5	I6
	Occurrenc	20	28	36	26	23	26
1	I1	20	1	2	3	1	3
2	I2	1	28	9	5	8	5
3	I3	2	9	36	12	9	11
4	I4	3	5	12	26	6	8
5	I5	1	8	9	6	23	8
6	I6	3	5	11	8	8	26
7	I7	2	7	10	6	8	5

Support (Hỗ trợ): có giá trị = 0.03 có nghĩa là ta muốn tìm các quy tắc kết hợp mà có ít nhất 3% giao dịch trong tập dữ liệu chứa cả hai mục (item). Nếu tỷ lệ xuất hiện của một mục (hoặc một tập hợp các mục) nhỏ hơn 0.03, quy tắc đó sẽ không được xem xét.

Confidence (Tin cậy): có giá trị = 0.5 có nghĩa là ta chỉ muốn xem xét các quy tắc mà trong đó, khi mục A đã xuất hiện, có ít nhất 50% xác suất mục B cũng sẽ xuất hiện. Nếu tỉ lệ này nhỏ hơn 0.5, quy tắc sẽ bị loại bỏ.

Điều kiện đầu tiên: Nếu ItemCode ở hàng và cột bằng nhau, không tính toán gì và trả về chuỗi rỗng.

Điều kiện thứ hai:

- Kiểm tra số lần ItemCode xuất hiện trong các hóa đơn có phải lớn hơn 0 không.
- Tính tỉ lệ hỗ trợ của bộ mục và so sánh với 0.03. Nếu tỷ lệ này đạt yêu cầu, nghĩa là ít nhất 3% giao dịch chứa bộ mục đó.
- Tính tỉ lệ tin cậy và so sánh với 0.5. Điều này có nghĩa là ít nhất 50% số giao dịch có mục A sẽ có mục B.

Công thức Excel:

$$= IF(\$DG3 = DH\$1, "", IF(AND(BF\$2 > 0, BF3/371 >= \$DF\$3, BF3/BF\$2 >= \$DF\$5), TEXT(BF3/371, "0.000")\&", "\&TEXT(BF3/BF\$2, "0.000"), ""))$$

Trong đó:

IF($DG3=DH$1, "", ...):

- Kiểm tra xem ô $DG3 có bằng với ô DH$1 không.

- Nếu đúng ($DG3=DH$1), công thức trả về một chuỗi trống "".
- Nếu sai, tiếp tục với phần IF lồng tiếp theo.

IF(AND(...), TEXT(...), ""):

- Nếu $DG3 không bằng DH$1, công thức thực hiện một kiểm tra có 3 điều kiện sử dụng hàm AND:
  - -BF$2 > 0: Kiểm tra xem giá trị ở ô BF$2 có lớn hơn 0 không.
  - -BF3/371 >= $DF$3: Kiểm tra xem giá trị của BF3 chia cho 371 có lớn hơn hoặc bằng giá trị ở $DF$3 không.
  - -BF3/BF$2 >= $DF$5: Kiểm tra xem giá trị của BF3 chia cho BF$2 có lớn hơn hoặc bằng giá trị ở $DF$5 không.
- Nếu tất cả các điều kiện trên đều thỏa mãn, thì kết quả sẽ là:
  - -TEXT(BF3/371, "0.000"): Chia giá trị ở ô BF3 cho 371, rồi định dạng kết quả với 3 chữ số thập phân.
  - -& ", " &: Nối dấu phẩy và khoảng trắng để tách hai giá trị.
  - -TEXT(BF3/BF$2, "0.000"): Chia BF3 cho BF$2, rồi định dạng kết quả với 3 chữ số thập phân.
- Nếu không thỏa mãn điều kiện nào đó trong hàm AND, công thức trả về một chuỗi trống "".

		I1	I2	I3	I4
Support					
0.03	I1				
Confidence	I2				
0.5	I3				
	I4				
	I5				
	I6				
	I7				
	I8				
	I9				0.035, 0.500

CHƯƠNG VII: ĐÁNH GIÁ PHƯƠNG PHÁP PHÂN LỚP

7.1. TỔNG QUAN

Khi xây dựng một mô hình Machine Learning, chúng ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình.

Hiệu năng của một mô hình thường được đánh giá dựa trên tập dữ liệu kiểm thử *(test data)*. Cụ thể, giả sử đầu ra của mô hình khi đầu vào là tập kiểm thử được mô tả bởi vector *y_pred* - là vector dự đoán đầu ra với mỗi phần tử là class được dự đoán của một điểm dữ liệu trong tập kiểm thử. Ta cần so sánh giữa vector dự đoán *y_pred* này với vector class thật của dữ liệu, được mô tả bởi vector *y_true*.

Ví dụ với bài toán có 3 lớp dữ liệu được gán nhãn là 0, 1, 2. Trong bài toán thực tế, các class có thể có nhãn bất kỳ, không nhất thiết là số, và không nhất thiết bắt đầu từ 0. Chúng ta hãy tạm giả sử các class được đánh số từ 0 đến C-1 trong trường hợp có C lớp dữ liệu. Có 10 điểm dữ liệu trong tập kiểm thử với các nhãn thực sự được mô tả bởi *y_true = [0, 0, 0, 0, 1, 1, 1, 2, 2, 2]*. Giả sử bộ phân lớp chúng ta đang cần đánh giá dự đoán nhãn cho các điểm này là *y_pred = [0, 1, 0, 2, 1, 1, 0, 2, 1, 2]*.

Có rất nhiều cách đánh giá một mô hình phân lớp. Tuỳ vào những bài toán khác nhau mà chúng ta sử dụng các phương pháp khác nhau. Các phương pháp thường được sử dụng là: accuracy score, confusion matrix, ROC curve, Area Under the Curve, Precision and Recall, F1 score, Top R error,...

7.2. ACCURACY

Cách đơn giản và hay được sử dụng nhất là accuracy (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

Trong ví dụ này, ta có thể đếm được có 6 điểm dữ liệu được dự đoán đúng trên tổng số 10 điểm. Vậy ta kết luận độ chính xác của mô hình là 0.6 (hay 60%). Để ý rằng đây là bài toán với chỉ 3 class, nên độ chính xác nhỏ nhất đã là khoảng 1/3, khi tất cả các điểm được dự đoán là thuộc vào một class nào đó.

7.3. CONFUSION MATRIX

Cách tính sử dụng accuracy như ở trên chỉ cho chúng ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là confusion matrix.

Về cơ bản, confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class. Để hiểu rõ hơn, hãy xem bảng dưới đây:

Total: 10	Predicted as: 0	Predicted as: 1	Predicted as: 2	
True: 0	2	1	1	4
True: 1	1	2	0	3
True: 2	0	1	2	3

Có tổng cộng 10 điểm dữ liệu. Chúng ta xét ma trận tạo bởi các giá trị tại vùng 3x3 trung tâm của bảng.

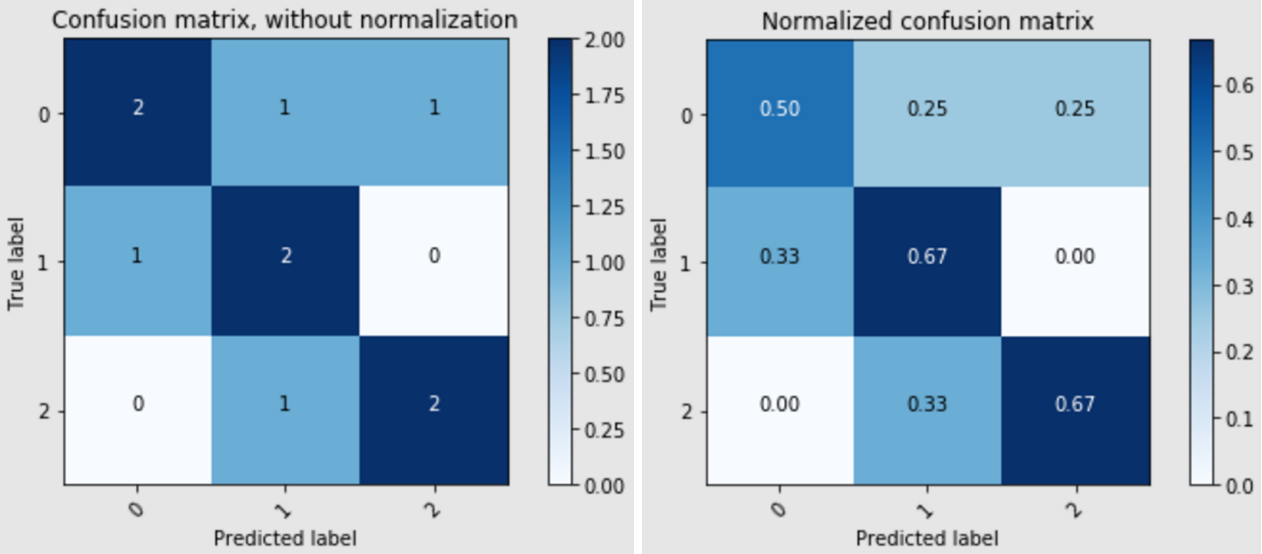
Ma trận thu được được gọi là confusion matrix. Nó là một ma trận vuông với kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ i, cột thứ j là số lượng điểm lẽ ra thuộc vào class i nhưng lại được dự đoán là thuộc vào class j. Như vậy, nhìn vào hàng thứ nhất (0), ta có thể thấy được rằng trong số bốn điểm thực sự thuộc lớp 0, chỉ có hai điểm được phân loại đúng, hai điểm còn lại bị phân loại nhầm vào lớp 1 và lớp 2.

Chúng ta có thể suy ra ngay rằng tổng các phần tử trong toàn ma trận này chính là số điểm trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số điểm được phân loại đúng của mỗi lớp dữ liệu. Từ đây có thể suy ra accuracy chính bằng tổng các phần tử trên đường chéo chia cho tổng các phần tử của toàn ma trận.

Cách biểu diễn trên đây của confusion matrix còn được gọi là unnormalized confusion matrix, tức confusion matrix chưa chuẩn hoá. Để có cái nhìn rõ hơn, ta có thể dùng normalized confusion matrix, tức confusion matrix được chuẩn hoá. Để có normalized confusion matrix, ta lấy mỗi hàng của unnormalized confusion matrix sẽ được chia cho tổng các phần tử trên hàng đó. Như

vậy, ta có nhận xét rằng tổng các phần tử trên một hàng của normalized confusion matrix luôn bằng 1.

Confusion matrix thường được minh hoạ bằng màu sắc để có cái nhìn rõ ràng hơn. Với các bài toán với nhiều lớp dữ liệu, cách biểu diễn bằng màu này rất hữu ích. Các ô màu đậm thể hiện các giá trị cao. Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt.

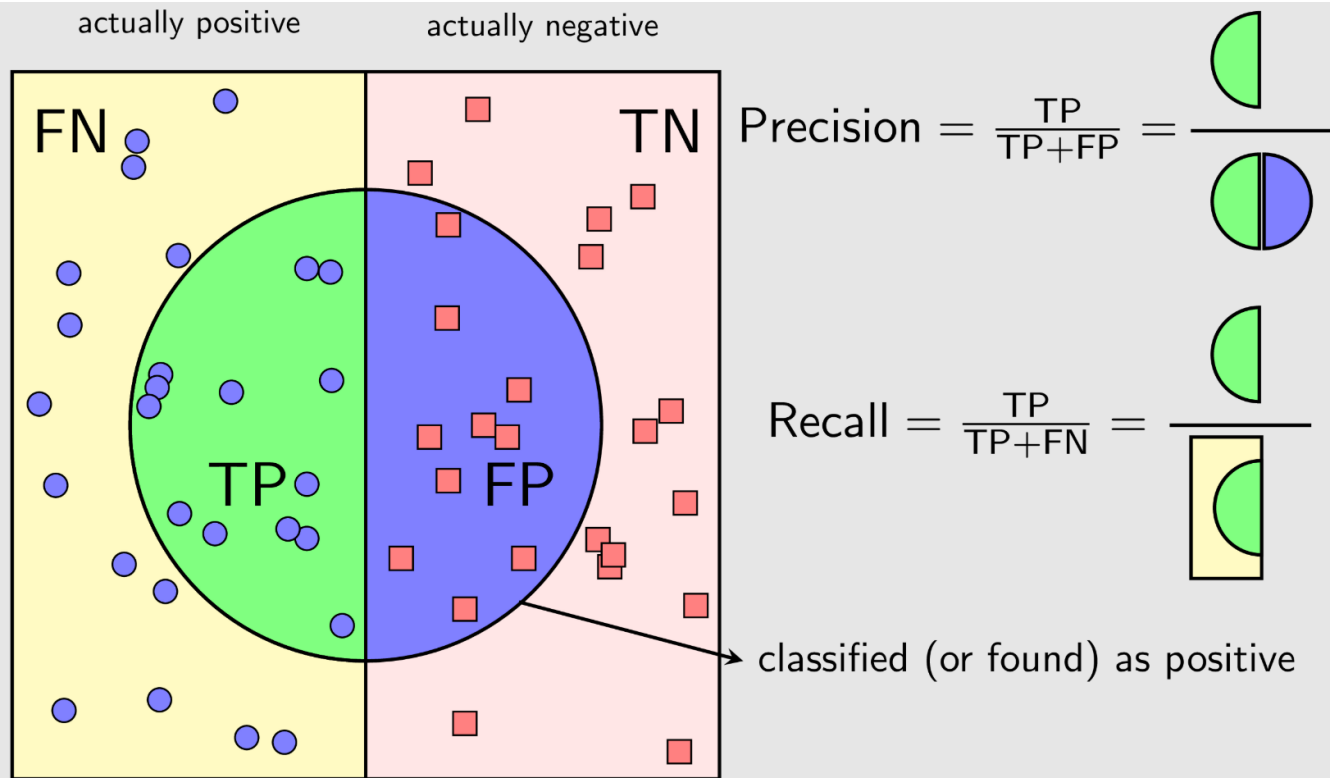


Từ hai hình trên ta thấy rằng confusion matrix đã chuẩn hoá mang nhiều thông tin hơn. Sự khác nhau được thấy ở ô trên cùng bên trái. Lớp dữ liệu 0 được phân loại không thực sự tốt nhưng trong unnormalized confusion matrix, nó vẫn có màu đậm như hai ô còn lại trên đường chéo chính.

7.4. PRECISION VÀ RECALL

Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đó hiệu quả thường được sử dụng là Precision-Recall.

Trước hết xét bài toán phân loại nhị phân. Ta cũng coi một trong hai lớp là positive, lớp còn lại là negative.



Với một cách xác định một lớp là positive, Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive (TP + FP).

Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive (TP + FN).

Một cách toán học, Precison và Recall là hai phân số có tử số bằng nhau nhưng mẫu số khác nhau:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Có thể nhận thấy rằng TPR và Recall là hai đại lượng bằng nhau. Ngoài ra, cả Precision và Recall đều là các số không âm nhỏ hơn hoặc bằng 1.

Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự positive là thấp.

Khi Precision = 1, mọi điểm tìm được đều thực sự là positive, tức không có điểm negative nào lẫn vào kết quả. Tuy nhiên, Precision = 1 không đảm bảo mô hình là tốt, vì câu hỏi đặt ra là liệu mô hình đã tìm được tất cả các điểm positive hay chưa. Nếu một mô hình chỉ tìm được đúng một điểm positive mà nó chắc chắn nhất thì ta không thể gọi nó là một mô hình tốt.

Khi Recall = 1, mọi điểm positive đều được tìm thấy. Tuy nhiên, đại lượng này lại không đo liệu có bao nhiêu điểm negative bị lẫn trong đó. Nếu mô hình phân loại mọi điểm là positive thì chắc chắn Recall = 1, tuy nhiên dễ nhận ra đây là một mô hình cực tồi.

Một mô hình phân lớp tốt là mô hình có cả Precision và Recall đều cao, tức càng gần một càng tốt. Có hai cách đo chất lượng của bộ phân lớp dựa vào Precision và Reall: Precision-Recall curve và F-score.

7.5. F1-SCORE

F1-score là harmonic mean của precision và recall (giả sử rằng hai đại lượng này khác 0):

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1 có giá trị trong khoảng (0,1], F1 càng cao bộ phân lớp càng tốt. Khi cả recall và precision đều bằng 1 (tốt nhất có thể), F1=1

7.6. THỰC HÀNH VỚI EXCEL

Dữ liệu đầu vào:

Iris type	x1	x2	x3	x4
Iris-setosa	4.6	3.1	1.5	0.2
Iris-setosa	4.6	3.4	1.4	0.3
Iris-setosa	4.8	3.4	1.6	0.2
Iris-setosa	5.7	4.4	1.5	0.4
Iris-setosa	4.8	3.4	1.9	0.2
Iris-setosa	5.2	4.1	1.5	0.1

Chúng ta cần phải chuyển kết quả về số để tính toán các bước tiếp.

0	Iris-setosa
1	Iris-versicolor
2	Iris-virginica

Iris type	x1	x2	x3	x4	y
Iris-setosa	4.6	3.1	1.5	0.2	0
Iris-setosa	4.6	3.4	1.4	0.3	0
Iris-setosa	4.8	3.4	1.6	0.2	0
Iris-setosa	5.7	4.4	1.5	0.4	0
Iris-setosa	4.8	3.4	1.9	0.2	0

Tiếp theo, ta sẽ phải tính toán các trọng số với công thức LINEST rồi sử dụng INDEX để lấy giá trị cần thiết.

		Obtained-by-LINEST
b	5	0.35469596
w1	4	-0.178346893
w2	3	-0.00549763
w3	2	0.267646734
w4	1	0.579921945

Sau khi có được trọng số, ta sẽ sử dụng công thức hồi quy tuyến tính để tính được giá trị được dùng để phân loại.

x1	x2	x3	x4	y	Numerical classification
4.6	3.1	1.5	0.2	0	0.03
4.6	3.4	1.4	0.3	0	0.06
4.8	3.4	1.6	0.2	0	0.02
5.7	4.4	1.5	0.4	0	-0.05
4.8	3.4	1.9	0.2	0	0.10

Chúng ta cần phải tính được điểm cut off để phân loại các giá trị vào một nhóm cụ thể. Đầu tiên, chúng ta phải tính được trung bình phân loại số của mỗi nhóm y và số lượng của mỗi loại. Sau đó, tính điểm cut off là trung bình có trọng số giữa 2 nhóm

mean	sample number	cutoff
-0.022849939	35	0.557350986
1.172715604	33	1.526466881
1.860003798	35	

Chúng ta chỉ cần phải so sánh phân loại số với điểm cut off để phân loại dữ liệu.

Numerical classification	Type classification
0.03	Iris-setosa
0.06	Iris-setosa
0.02	Iris-setosa
-0.05	Iris-setosa

Tiếp theo, chúng ta tính độ lệch bình phương giữa giá trị của một điểm dữ liệu với trung bình của nhóm đó và tìm xem nếu có nhóm đầu vào có khác nhóm dự đoán không nếu có thì để vào số 1.

Difference	Within g-variance
0	0.003313387
0	0.007593425
0	0.002209755
0	0.000886973

Chúng ta sẽ tính phương sai trong nhóm bằng cách cộng lại toàn bộ giá trị cột Within g-variance, phương sai giữa các nhóm với công thức:

$$= (X1 - X)\text{^}2 + (X2 - X)\text{^}2 + X3 - X)\text{^}2$$

Trong đó:

- X: trung bình tổng thể của tất cả các nhóm
- X1, X2, X3: trung bình của một nhóm

Sau khi có được hai giá trị trên, ta sẽ tính được Inter/within ratio (tỷ lệ giữa phương sai giữa các nhóm và phương sai trong nhóm) bằng cách chia Inter-group variance với Within-group variance.

Inter-group variance	1.815626742
Within-group variance	2.411469782
inter/within ratio	0.752912915

Sau khi tính được xong trang training_dataset, chúng ta sẽ qua trang testing_dataset tính tiếp với trọng số từ trang trước và không tính phương sai nữa.

Chúng ta tính số dự đoán sai và độ chính xác của mô hình

Cross validation missed =	2
Cross validation accuracy=	95.74%

7.7. THỰC HÀNH VỚI CODE PYTHON

```
import pandas as pd
import numpy as np

class ValidateNoLib:
  def __init__(self, ytrue, ypred):
    # Giả sử ytrue là DataFrame, chọn cột đầu tiên
    self.ytrue = ytrue.iloc[:, 0] if isinstance(ytrue, pd.DataFrame) else
pd.Series(ytrue).reset_index(drop=True)
    self.ypred = ypred.iloc[:, 0] if isinstance(ypred, pd.DataFrame) else
pd.Series(ypred).reset_index(drop=True)

    # Lấy các lớp
    self.classes = np.unique(np.concatenate((self.ytrue, self.ypred)))

  def getSampleSize(self):
    return len(self.ytrue)

  def getSampleClasses(self):
    return self.classes

  def confusionMatrix(self):
    self.matrix = np.zeros((len(self.classes), len(self.classes)), dtype=int)
    for true_label, pred_label in zip(self.ytrue, self.ypred):
      self.matrix[true_label, pred_label] += 1
    return self.matrix

  def accuracy(self):
    # Tính độ chính xác
    correct_predictions = (self.ytrue == self.ypred).sum()
    return correct_predictions / self.getSampleSize()

  def precision(self, label):
    # Tính precision cho một nhãn cụ thể
    tp = ((self.ytrue == label) & (self.ypred == label)).sum()
    fp = ((self.ytrue != label) & (self.ypred == label)).sum()
    return tp / (tp + fp) if (tp + fp) > 0 else 0.0
```

```
def recall(self, label):
    # Tính recall cho một nhãn cụ thể
    tp = ((self.ytrue == label) & (self.ypred == label)).sum()
    fn = ((self.ytrue == label) & (self.ypred != label)).sum()
    return tp / (tp + fn) if (tp + fn) > 0 else 0.0

def fscore(self, label, beta=1):
    # Tính F-score cho một nhãn cụ thể
    p = self.precision(label)
    r = self.recall(label)
    return (1 + beta**2) * (p * r) / (beta**2 * p + r) if (beta**2 * p + r) > 0 else 0.0
```

CHƯƠNG VIII: K-NEAREST NEIGHBORS (KNN)

8.1. TỔNG QUAN

8.1.1. K-nearest neighbors (KNN) là gì?

K-nearest neighbors là một loại thuật toán học có giám sát được sử dụng cho cả hồi quy và phân loại. K-nearest neighbors cố gắng dự đoán đúng lớp cho dữ liệu kiểm tra bằng cách tính khoảng cách giữa dữ liệu kiểm tra và tất cả các điểm huấn luyện. Sau đó chọn số K điểm gần với dữ liệu thử nghiệm. Thuật toán K-nearest neighbors tính toán xác suất dữ liệu kiểm tra thuộc các lớp dữ liệu huấn luyện 'K' và lớp có xác suất cao nhất sẽ được chọn. Trong trường hợp hồi quy, giá trị là giá trị trung bình của điểm huấn luyện đã chọn 'K'.

Thuật toán K-nearest neighbors giả định sự giống nhau giữa trường hợp/dữ liệu mới và các trường hợp có sẵn và đặt trường hợp mới vào danh mục giống nhất với các danh mục có sẵn.

Thuật toán K-nearest neighbors lưu trữ tất cả dữ liệu có sẵn và phân loại một điểm dữ liệu mới dựa trên sự giống nhau. Điều này có nghĩa là khi dữ liệu mới xuất hiện thì nó có thể dễ dàng được phân loại thành danh mục bộ tốt bằng cách sử dụng thuật toán K-nearest neighbors.

Thuật toán K-nearest neighbors là một thuật toán phi tham số, có nghĩa là nó không đưa ra bất kỳ giả định nào về dữ liệu cơ bản. KNN có thể được sử dụng cho hồi quy cũng như phân loại nhưng chủ yếu nó được sử dụng cho các bài toán Phân loại.

Thuật toán K-nearest neighbors còn được gọi là thuật toán học lười vì nó không học từ tập huấn luyện ngay lập tức thay vào đó nó lưu trữ tập dữ liệu và tại thời điểm phân loại, nó thực hiện một hành động trên tập dữ liệu.

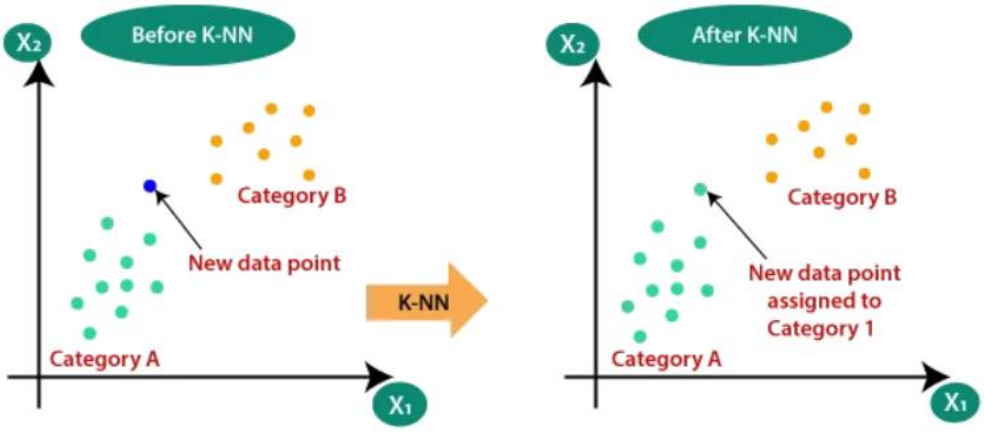
KNN ở giai đoạn huấn luyện chỉ lưu trữ tập dữ liệu và khi nhận được dữ liệu mới, nó sẽ phân loại dữ liệu đó thành một danh mục gần giống với dữ liệu mới.

Giả sử ta có hình ảnh của một sinh vật trông giống mèo và chó nhưng chúng ta muốn biết đó là mèo hay chó. Vì vậy, để nhận dạng này, chúng ta có thể sử dụng thuật toán KNN vì nó hoạt động dựa trên thước đo độ tương tự. Mô hình KNN của chúng tôi sẽ tìm thấy các tính năng tương tự của tập dữ liệu mới với hình ảnh chó và mèo và dựa trên các tính năng giống nhau nhất, nó sẽ xếp nó vào danh mục chó và mèo.

Ảnh 23: KNN Classifier (1)

8.1.2. Tại sao lại cần K-nearest neighbors?

Giả sử có hai loại, tức là Loại A và Loại B, và ta có điểm dữ liệu mới x_1, vậy điểm dữ liệu này sẽ nằm trong loại nào trong số các loại này. Để giải quyết loại vấn đề này thì cần thuật toán K-nearest neighbors. Với sự trợ giúp của KNN, chúng ta có thể dễ dàng xác định danh mục hoặc lớp của một tập dữ liệu cụ thể. Hãy xem xét sơ đồ dưới đây:



Ảnh 24: KNN Classifier (2)

8.2. CÁCH HOẠT ĐỘNG CỦA THUẬT TOÁN K-NEAREST NEIGHBORS

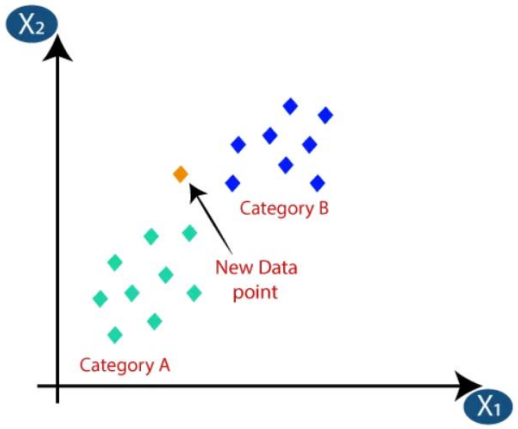
8.2.1. Giải thuật

Hoạt động của KNN có thể được giải thích dựa trên thuật toán dưới đây:

1. Khởi tạo K số hàng xóm ta đã chọn - Chọn số K hàng xóm.

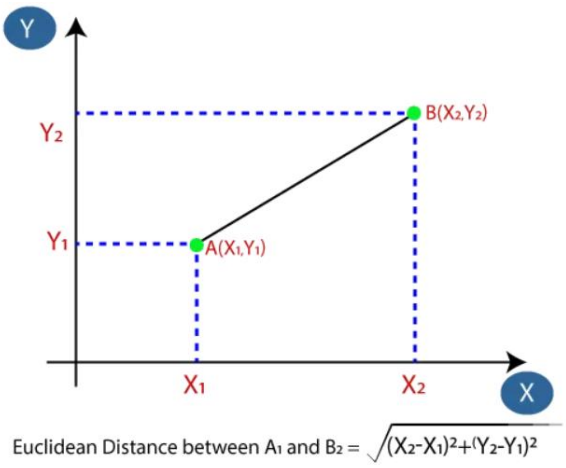
2. Đối với mỗi ví dụ trong dữ liệu - Tính khoảng cách Euclide của K số hàng xóm
    a. Tính khoảng cách giữa ví dụ truy vấn và ví dụ hiện tại từ dữ liệu.
    b. Thêm khoảng cách và chỉ mục của ví dụ vào bộ sưu tập có thứ tự.
3. Sắp xếp tập hợp các khoảng cách và chỉ số theo thứ tự từ nhỏ nhất đến lớn nhất (theo thứ tự tăng dần) theo khoảng cách.
4. Chọn K mục đầu tiên từ bộ sưu tập đã sắp xếp
5. Lấy nhãn của K mục đã chọn
6. Nếu hồi quy, trả về giá trị trung bình của nhãn K
7. Nếu phân loại, trả về chế độ của nhãn K - Gán các điểm dữ liệu mới cho danh mục đó mà số lượng lân cận là tối đa.

Giả sử chúng ta có một điểm dữ liệu mới và chúng ta cần đưa nó vào danh mục bắt buộc.

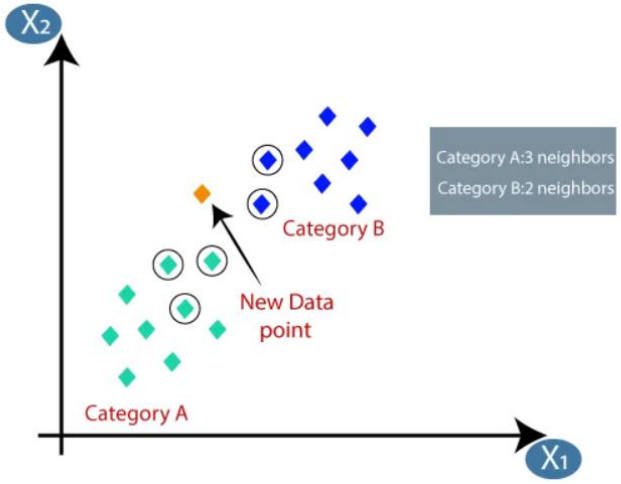


Đầu tiên ta sẽ chọn số hàng xóm nên ta sẽ chọn k=5.

Tiếp theo, ta sẽ tính khoảng cách Euclide giữa các điểm dữ liệu. Khoảng cách Euclide là khoảng cách giữa hai điểm mà chúng ta đã nghiên cứu trong hình học.



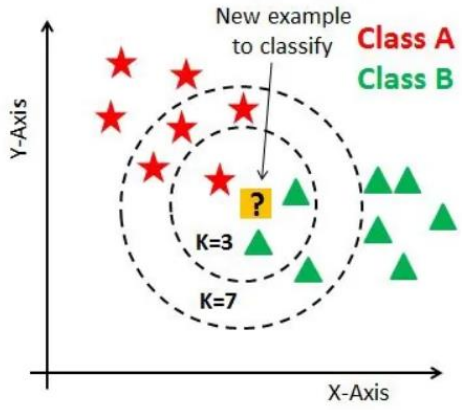
Bằng cách tính khoảng cách Euclide, ta có được những người hàng xóm gần nhất, như ba người hàng xóm gần nhất thuộc loại A và hai người hàng xóm gần nhất thuộc loại B.



Ta có thể thấy 3 người hàng xóm gần nhất thuộc loại A, do đó điểm dữ liệu mới này phải thuộc loại A.

Làm cách nào để chọn giá trị K trong thuật toán K-nearest neighbors?

Giá trị K cho biết số lượng hàng xóm gần nhất. Ta phải tính khoảng cách giữa các điểm kiểm tra và các điểm nhãn đã được huấn luyện. Việc cập nhật số liệu khoảng cách sau mỗi lần lặp lại tốn kém về mặt tính toán và đó là lý do tại sao K-nearest neighbors là một thuật toán lười học.



Ta có thể xác minh từ hình ảnh trên, nếu tiếp tục với K=3 thì dự đoán rằng đầu vào thử nghiệm thuộc về lớp B và nếu tiếp tục với K=7 thì chúng tôi dự đoán rằng đầu vào thử nghiệm đó thuộc về lớp A. Đó là cách ta có thể tưởng tượng rằng giá trị K có tác động mạnh mẽ đến hiệu suất K-nearest neighbors.

8.2.2. Làm thế nào để chọn giá trị K tối ưu?

Trong thuật toán K-nearest neighbors (KNN), tham số K biểu thị số lượng lân cận gần nhất được sử dụng để đưa ra dự đoán cho một điểm kiểm tra nhất định. Việc chọn giá trị K là một bước quan trọng trong thuật toán KNN vì nó có thể ảnh hưởng đến hiệu năng của mô hình.

Có một số cách tiếp cận khác nhau có thể được sử dụng để chọn giá trị k trong thuật toán KNN:

- **Xác thực chéo:** Một cách tiếp cận là sử dụng xác thực chéo để tìm giá trị của k mang lại hiệu suất tốt nhất. Trong xác thực chéo, tập dữ liệu được chia thành tập huấn luyện và tập xác thực. Mô hình được huấn luyện trên tập huấn luyện và được kiểm tra trên tập xác thực cho các giá trị khác nhau của k. Giá trị k mang lại hiệu suất tốt nhất trên bộ xác thực được chọn làm giá trị cuối cùng của k.
- **Quy tắc ngón tay cái:** Một cách tiếp cận khác là sử dụng phương pháp phỏng đoán hoặc "quy tắc ngón tay cái" để chọn giá trị của k. Ví dụ, một heuristic phổ biến là chọn k là căn bậc hai của tổng số điểm trong tập huấn luyện.
- **Tối ưu hóa siêu tham số:** Một cách tiếp cận khác là sử dụng phương pháp tối ưu hóa siêu tham số để tìm kiếm giá trị tối ưu của k. Tối ưu hóa siêu tham số bao gồm việc đào

tạo mô hình với các giá trị k khác nhau và đánh giá hiệu suất của từng mô hình. Giá trị k mang lại hiệu suất tốt nhất được chọn làm giá trị cuối cùng của k.

8.2.3. Tính khoảng cách

Bước đầu tiên là tính khoảng cách giữa điểm mới và từng điểm huấn luyện. Có nhiều phương pháp khác nhau để tính khoảng cách này, trong đó các phương pháp được biết đến phổ biến nhất là - khoảng cách Euclidian, Manhattan (đối với khoảng cách liên tục) và khoảng cách Hamming (đối với khoảng cách phân loại).

Khoảng cách Euclide: Khoảng cách Euclide được tính bằng căn bậc hai của tổng bình phương chênh lệch giữa một điểm mới (x) và một điểm hiện có (y).

$$d = \sqrt{(X_2 - X_2)^2 + (Y_2 - Y_1)^2}$$

Khoảng cách Manhattan: Đây là khoảng cách giữa các vectơ thực bằng cách sử dụng tổng chênh lệch tuyệt đối của chúng.

$$\sum_{i=1}^{k} |x_i - y_i|$$

Khoảng cách Hamming: Nó được sử dụng cho các biến phân loại. Nếu giá trị (x) và giá trị (y) giống nhau thì khoảng cách D sẽ bằng 0. Ngược lại D=1.

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

8.3. ƯU VÀ NHƯỢC ĐIỂM CỦA KNN

Ưu điểm:

- Dễ dàng triển khai
- Có thể hiệu quả hơn nếu dữ liệu huấn luyện lớn.
- Khả năng chống chịu tốt với dữ liệu huấn luyện có nhiễu.

Khuyết điểm:

- Luôn cần xác định giá trị của K, đôi khi có thể phức tạp.

– Chi phí tính toán cao do phải tính khoảng cách giữa các điểm dữ liệu cho tất cả các mẫu huấn luyện.

8.4. THỰC HÀNH KNN VỚI EXCEL

Dữ liệu đầu vào:

Name	Age	Income (1000s)	Cards	Response
N1	71	32	3	No
N2	33	144	8	No
N3	29	163	5	Yes

Dự đoán kết quả cho dữ liệu sau:

	Age	Income	Cards have
Daisy	27	155	5
K	Daisy's likely reponse:		
1			
3			
5			
7			
9			
11			

Đầu tiên chúng ta cần tìm giá trị lớn nhất, nhỏ nhất với mỗi thuộc tính dữ liệu đầu vào.

Công thức:

$$= MAX(X, X1)$$

$$= MIN(X, X1)$$

Trong đó:

– X: vùng dữ liệu cần tìm giá trị lớn nhất nhỏ nhất

– X1: giá trị của dữ liệu cần dự đoán

max-age	75
min-age	18
max-income	200
min-income	10
max-cards	10
min-cards	0

Sau đó, chúng ta sẽ chuẩn hóa dữ liệu đầu vào và dữ liệu dự đoán để biết được giá trị đầu vào tương ứng với bao nhiêu phần trăm của khoảng giá trị nhỏ nhất tới lớn nhất. **Công thức**:

$$= 100/(X1 - X2) * (X3 - X2)$$

Trong đó:

- X1: giá trị lớn nhất của một dãy dữ liệu thuộc tính đầu vào
- X2: giá trị nhỏ nhất của một dãy dữ liệu thuộc tính đầu vào
- X3: giá trị đầu vào

normalized age	normalized income	normalized cards
92.9824561	11.5789474	30
26.3157895	70.5263158	80
19.2982456	80.5263158	50

	Age	Income	Cards have			
Daisy	27	155	5	15.78947	76.3157895	50

Tiếp theo, chúng ta sẽ tính khoảng cách giữa giá trị dự đoán và giá trị đầu vào.

Công thức: $= SQRT((X1 - Z1)\text{^}2 + (X2 - Z2)\text{^}2 + (X3 - Z3)\text{^}2)$

Trong đó:

- X1, X2, X3 là các giá trị đặc trưng của dữ liệu cần dự đoán
- Z1, Z2, Z3 là các giá trị đặc trưng của dữ liệu đầu vào

Sau đó, chúng ta sẽ tính nghịch đảo của khoảng cách và gán trọng số để tăng cường độ chính xác của dự đoán.

Công thức:

- $= 1/(1 + X)$ tính nghịch đảo của khoảng cách X
- $= X1/SUM(X2)$ tính trọng số của một dòng dữ liệu

Cách này giúp làm nổi bật tầm quan trọng của các điểm lân cận trong quá trình dự đoán.

distance	reciprocal	weight
102.71132	0.009642149	0.002365
32.31596091	0.030015643	0.007362
5.480876966	0.154300106	0.037847

Tiếp đó, chúng ta cần phải tính được khoảng cách nhỏ nhất thứ k để loại bỏ các khoảng cách lớn hơn nó.

Công thức:

$$= SMALL(X, X1)$$

Hàm này giúp ta tìm được giá trị nhỏ thứ X1 trong phạm vi dữ liệu X

K	Daisy's likely reponse:	Small
1		2.740438
3		10.61077
5		13.53267
7		21.34303
9		22.89995
11		23.37287

Từ đây chúng ta sẽ phải tính tổng của tất cả các trọng số có khoảng cách nhỏ hơn khoảng cách nhỏ nhất thứ k và có cùng giá trị response.

Công thức: $= SUMIFS(X, Z1, B1, Z2, "<= "\&B2)$

Trong đó:

- X: Phạm vi chứa các giá trị cần tính tổng
- Z1, Z2: Các phạm vi điều kiện, là các cột mà bạn đang kiểm tra điều kiện
- B1: Điều kiện đầu tiên, áp dụng cho phạm vi Z1.
- "<="&B2: Điều kiện thứ hai, yêu cầu các giá trị trong Z2 phải nhỏ hơn hoặc bằng giá trị trong ô B2

Excel sẽ xem xét các ô trong phạm vi Z1 và Z2.

Nếu một ô trong Z1 thỏa mãn điều kiện B1 và một ô tương ứng trong Z2 thỏa mãn điều kiện nhỏ hơn hoặc bằng B2, giá trị tương ứng trong phạm vi X sẽ được cộng vào tổng.

Sau đó, chúng ta chỉ cần tìm giá trị lớn nhất của mỗi loại response với hàm MAX và so sánh xem giá trị đó tương ứng với phân loại response nào rồi điền vào ô tương ứng.

Công thức: $= @INDEX(X, 1, MATCH(Z, B, 0))$

Trong đó:

MATCH(Z, B, 0):

- MATCH tìm vị trí của giá trị Z trong phạm vi B
- Tham số 0 chỉ định MATCH sẽ tìm kiếm giá trị chính xác
- Kết quả của MATCH là chỉ số cột trong X tương ứng với giá trị Z trong B

INDEX(X, 1, ...):

- INDEX trả về giá trị tại một vị trí nhất định trong phạm vi X
- 1 là hàng đầu tiên trong phạm vi X
- Cột của giá trị này được xác định bởi kết quả của hàm MATCH

Yes	Not Sure	No	Max Weight
0	0.06557647	0	0.065576473
0.05897309	0.06557647	0	0.065576473
0.09573298	0.06557647	0	0.095732981
0.10671112	0.07747279	0	0.106711116
0.1169741	0.08783326	0	0.1169741
0.1169741	0.09803278	0.01006384	0.1169741

	Age	Income	Cards have
Daisy	27	155	5
K	Daisy's likely reponse:		
1	Not Sure		
3	Not Sure		
5	Yes		
7	Yes		
9	Yes		
11	Yes		

8.5. THỰC HÀNH CÀI ĐẶT KNN VỚI CODE PYTHON

8.5.1. Dataset

Download full dataset tại:
https://github.com/NT02IT/BasicMachineLearning/blob/main/datasets/knn/bank.csv

Dataset chứa thông tin nhân khẩu học của các khách hàng gồm các trường: age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome và quyết định cho vay của ngân hàng.

Dataset gồm 16 features, 4521 dòng dữ liệu.

8.5.2. Xây dựng mô hình bằng code Python

8.5.2.1. Sử dụng thư viện Sklearn

Bước 1: Tạo class UseSklearn, trong hàm khởi tạo viết code để đọc dataframe từ URL csv nhận được sau đó chia ra thành các tập *X_train, y_train, X_test, y_test*

```
class UseSklearn:
  def __init__(self, datasetURL, train_size=0.5):
    csv_handler = CSVHandler(datasetURL)
    self.dataframe = csv_handler.read_csv()

    self.dataframe, self.label_encoders = Normalization.encode_dataframe(self.dataframe)

    X = self.dataframe.iloc[:, :-1]   # Chọn tất cả các hàng và cột trừ cột cuối cùng
    y = self.dataframe.iloc[:, -1]    # Chọn cột cuối cùng

    # Chia dữ liệu làm 2 phần training và testing
    split_index = int(len(self.dataframe) * train_size)
    self.X_train = X.iloc[:split_index]
    self.y_train = y.iloc[:split_index]
    self.X_test = X.iloc[split_index:]
    self.y_test = y.iloc[split_index:]
```

Bước 2: Tiếp tục viết phương thức training để khởi tạo model và tìm ra K tối ưu bằng cách cho model huấn luyện với k nằm trong khoảng *k_min* đến *k_max* được lấy từ tham số đầu vào.

```
def train(self, min_k=1, max_k=30):
    self.mse_values = []
    self.k_values = range(min_k, max_k + 1)
```

```
    for k in self.k_values:
        print(f"\rTraining with k = {k}...", end='', flush=True)
        # Tạo model KNN với k láng giềng
        self.model = KNeighborsClassifier(n_neighbors=k)
        self.model.fit(self.X_train, self.y_train)

        # Dự đoán trên tập test
        y_pred = self.model.predict(self.X_test)

        # Tính MSE và lưu lại
        mse = mean_squared_error(self.y_test, y_pred)
        self.mse_values.append(mse)

    # Tìm giá trị k tối ưu
    optimal_k = self.k_values[self.mse_values.index(min(self.mse_values))]
    print(f'\r\nOptimal k: {optimal_k}, Minimum MSE: {min(self.mse_values):.4f}', end='',
flush=True)
    self.model = KNeighborsClassifier(optimal_k)
    self.model.fit(self.X_train, self.y_train)

    return self.mse_values, self.k_values
```

Bước 3: Viết phương thức predict để dự đoán cho tập dữ liệu nhập vào:

```
def predict(self, input_data):
    return self.model.predict(input_data)
```

Bước 4: Viết phương thức test để kiểm tra độ chính xác của model:

```
def test(self):
    y_pred = self.predict(self.X_test)

    validateNoLib = ValidateNoLib(self.y_test, y_pred)
    print("Số lượng mẫu test:", validateNoLib.getSampleSize())
    print("Các phân lớp:", validateNoLib.getSampleClasses())
    print("Ma trận nhầm lẫn:\n", validateNoLib.confusionMatrix())
    print(f"Độ chính xác: {round(validateNoLib.accuracy()*100,2)}%")
```

Bước 5: Vào file main khởi tạo model và gọi lần lượt các phương thức để xem kết quả trên màn hình console.

8.5.2.2. Không sử dụng thư viện

Bước 1: Tạo class *WithNoLib*, trong hàm khởi tạo viết tương tự hàm khởi tạo khi sử dụng Sklearn

Bước 2: Viết hàm *euclidean_distance* để tính khoảng cách euclide giữa 2 điểm

```
def euclidean_distance(self, pointA, pointB):
    pointA = np.array(pointA)
    pointB = np.array(pointB)
    return np.linalg.norm(pointA - pointB)
```

Có thể tính theo công thức không thông qua thư viện numpy nhưng tốc độ xử lý sẽ chậm hơn do phải chạy vòng lặp.

```
def euclidean_distance(self, pointA, pointB):
    tmp = 0
    for i in range(len(pointA)):
      tmp += (float(pointA.iloc[i]) - float(pointB.iloc[i])) ** 2
    return math.sqrt(tmp)
```

Bước 3: Viết hàm predict để dự đoán kết quả với 1 giá trị K bằng cách tính tất cả khoảng cách của mỗi điểm trong tập X_test đến X_train, sau đó chọn ra K hàng xóm gần nhất và lấy nhãn phổ biến nhất trong K hàng xóm này gán cho mỗi X_test.

```
def predict(self, X_train, y_train, X_test, k):
    X_train_np = X_train.to_numpy()
    X_test_np = X_test.to_numpy()
    y_train_np = y_train.to_numpy()

    # Tính toán ma trận khoảng cách Euclidean giữa X_test và X_train
    distances = cdist(X_test_np, X_train_np, metric='euclidean')

    y_pred = []
    for i, dist in enumerate(distances):
      print(f"\rPredict for test point {i}...", end='', flush=True)

      # Lấy k chỉ số hàng xóm gần nhất
      neighbors_idx = np.argsort(dist)[:k]
      neighbors_labels = y_train_np[neighbors_idx]

      # Lấy nhãn phổ biến nhất trong k hàng xóm
      predicted_label = Counter(neighbors_labels).most_common(1)[0][0]
      y_pred.append(predicted_label)
    return np.array(y_pred)
```

Bước 4: Viết hàm train_with_k để gọi hàm predict xử lý và trả về y dự đoán, sau đó hàm này sẽ tính mse giữa giá trị dự đoán và giá trị thực tế để lưu giữ lại sau mỗi lần tính với K thay đổi.

```
def train_with_k(self, k, min_k):
```

```
    print(f"\rTraining with k = {k}...", end='', flush=True)
    y_pred = self.predict(self.X_train, self.y_train, self.X_test, k)
    mse = np.mean((self.y_test - y_pred) ** 2)

    self.mse_values[k - min_k] = mse
```

Bước 5: Viết hàm *train()* để chạy vòng lặp các giá trị K từ k_min đến k_max được nhập vào sau đó tìm ra giá trị K tốt nhất với tiêu chí mse ứng với K đó là nhỏ nhất.

```
def train(self, min_k=1, max_k=10):
    self.mse_values = [None] * (max_k - min_k + 1)  # Khởi tạo danh sách chứa MSE cho mỗi k
    self.k_values = range(min_k, max_k + 1)

    for k in self.k_values:
      self.train_with_k(k, min_k)

    # Tìm giá trị k tối ưu
    if None not in self.mse_values:
      self.optimal_k = self.k_values[np.argmin(self.mse_values)]
      print(f'\r\nOptimal k: {self.optimal_k}, Minimum MSE: {min(self.mse_values):.4f}', end='',
flush=True)

    return self.mse_values, self.k_values
```

Có thể sử dụng đa luồng để tăng tốc thuật toán xử lý với dữ liệu lớn

```
def train(self, min_k=1, max_k=10):
    self.mse_values = [None] * (max_k - min_k + 1)  # Khởi tạo danh sách chứa MSE cho mỗi k
    self.k_values = range(min_k, max_k + 1)

    threads = []

    # Tạo các luồng và bắt đầu chúng
    for k in self.k_values:
      thread = threading.Thread(target=self.train_with_k, args=(k, min_k))
      threads.append(thread)
      thread.start()

    # Đợi tất cả các luồng kết thúc
    for thread in threads:
      thread.join()

    # Tìm giá trị k tối ưu
    if None not in self.mse_values:
      self.optimal_k = self.k_values[np.argmin(self.mse_values)]
      print(f'\r\nOptimal k: {self.optimal_k}, Minimum MSE: {min(self.mse_values):.4f}', end='',
flush=True)
```

```
    return self.mse_values, self.k_values
```

Bước 6: Viết hàm *test()* để kiểm tra độ chính xác của mô hình

```
def test(self):
    y_pred = self.predict(self.X_train, self.y_train, self.X_test, self.optimal_k)

    validateNoLib = ValidateNoLib(self.y_test, y_pred)
    print("\rSố lượng mẫu test:", validateNoLib.getSampleSize(), end='', flush=True)
    print("Các phân lớp:", validateNoLib.getSampleClasses())
    print("Ma trận nhầm lẫn:\n", validateNoLib.confusionMatrix())
    print(f"Độ chính xác: {round(validateNoLib.accuracy()*100,2)}%")
```

Bước 7: Vào file main khởi tạo model và gọi lần lượt các hàm để xem kết quả

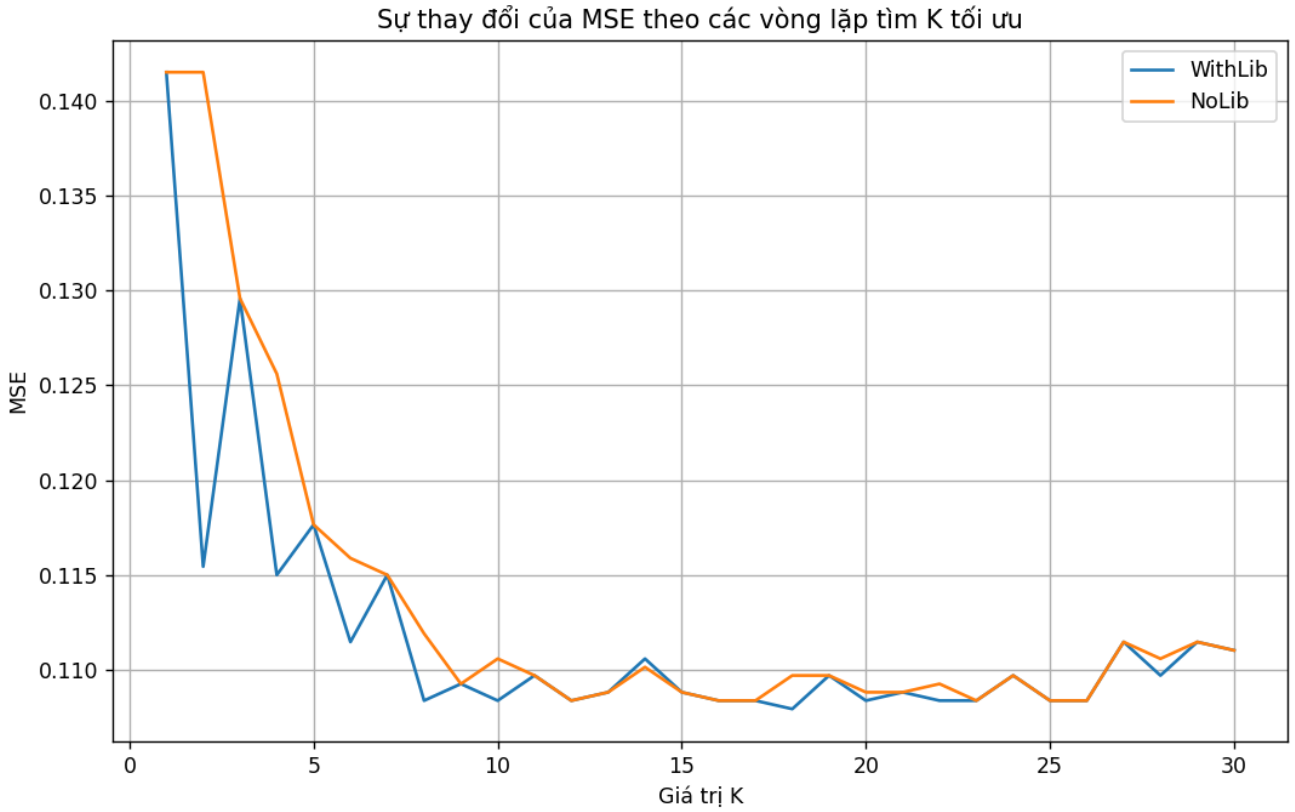
8.6. SO SÁNH CÁC CÁCH CÀI ĐẶT KNN

8.6.1. Tổng quan

Thực hiện 2 thuật toán cùng trên 1 dataset gồm 16 features, 4521 dòng dữ liệu, so sánh được thực hiện trên các điều kiện môi trường như nhau:

- Thực hiện trên cùng 1 máy tính, cùng môi trường python
- Dùng 50% số dòng (2261 dòng) để huấn luyện và 50% còn lại để test.
- k_min = 1, k_max = 30
- Thời gian được so sánh thông qua thời gian thực hiện hàm *train()*

Biểu đồ so sánh sự thay đổi của MSE theo các lần lặp tìm K tối ưu giữa 2 phương pháp sử dụng thư viện Sklearn và không sử dụng thư viện:



Bảng so sánh hiệu suất cả các mô hình

Tiêu Chí	Use Sklearn	No lib
K	18	12
MSE	0.1079	0.1084
Thời gian chạy	4.273397s	5.377669s
Độ chính xác	89.21%	89.16%

8.6.2. Nhận xét

Về độ chính xác: Cách không sử dụng thư viện tìm được k=12, với MSE tối thiểu là 0.1084, kém hơn một chút so với Sklearn (0.1079). Điều này có thể do thuật toán không được tối ưu hoặc thiếu một số tinh chỉnh.

Thời gian: Cách không sử dụng thư viện chạy với 4.306 giây khi được chạy đa luồng, gần như tương đương với Sklearn, tuy nhiên nếu khi chưa chạy đa luồng thì khi không sử dụng thư viện chạy lâu hơn nhiều so với sử dụng Sklearn.

CHƯƠNG IX: K-MEAN CLUSTERING

9.1. TỔNG QUAN

K-means Clustering là một thuật toán phân cụm (clustering) phổ biến được dùng để chia một tập dữ liệu thành K cụm (clusters), dựa trên khoảng cách giữa các điểm dữ liệu. Thuật toán này thuộc nhóm học không giám sát (unsupervised learning), có nghĩa là nó không cần nhãn phân lớp để thực hiện việc phân nhóm.

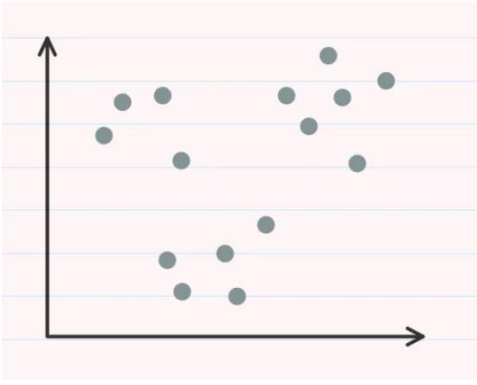
9.2. CÁCH HOẠT ĐỘNG CỦA THUẬT TOÁN K-MEANS CLUSTERING

Thuật toán K-Means là một quá trình lặp lại sử dụng một loạt các bước đơn giản, tuần tự để đạt được mục tiêu chính là phân cụm dữ liệu. Các giai đoạn chính:

(1) Chọn số cụm (K)

Đây là chữ 'K' trong K-Means. K đại diện cho số lượng cụm bạn muốn tạo. Bạn có thể chọn K dựa trên sự hiểu biết của bạn về dữ liệu hoặc bằng cách sử dụng các phương pháp như phương pháp khuỷu tay, bao gồm việc tìm số cụm tối ưu để giảm sự biến đổi trong cụm.

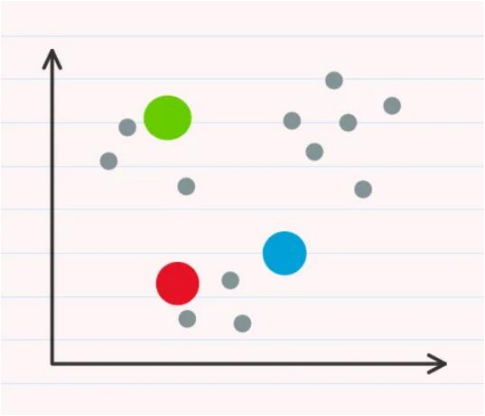
Để minh họa quy trình phân cụm K-Means, hãy xem xét một biểu đồ có nhiều điểm dữ liệu khác nhau, trong đó mỗi điểm biểu thị một quan sát hoặc trường hợp riêng lẻ trong tập dữ liệu của chúng tôi. Trong ví dụ này, ta sẽ giả định rằng chúng tôi muốn tạo 3 cụm từ các điểm dữ liệu này.



(2) Chọn ngẫu nhiên K điểm dữ liệu riêng biệt

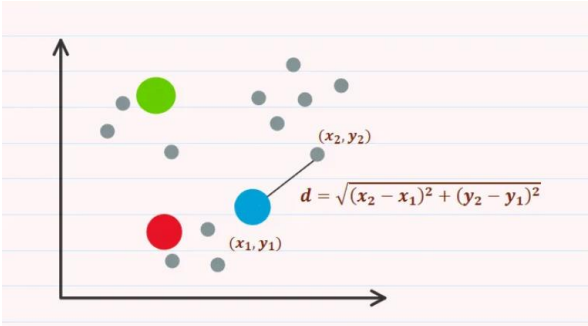
Khi đã xác định giá trị của K, thuật toán K-Means sẽ chọn ngẫu nhiên K điểm dữ liệu riêng biệt từ tập dữ liệu để làm trọng tâm của cụm ban đầu. Trọng tâm là một điểm tưởng tượng tượng

trưng cho tâm của một cụm. Thuật toán sẽ cập nhật lặp đi lặp lại vị trí của các trọng tâm này để giảm thiểu khoảng cách tổng thể giữa các điểm dữ liệu và trọng tâm được chỉ định của chúng.



(3) Đo khoảng cách giữa mỗi điểm dữ liệu và K trọng tâm ban đầu

Đối với mỗi điểm dữ liệu trong tập dữ liệu, thuật toán sẽ tính khoảng cách giữa điểm dữ liệu và từng K centroid. Thước đo khoảng cách thường được sử dụng là khoảng cách Euclide (d), đo khoảng cách đường thẳng giữa hai điểm.

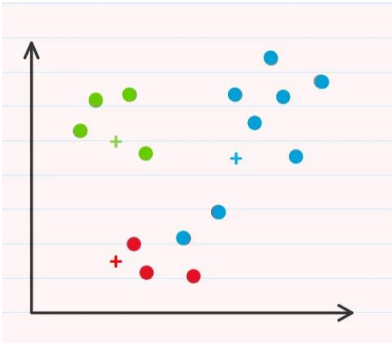


(4) Gán từng điểm dữ liệu vào cụm gần nhất

Sau khi tính toán khoảng cách, thuật toán gán từng điểm dữ liệu vào cụm có trọng tâm gần nhất. Điều này có nghĩa là mỗi điểm dữ liệu được liên kết với cụm có trọng tâm gần điểm dữ liệu nhất.

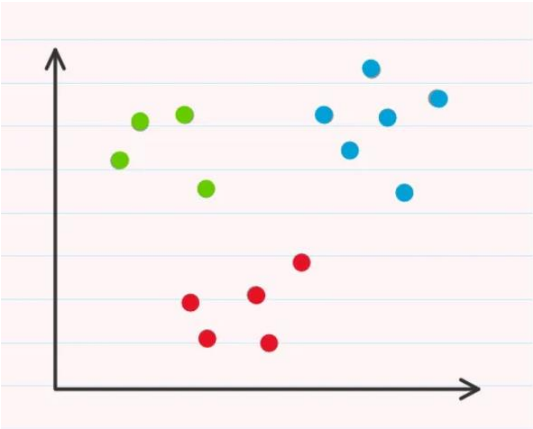
(5) Tính toán trọng tâm mới hoặc giá trị trung bình của mỗi cụm

Khi tất cả các điểm dữ liệu đã được gán cho một cụm, thuật toán sẽ tính toán các trọng tâm mới cho mỗi cụm. Trọng tâm mới là giá trị trung bình (hoặc trung bình) của tất cả các điểm dữ liệu trong cụm đó.



(6) Lặp lại các bước 3–5 cho đến khi hội tụ

Sau đó, thuật toán lặp lại quá trình gán điểm dữ liệu cho các cụm gần nhất và tính toán lại trọng tâm. Quá trình lặp lại này tiếp tục cho đến khi việc gán cụm không còn thay đổi đáng kể giữa các lần lặp hoặc đạt đến số lần lặp được chỉ định. Điều này được gọi là sự hội tụ của thuật toán.



9.3. ƯU ĐIỂM VÀ NHƯỢC ĐIỂM CỦA K-MEANS

Ưu điểm:

- K-Means dễ hiểu và dễ triển khai, khiến nó trở thành lựa chọn phổ biến cho các nhiệm vụ phân cụm.
- Thuật toán có thể xử lý các tập dữ liệu lớn một cách hiệu quả và mở rộng quy mô tốt cho số lượng cụm tương đối nhỏ.
- K-Means không được giám sát, nghĩa là nó có thể khám phá các mẫu trong dữ liệu chưa được gắn nhãn và có thể áp dụng trên nhiều miền khác nhau.

Nhược điểm:

- Hiệu suất của thuật toán có thể bị ảnh hưởng bởi việc lựa chọn trọng tâm cụm ban đầu, điều này có thể dẫn đến kết quả phân cụm dưới mức tối ưu.
- K-Means có thể gặp khó khăn với dữ liệu nhiễu và các dữ liệu ngoại lệ, vì chúng có thể làm biến dạng tâm và ranh giới của cụm.
- K-Means giả định rằng các cụm có dạng hình cầu lồi, có thể không phù hợp với các tập dữ liệu có cụm không lồi hoặc có hình dạng tùy ý.

CHƯƠNG X: KẾT LUẬN

Trong quá trình tìm hiểu và nghiên cứu cơ bản về Machine Learning, nhóm đã có cơ hội nắm bắt rõ hơn về các thuật toán nền tảng trong lĩnh vực này và lợi ích mà Machine Learning mang lại trong việc giải quyết các bài toán phức tạp.

Machine Learning không chỉ là một lĩnh vực khoa học dữ liệu hiện đại mà còn là nền tảng cho các ứng dụng thông minh, tự động hóa và dự đoán hiệu quả. Nhóm đã tìm hiểu các thuật toán quan trọng như:

- **Linear Regression (Hồi quy tuyến tính):** Một kỹ thuật học máy cơ bản, được sử dụng để dự đoán các giá trị liên tục dựa trên mối quan hệ tuyến tính giữa các biến.
- **Logistic Regression (Hồi quy logistic):** Thích hợp cho các bài toán phân loại nhị phân, giúp dự đoán khả năng thuộc về một trong hai nhóm dữ liệu.
- **Naive Bayes:** Một thuật toán xác suất dựa trên Định lý Bayes, hiệu quả trong việc xử lý phân loại văn bản và phân tích cảm xúc.
- **Neural Network (Mạng nơ-ron):** Mô phỏng cách hoạt động của não bộ, mạnh mẽ trong việc nhận diện mẫu, xử lý hình ảnh và ngôn ngữ tự nhiên.
- **Frequent Itemset Mining:** Kỹ thuật khai phá dữ liệu để tìm các mẫu phổ biến trong tập dữ liệu lớn, đặc biệt hữu ích trong bài toán phân tích transactions.
- **KNN (K-Nearest Neighbors):** Thuật toán phân loại hoặc hồi quy dựa trên khoảng cách, xác định nhãn của một điểm dữ liệu dựa vào nhãn của k điểm lân cận gần nhất trong không gian đặc trưng.
- **KMeans:** Thuật toán phân cụm chia dữ liệu thành k nhóm dựa trên sự tương đồng, bằng cách tối thiểu hóa khoảng cách giữa các điểm trong một cụm với trung tâm cụm (centroid).

Ngoài ra, nhóm đã nghiên cứu về **Data Mining (khai phá dữ liệu)** – một bước quan trọng trong quy trình Machine Learning, nhằm trích xuất thông tin có ý nghĩa từ dữ liệu lớn, không cấu trúc.

Trong quá trình nghiên cứu, nhóm nhận thấy việc sử dụng các thư viện hỗ trợ như Sklearn đã góp phần đơn giản hóa việc triển khai các mô hình Machine Learning. Đồng thời, sự hỗ trợ từ cộng đồng lớn và các tài liệu phong phú giúp nhóm nhanh chóng giải quyết những vướng mắc trong học thuật và thực hành.

Tổng kết, Machine Learning không chỉ là một lĩnh vực tiềm năng mà còn mang lại cơ hội phát triển đột phá cho các ứng dụng thực tế, từ kinh doanh đến công nghệ. Việc nghiên cứu cơ bản về Machine Learning không chỉ là bước đầu xây dựng nền tảng kiến thức mà còn là một hành trang cần thiết để giải quyết các thách thức trong tương lai.

TÀI LIỆU THAM KHẢO

[1] Sách Machine Learning cơ bản – tác giả Vũ Hữu Tiệp – machinelearningcoban.com

[2] Sách Deep Learning cơ bản – tác giả Nguyễn Thanh Tuấn – nttuan8.com