ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỌC MÁY TÍNH

KHAI THÁC DỮ LIỆU VÀ ỨNG DỤNG – CS313.N22

Bài tập 4.1: *Sử dụng ngôn ngữ Python triển khai bài toán phân lớp với các thuật toán cơ bản*

Nhóm 3:

Bùi Nguyễn Anh Trung (NT)	20520332
Nguyễn Trần Minh Anh	20520394
Nguyễn Văn Đức Ngọc	20521666
Lê Nguyễn Bảo Hân	20520174
Hồ Thanh Tịnh	20520813

Hồ Chí Minh, 12 tháng 04 năm 2023

Nội dung

Bảng phân công

Thành viên / Công Việc	Bùi Nguyễn Anh Trung	Hồ Thanh Tịnh	Nguyễn Trần Minh Anh	Lê Nguyễn Bảo Hân	Nguyễn Văn Đức Ngọc
Phân công, quản lý công việc chung			✓		
Tìm hiểu thuật toán Decision Tree		✓			
Tìm hiểu thuật toán Naïve Bayes	✓				
Tìm hiểu thuật toán KNN			✓		
Tìm hiểu thuật toán Logistic Regression					✓
Tìm hiểu GridSearch và cân bằng dữ liệu				✓	
Tổng hợp nội dung và định dạng báo cáo	✓	✓	✓	✓	✓
Demo			✓		
Làm slide	✓	✓	✓	✓	✓
Mức độ hoàn thiện (%)	100%	100%	100%	100%	100%

1. Các thuật toán phân lớp

1.1. Thuật toán Decision Tree

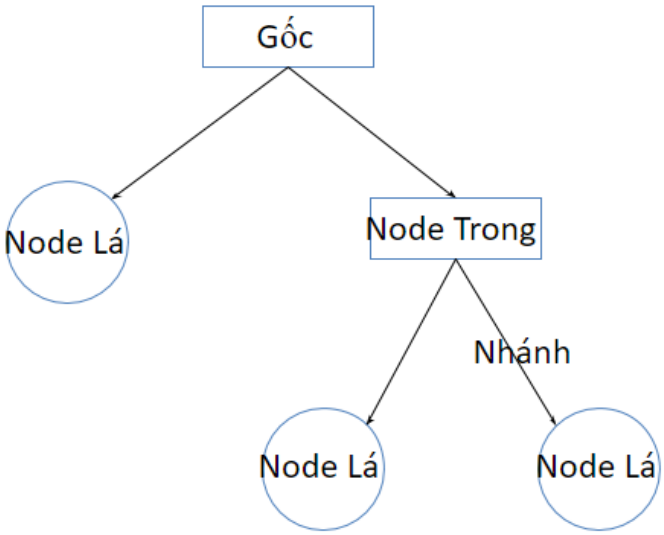
Cây quyết định được phát triển trong nhiều lĩnh vực và có nguồn gốc từ nhiều nhà khoa học và nghiên cứu khác nhau. Tuy nhiên, có thể kể đến *Ronald A. Fisher, J. C. R. Licklider* và *Ross Quinlan* là những người có đóng góp đáng kể trong việc phát triển cây quyết định.

1.1.1. Tổng quan

Cây quyết định là một phương pháp phân tích dữ liệu, là một trong những mô hình học máy đơn giản nhưng rất hiệu quả trong nhiều bài toán phân loại (Classification) và hồi quy (Regression) giá trị của các đối tượng dựa trên các thuộc tính của chúng.

Nó được biểu diễn dưới dạng một cây phân cấp, trong đó:

- Mỗi nút trên cây đại diện cho một thuộc tính
- Các nhánh của nút đại diện cho các giá trị có thể có của thuộc tính đó.
- Các quyết định cuối cùng được đưa ra trên các nút lá của cây.



Hình 1.1. Mô hình cây quyết định

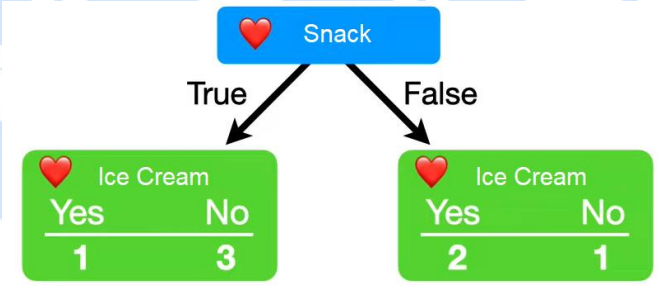
1.1.2. Ý tưởng và cách hoạt động

Quá trình xây dựng cây quyết định bao gồm tìm cách phân chia các dữ liệu thành các nhóm sao cho:

- Các đối tượng trong cùng một nhóm có cùng một thuộc tính.
- Các đối tượng khác nhau có thuộc tính khác nhau.
- Các thuộc tính của đối tượng có thể thuộc các kiểu dữ liệu khác nhau như Nhị phân, Định danh, Thứ tự và Số lượng.
  Tuy nhiên, thuộc tính phân lớp phải có kiểu dữ liệu là Nhị phân hoặc Thứ tự.

Để sử dụng cây quyết định, chúng ta cần tìm ra một cây quyết định dự báo tốt trên tập huấn luyện và sử dụng cây quyết định này để dự đoán lớp của các dữ liệu chưa biết.
Với dữ liệu về các đối tượng gồm các thuộc tính cùng với lớp của nó, cây quyết định sẽ sinh ra các luật để dự đoán lớp của các dữ liệu chưa biết.

Ví dụ:



Hình 1.2 Mô hình cây quyết định

Theo mô hình trên, ta thấy: Dựa trên một người thích ăn **Snack** thì thường không thích ăn **Kem** và ngược lại.

1.1.2.1. Thuật toán Iterative Dichotomiser 3 (ID3)

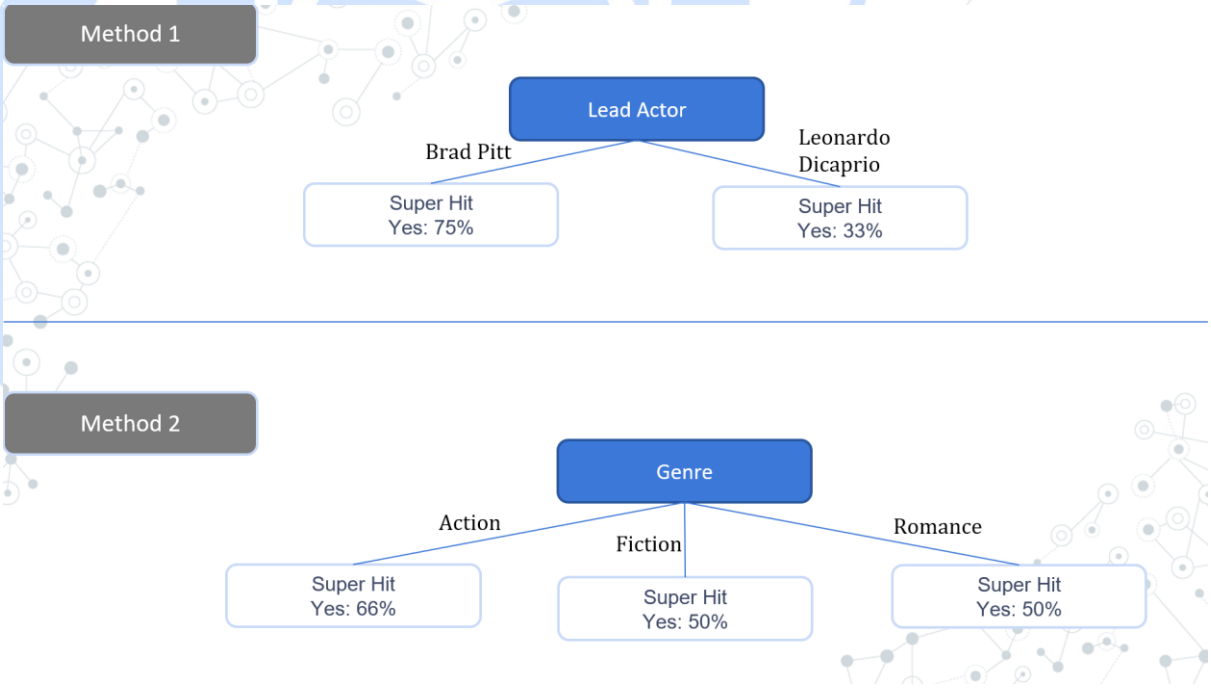
ID3 (J. R. Quinlan 1993) sử dụng phương pháp tham lam tìm kiếm từ trên xuống thông qua không gian của các nhánh có thể không có backtracking. ID3 sử dụng *Entropy* và *Information Gain* để xây dựng một cây quyết định.

Ta xét ví dụ 2:

Lead Actor	**Genre**	**Hit(Y/N)**
Brad Pitt	Action	Yes
Brad Pitt	Fiction	Yes
Brad Pitt	Romance	No
Brad Pitt	Action	Yes
Leonardo Dicaprio	*Action*	*No*
Leonardo Dicaprio	*Fiction*	*No*
Leonardo Dicaprio	*Romance*	*Yes*

Xem xét sự thành công của một bộ phim thông qua hai yếu tố: *diễn viên chính* của phim và *thể loại* phim.

Giả sử, muốn xác định độ thành công của bộ phim chỉ trên 1 yếu tố, ta có hai cách thực hiện sau: qua *diễn viên chính* của phim và qua *thể loại phim*.



Hình 2. Sơ đồ độ thành công của phim qua hai yếu tố diễn viên chính và thể loại

Qua sơ đồ, ta có thể thấy rõ ràng ràng, với phương pháp thứ nhất, ta phân loại được rõ ràng, trong khi phương pháp thứ hai, ta có một kết quả lộn xộn

hơn. Và tương tự, cây quyết định sẽ thực hiện như trên khi thực hiện việc chọn các biến.

Có rất nhiều hệ số khác nhau mà phương pháp cây quyết định sử dụng để phân chia. Hai hệ số phổ biến là ***Information Gain*** và **Gain Ratio** (ngoài ra còn hệ số Gini).

1.1.2.1.1. Entropy trong Decision Tree

Entropy là thuật ngữ thuộc Nhiệt động lực học, là thước đo của sự biến đổi, hỗn loạn hoặc ngẫu nhiên. Entropy càng thấp, tức là tập dữ liệu càng có tính chất xác định và dễ dàng phân loại, còn entropy càng cao thì tập dữ liệu càng có tính không xác định và khó phân loại.

Năm 1948, Shannon đã mở rộng khái niệm Entropy sang lĩnh vực nghiên cứu, thống kê với công thức như sau:

Với một phân phối xác suất của một biến rời rạc x có thể nhận n giá trị khác nhau $x_1, x_2, \ldots, x_n$.

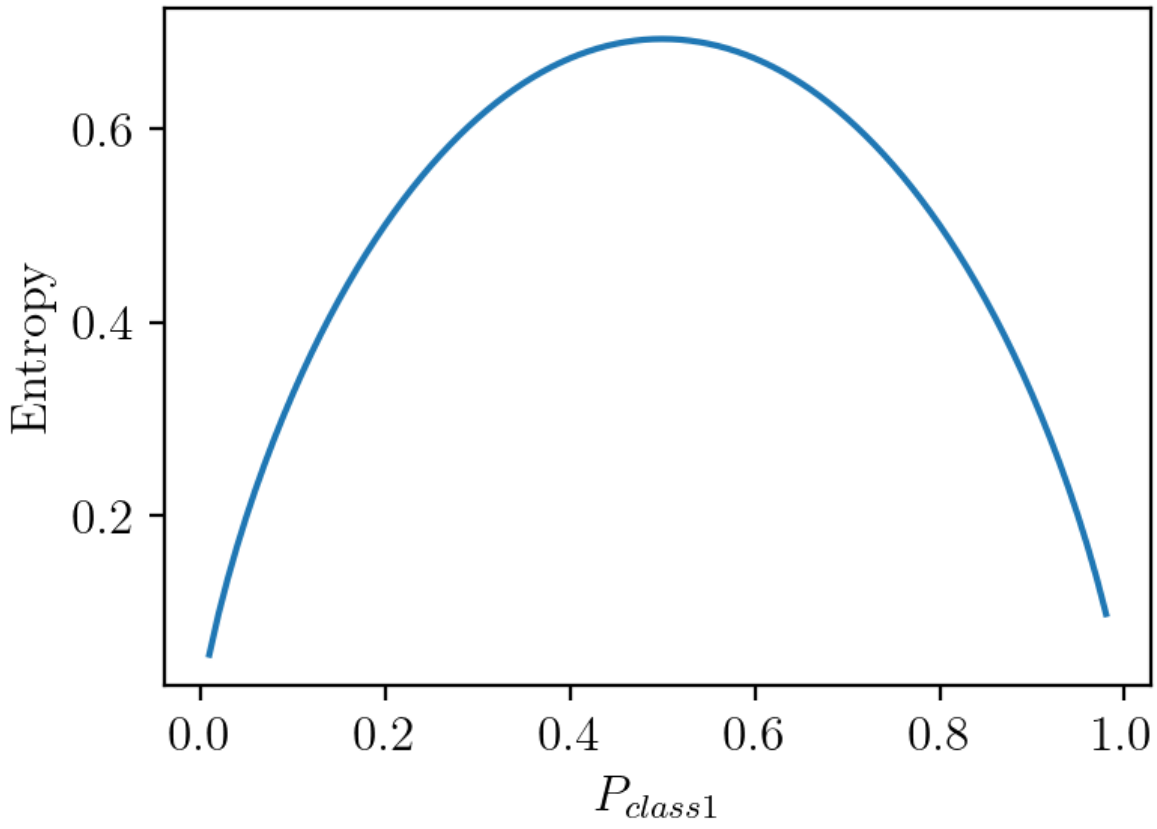
Giả sử rằng xác suất để x nhận các giá trị này là $p_i = p(x = x_i)$.

Ký hiệu phân phối này là $p = (p_1, p_2, \ldots, p_n)$. Entropy của phân phối này được định nghĩa là:

$$H(p) = -\sum_{n=1}^{n} p_i \log(p_i)$$

Giả sử bạn tung một đồng xu, entropy sẽ được tính như sau:

$$H = -[0.5\ ln(0.5) + 0.5\ ln(0.5)]$$



Hình 3. Phân phối Entropy

Hình vẽ trên biểu diễn sự thay đổi của hàm entropy. Ta có thể thấy rằng, entropy đạt tối đa khi xác suất xảy ra của hai lớp bằng nhau.

- P tinh khiết: $p_i = 0$ hoặc $p_i = 1$
- P vẩn đục: $p_i = 0.5$, khi đó hàm Entropy đạt đỉnh cao nhất

1.1.2.1.2. Information Gain trong Decision Tree

Information Gain dựa trên sự giảm của hàm Entropy khi tập dữ liệu được phân chia trên một thuộc tính. Để xây dựng một cây quyết định, ta phải tìm tất cả thuộc tính trả về Infomation Gain cao nhất.

Để xác định các nút trong mô hình cây quyết định, ta thực hiện tính Infomation Gain tại mỗi nút theo trình tự sau:

- **Bước 1**: Tính toán hệ số Entropy của biến mục tiêu S có N phần tử với N_c phần tử thuộc lớp c cho trước:

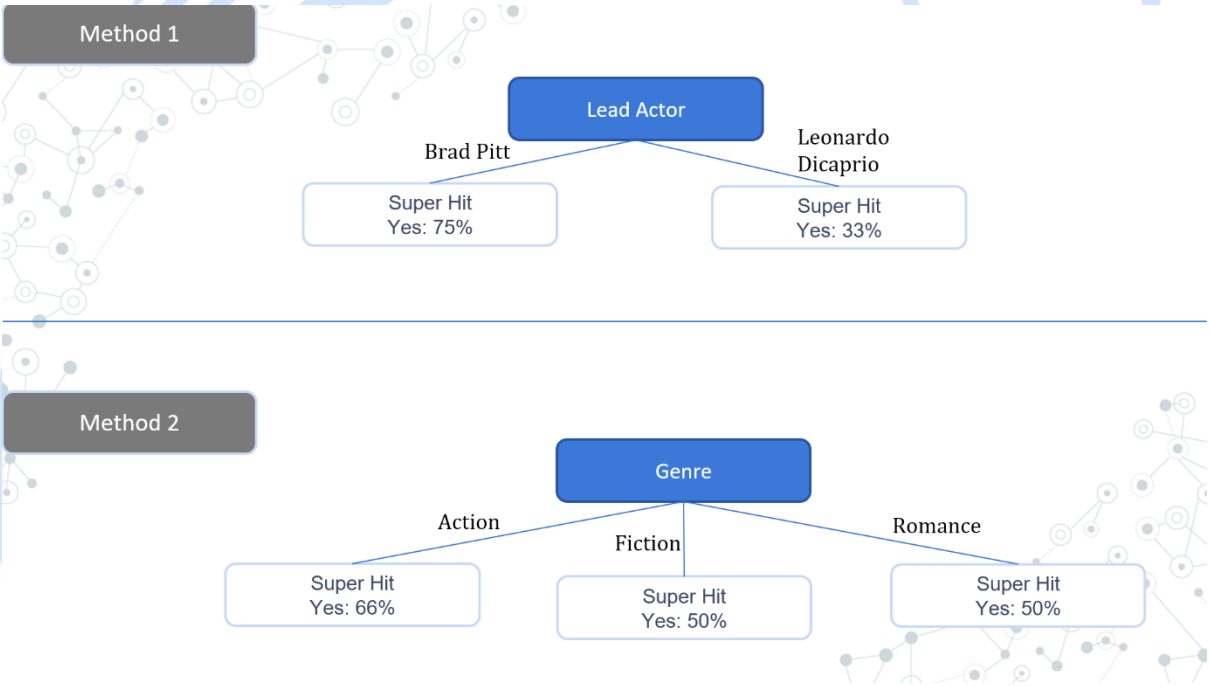
$$H(S) = -\sum_{c=1}^{c} \left(\frac{N_c}{N}\right) log\left(\frac{N_c}{N}\right)$$

- **Bước 2**: Tính hàm số Entropy tại mỗi thuộc tính: với thuộc tính x, các điểm dữ liệu trong S được chia ra K child node $S_1, S_2, ..., S_K$ với số điểm trong mỗi child node lần lượt là $m_1, m_2, ..., m_K$, ta có:

$$H(x, S) = \sum_{k=1}^{k} \left(\frac{m_k}{N}\right) * H(S_k)$$

- **Bước 3**: Chỉ số Gain Information được tính bằng:

$$\mathbf{G(x, S) = H(S) - H(x,S)}$$



Với ví dụ 2 trên, ta tính được hệ số Entropy như sau:

$$Entropy_{Parent} = -(0.57*ln(0.57) + 0.43*ln(0.43)) = 0.68$$

Hệ số Entropy theo phương pháp thứ nhất, chia theo diễn viên chính:

$$Entropy_{left} = -(.75*ln(0.75) + 0.25*ln(0.25)) = 0.56$$

$$Entropy_{right} = -(.33*ln(0.33) + 0.67*ln(0.67)) = 0.63$$

Ta có thể tính hệ số ***Information Gain*** như sau:

$$Information\ Gain = 0.68 - (4*0.56 + 3*0.63)/7 = 0.09$$

Hệ số Entropy với phương pháp thứ hai, chia theo thể loại phim:

$$\underline{\text{Entropy}}_{left} = -(.67*ln(0.67) + 0.33*ln(0.33)) = 0.63$$

$$Entropy_{middle} = -(.5*ln(0.5) + 0.5*ln(0.5)) = 0.69$$

$$Entropy_{right} = -(.5*ln(0.5) + 0.5*ln(0.5)) = 0.69$$

Hệ số ***Information Gain***:

$$Information\ Gain = 0.68 - (3*0.63 + 2*0.69 + 2*0.69)/7 = 0.02$$

So sánh kết quả, ta thấy nếu chia theo phương pháp 1 thì ta được giá trị hệ số Information Gain lớn hơn gấp 4 lần so với phương pháp 2. Như vậy, giá trị thông tin ta thu được theo phương pháp 1 cũng nhiều hơn phương pháp 2.

1.1.2.2. Thuật toán C4.5

Thuật toán C4.5 là thuật toán cải tiến của ID3.

Trong thuật toán ID3, Information Gain được sử dụng làm độ đo. Tuy nhiên, phương pháp này lại ưu tiên những thuộc tính có số lượng lớn các giá trị mà ít xét tới những giá trị nhỏ hơn. Do vậy, để khắc phục nhược điểm trên, ta sử dụng độ đo Gain Ratio (trong thuật toán C4.5) như sau:

Đầu tiên, ta chuẩn hoá information gain với trị thông tin phân tách (split information):

$$Gain\ Ratio = \frac{Information\ Gain}{Split\ Info}$$

Trong đó: Split Info được tính như sau:

$$-\sum_{i=1}^{n} Di\ log_2 Di$$

Giả sử chúng ta phân chia biến thành n nút con và *Di* đại diện cho số lượng bản ghi thuộc nút đó. Do đó, hệ số Gain Ratio sẽ xem xét được xu hướng phân phối khi chia cây.

Áp dụng cho ví dụ trên và với cách chia thứ nhất, ta có

$$SplitInfo = -\left[\left(\frac{4}{7}\right) * log2\left(\frac{4}{7}\right)\right] - \left[\left(\frac{3}{7}\right) * log2\left(\frac{3}{7}\right)\right] = 0.98$$

$$Gain\ Ratio\ = \frac{0.09}{0.98} = 0.092$$

1.1.2.3. Điều kiện dừng

Khi nào thì ngừng chia nhỏ?

Trong các thuật toán Decision tree, với phương pháp chia trên, ta sẽ chia mãi các node nếu nó chưa tinh khiết. Như vậy, ta sẽ thu được một tree mà mọi điểm trong tập huấn luyện đều được dự đoán đúng (giả sử rằng không có hai input giống nhau nào cho output khác nhau). Khi đó, cây có thể sẽ rất phức tạp (nhiều node) với nhiều leaf node chỉ có một vài điểm dữ liệu. Như vậy, nhiều khả năng *overfitting* sẽ xảy ra.

Để tránh trường hợp này, ta có thể dừng cây theo một số phương pháp sau đây:

- Nếu node đó có entropy bằng 0, tức mọi điểm trong node đều thuộc một class.
- Nếu node đó có số phần tử nhỏ hơn một ngưỡng nào đó. Trong trường hợp này, ta chấp nhận có một số điểm bị phân lớp sai để tránh overfitting. Class cho leaf node này có thể được xác định dựa trên class chiếm đa số trong node.
- Nếu khoảng cách từ node đó đến root node đạt tới một giá trị nào đó. Việc hạn chế *chiều sâu của tree* này làm giảm độ phức tạp của tree và phần nào giúp tránh overfitting.
- Nếu tổng số leaf node vượt quá một ngưỡng nào đó.

- Nếu việc phân chia node đó không làm giảm entropy quá nhiều (information gain nhỏ hơn một ngưỡng nào đó).

Ngoài ra, ta còn có phương pháp cắt tỉa cây.

1.1.3. Một số thuật toán khác

Ngoài ***ID3*** và ***C4.5***, ta còn một số thuật toán khác như:

- *Thuật toán CHAID:* tạo cây quyết định bằng cách sử dụng thống kê chi-square để xác định các phân tách tối ưu. Các biến mục tiêu đầu vào có thể là số (liên tục) hoặc phân loại.
- *Thuật toán C&R:* sử dụng phân vùng đệ quy để chia cây. Tham biến mục tiêu có thể dạng số hoặc phân loại.
- *MARS*
- *Conditional Inference Trees*

1.1.4. Nhận xét

Ưu điểm

Cây quyết định là một thuật toán đơn giản và phổ biến. Thuật toán này được sử dụng rộng rãi bởi những lợi ích của nó:

- *Cây quyết định dễ hiểu.* Người ta có thể hiểu mô hình cây quyết định sau khi được giải thích ngắn.
- *Việc chuẩn bị dữ liệu cho một cây quyết định là cơ bản hoặc không cần thiết.* Các kỹ thuật khác thường đòi hỏi chuẩn hóa dữ liệu, cần tạo các biến phụ (*dummy variable*) và loại bỏ các giá trị rỗng.
- *Cây quyết định có thể xử lý cả dữ liệu có giá trị bằng số và dữ liệu có giá trị là tên thể loại.* Các kỹ thuật khác thường chuyên để phân tích các bộ dữ liệu chỉ gồm một loại biến. Chẳng hạn, các luật quan hệ chỉ có thể dùng cho các biến tên, trong khi mạng nơ-ron chỉ có thể dùng cho các biến có giá trị bằng số.

- *Cây quyết định là một mô hình hộp trắng.* Nếu có thể quan sát một tình huống cho trước trong một mô hình, thì có thể dễ dàng giải thích điều kiện đó bằng logic Boolean. Mạng nơ-ron là một ví dụ về mô hình hộp đen, do lời giải thích cho kết quả quá phức tạp để có thể hiểu được.
- *Có thể thẩm định một mô hình bằng các kiểm tra thống kê.* Điều này làm cho ta có thể tin tưởng vào mô hình.
- *Cây quyết định có thể xử lý tốt một lượng dữ liệu lớn trong thời gian ngắn.* Có thể dùng máy tính cá nhân để phân tích các lượng dữ liệu lớn trong một thời gian đủ ngắn để cho phép các nhà chiến lược đưa ra quyết định dựa trên phân tích của cây quyết định.

Nhược điểm

Kèm với đó, cây quyết định cũng có những nhược điểm cụ thể:

- Mô hình cây quyết định *phụ thuộc rất lớn vào dữ liệu* của bạn. Thậm chí, với một sự thay đổi nhỏ trong bộ dữ liệu, cấu trúc mô hình cây quyết định có thể thay đổi hoàn toàn.
- Cây quyết định thường gặp vấn đề *overfitting*

Pruning

Một phương pháp khác có thể giúp chúng ta tránh overfitting. Nó giúp cải thiện hiệu suất của cây bằng cách cắt các nút hoặc nút con không quan trọng. Nó loại bỏ các nhánh có tầm quan trọng rất thấp.

Chủ yếu có 2 cách để Pruning:

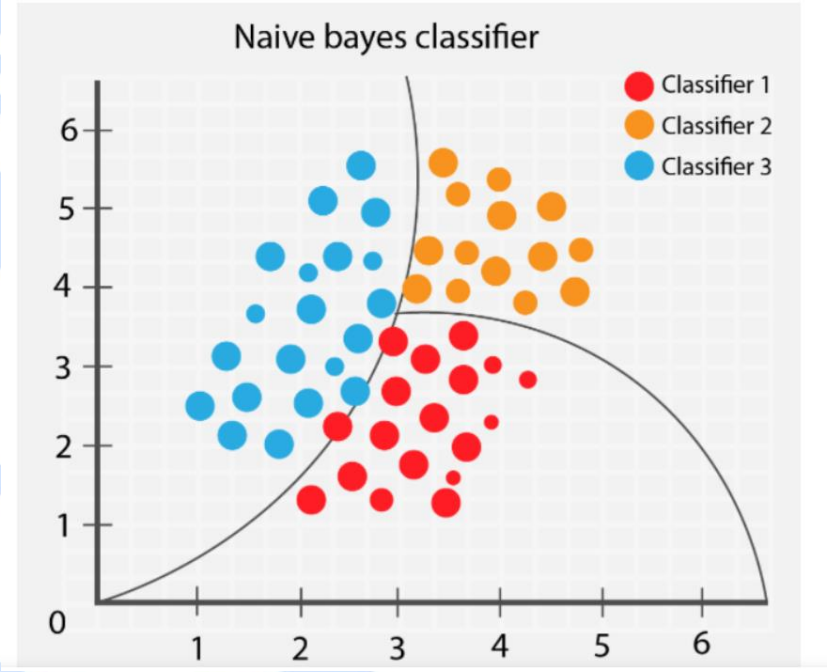
- Pre-pruning: chúng ta có thể ngừng phát triển cây sớm hơn, có nghĩa là chúng ta có thể tỉa / loại bỏ / cắt một nút nếu nó có tầm quan trọng thấp trong khi phát triển cây.
- Post-pruning: khi cây của chúng ta đã được xây dựng đến độ sâu của nó, chúng ta có thể bắt đầu tỉa các nút dựa trên ý nghĩa của chúng.

1.2. Thuật toán Naïve Bayes

1.2.1. Tổng quan

Thuật toán Naive Bayes là một thuật toán học có giám sát trong lĩnh vực Machine Learning. Nó được sử dụng rất phổ biến trong các bài toán phân loại và dự đoán. Thuật toán này dựa trên công thức Bayes để tính xác suất của một điểm dữ liệu được phân loại vào một lớp nào đó.

Thuật toán Naive Bayes được sử dụng rộng rãi trong các bài toán phân loại văn bản, phát hiện spam email, phân loại ảnh, dự đoán thời tiết, và nhiều bài toán khác.



Hình 4. Minh họa thuật toán Naïve Bayes

1.2.2. Ý tưởng

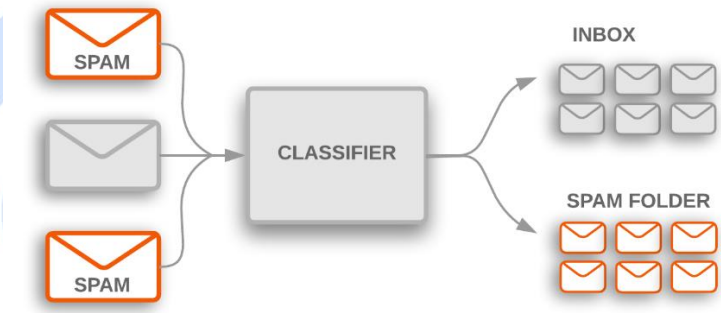
Công thức Bayes được sử dụng để tính xác suất của một sự kiện A khi đã biết thông tin về sự kiện B liên quan đến A.

Trong trường hợp của Naive Bayes, chúng ta áp dụng công thức Bayes để tính xác suất của một điểm dữ liệu được phân loại vào một lớp cụ thể, khi đã biết các giá trị của các đặc trưng (features) của điểm dữ liệu đó.

Thuật toán Naive Bayes giả định rằng các đặc trưng của một điểm dữ liệu độc lập với nhau, tức là không có sự tương quan giữa chúng. Điều này giúp cho việc tính toán xác suất trở nên đơn giản hơn, vì ta chỉ cần tính xác suất của từng đặc trưng độc lập với nhau và sau đó kết hợp chúng lại để tính xác suất của cả điểm dữ liệu.

1.2.3. Cách hoạt động

Ví dụ: Chúng ta có một tập dữ liệu gồm các email đã được phân loại thành hai lớp: "spam" và "không spam". Mỗi email được biểu diễn bởi một vector tính năng, trong đó mỗi phần tử của vector tương ứng với một từ trong email và giá trị của nó là số lần xuất hiện của từ đó trong email. Chúng ta muốn sử dụng Naive Bayes để phân loại các email mới vào một trong hai lớp này.



Hình 5. Bài toán phân loại email spam

Bước 1: ***Thu thập, tiền xử lý và chuẩn bị dữ liệu***

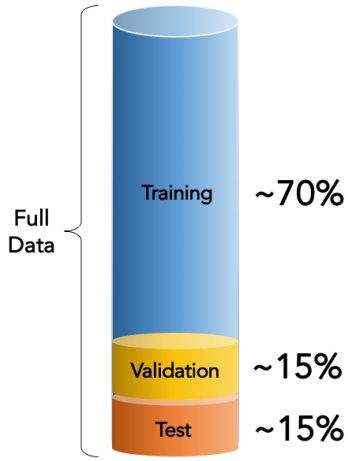
Bạn cần tiền xử lý dữ liệu, như loại bỏ dữ liệu nhiễu, chuẩn hoá dữ liệu, chọn lọc đặc trưng (feature selection) và rút trích đặc trưng (feature extraction).

Từ đó ta xây dựng một tập dữ liệu với các email đã được phân loại thành hai lớp: "spam" và "không spam".

Các email này cần được biểu diễn bằng các vector tính năng, trong đó mỗi phần tử của vector tương ứng với một từ trong email và giá trị của nó là số lần xuất hiện của từ đó trong email.

Chúng ta cũng cần phải chia tập dữ liệu thành hai phần: tập huấn luyện và tập kiểm tra. Tập huấn luyện sẽ được sử dụng để huấn luyện mô hình Naive

Bayes, trong khi tập kiểm tra sẽ được sử dụng để đánh giá hiệu suất của mô hình.



Hình 6. Chia tập dữ liệu thành train – validation – test theo tỷ lệ

Bước 2: *Tính toán xác suất tiền nghiệm (Prior probability)*

Xác suất tiên nghiệm là xác suất của các lớp "spam" và "không spam" trên toàn bộ tập huấn luyện. Chúng ta tính toán xác suất này bằng cách đếm số lượng email trong mỗi lớp và chia cho tổng số email.

Bước 3: *Tính toán xác suất điều kiện*

Xác suất điều kiện là xác suất của mỗi từ trong một email đã biết, dựa trên lớp của email đó.

Chúng ta tính toán xác suất điều kiện này bằng cách đếm số lần xuất hiện của từ đó trong các email thuộc về mỗi lớp và chia cho tổng số lần xuất hiện của từ đó trong toàn bộ tập huấn luyện.

Bước 4: *Tính toán xác suất hậu nghiệm (posterior probability)*

Xác suất hậu nghiệm là xác suất của một email thuộc về một lớp nhất định, dựa trên các tính năng của nó. Chúng ta tính toán xác suất này bằng cách sử dụng công thức Bayes:

$$P(spam|email) = \frac{P(email|spam) * P(spam)}{P(email)}$$

Trong đó:

- $P(spam|email)$là xác suất hậu nghiệm của email thuộc về lớp "spam".
- $P(email|spam)$ là xác suất điều kiện của email đã biết, dựa trên lớp "spam".
- $P(spam)$ là xác suất tiên nghiệm của lớp "spam".
- $P(email)$ là xác suất của email đã biết.

Chúng ta tính toán xác suất hậu nghiệm cho cả hai lớp và chọn lớp có xác suất cao hơn.

Bước 5: ***Đánh giá hiệu suất của mô hình***

Chúng ta sử dụng tập kiểm tra để đánh giá hiệu suất của mô hình Naive Bayes bằng cách tính toán tỷ lệ phân loại đúng trên tập kiểm tra. Các chỉ số đánh giá như độ chính xác, độ phủ, độ chuẩn xác cũng được tính toán để đánh giá mô hình.

Lưu ý rằng, việc lựa chọn đúng loại Naive Bayes trong *bước 2*, *bước 3* cho từng bài toán là rất quan trọng. Ví dụ, Bernoulli Naive Bayes được sử dụng cho các bài toán phân loại văn bản, trong khi Gaussian Naive Bayes thường được sử dụng cho các bài toán dự đoán số liệu liên tục.

1.2.4. Nhận xét

Ưu điểm của Naive Bayes so với các thuật toán phân loại khác:

1. Tốc độ huấn luyện nhanh và yêu cầu ít dữ liệu huấn luyện hơn so với các thuật toán phân loại khác.
2. Độ chính xác cao trong việc phân loại các tập dữ liệu rời rạc và đơn giản.
3. Có thể sử dụng cho các bài toán phân loại đa lớp.

Nhược điểm của Naive Bayes so với các thuật toán phân loại khác:

1. Giả định về độc lập giữa các biến đầu vào không phải lúc nào cũng đúng trong thực tế.
2. Không xử lý tốt với các biến liên tục, đòi hỏi chuyển sang dạng rời rạc, gây mất thông tin và sai sót.
3. Khi xử lý các bài toán có số lượng biến đầu vào lớn, Naive Bayes có thể dẫn đến hiện tượng (dead end), khi một biến đầu vào ảnh hưởng quá ít đến kết quả dự đoán.

So với các thuật toán phân loại khác như Decision Tree, Random Forest, SVM, Neural Networks, Naive Bayes có thể cho kết quả dự đoán tốt hơn trong những bài toán có số lượng biến đầu vào ít và dữ liệu rời rạc, nhưng thường không được sử dụng trong các bài toán yêu cầu độ chính xác cao và phức tạp hơn.

1.3. Thuật toán K-Nearest Neighbor

1.3.1. Tổng quan

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó dự đoán kết quả của dữ liệu mới. [1]

K-nearest neighbor có thể áp dụng được cho cả hai bài toán chính của Supervised learning là Classification và Regression. KNN còn được gọi là một thuật toán Instance-based hay Memory-based learning.

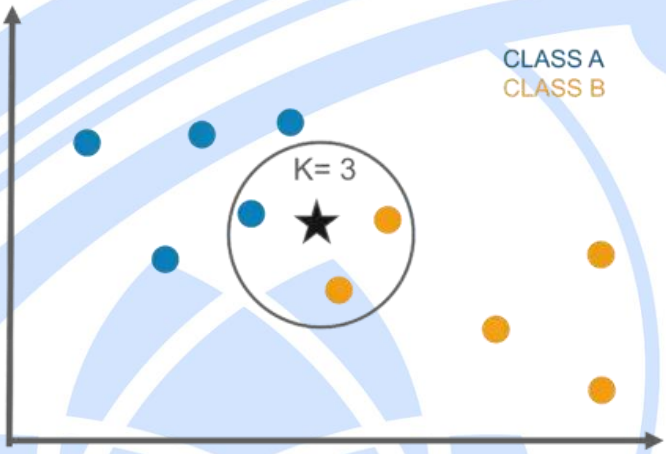
1.3.2. Ý tưởng

Với KNN trong một bài toán phân lớp, label của một điểm dữ liệu mới được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Nhãn của dữ liệu có thể được quyết định bằng major voting (bầu chọn theo số phiếu) giữa

các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label.

Tóm tắt lại, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận), mà không cần xét tới việc dữ liệu gần đó có phải dữ liệu nhiễu hay không.

1.3.3. Cách hoạt động



Hình 7. Minh họa KNN với k=3

Ở ví dụ trên với k = 3, trong 3 điểm gần nhất có 2 điểm lớp A, 1 điểm lớp B, do đó knn sẽ suy ra lable của điểm dữ liệu mới là A. Việc tìm khoảng cách giữa 2 điểm có nhiều công thức có thể sử dụng, tùy trường hợp mà chúng ta lựa chọn cho phù hợp. Đây là 3 công thức thường được dùng để tính khoảng cách giữa 2 điểm dữ liệu x, y có n thuộc tính:

$$d_{euclidean} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Euclidean (p = 2) là công thức phổ biến tình khoảng cách phổ biến nhất trong KNN, thường được sử dụng khi các vector giá trị là số thực. Từ đó ta tính ra được khoảng cách của 2 điểm dựa trên một đoạn thẳng nối liền 2 điểm với nhau.

$$d_{manhattan} = \sum_{i=1}^{n} |x_i - y_i|$$

Đây là công thức tính khoảng cách phổ biến thứ hai trong KNN (p = 1), bằng cách tính giá trị tuyệt đối của khoảng cách giữa 2 điểm. Công thức Manhattan thường được ví như một chuyến taxi đi từ điểm này tới điểm khác, vì nó thường được dùng để minh họa cho sự cách biệt khoảng cách giữa hai địa danh trên bản đồ, hay giữa hai điểm trên lược đồ.

$$d_{minkowski} = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Minkowski là dạng tổng quát của Euclidean và Manhattan, vì nó có thể chuyển đổi qua lại giữa nhiều công thức nhờ thay đổi giá trị p. Nếu p = 1 thì nó sẽ biểu diễn công thức Manhattan, nếu p = 2 thì là công thức Euclidean. Nhờ việc giá trị p dễ dàng được thay đổi mà công thức Minkowski có thể trở nên linh hoạt trong việc sử dụng và áp dụng thuật toán.

1.3.4. Nhận xét

Ưu điểm

- Dễ dàng hiểu được và sử dụng
- Có thể triển khai nhanh chóng và dễ dàng
- Ít tham số

Nhược điểm

- Vì là một lazy algorithm, nên KNN cũng tiêu tốn khá nhiều bộ nhớ trong quá trình sử dụng
- Không hoạt động hiệu quả với dữ liệu phức tạp và có nhiều thuộc tính
- Dễ gây ra hiện tượng quá khớp dữ liệu (overfitting)

1.4. Thuật toán Logistic Regression

1.4.1. Tổng quan

Logistic Regression là một phương pháp phân tích thống kê khác được Machine Learning mượn. Nó được sử dụng khi biến phụ thuộc của chúng ta là lưỡng phân hoặc nhị phân. Nó chỉ có nghĩa là một biến chỉ có 2 đầu ra, ví dụ: Một người có sống sót sau tai nạn này hay không, Học sinh có vượt qua kỳ thi này hay không. Kết quả có thể là có hoặc không (2 đầu ra 0,1). Kỹ thuật Regression này tương tự như Linear Regression và có thể được sử dụng để dự đoán Xác suất cho các bài toán phân loại.

1.4.2. Ý tưởng và cách hoạt động

Ta có bộ số X có D mẫu dữ liệu,và bộ số y là các giá trị dự đoán có giá trị 0,1

$X^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, .., x_n^{(i)})$ (Bộ giá trị ở dòng thứ I ,Có n thuộc tính x)

Ta có hàm biểu diễn quan hệ giữa X và y theo bộ tham số w = $(w_0, w_1, w_2, .., w_n)$

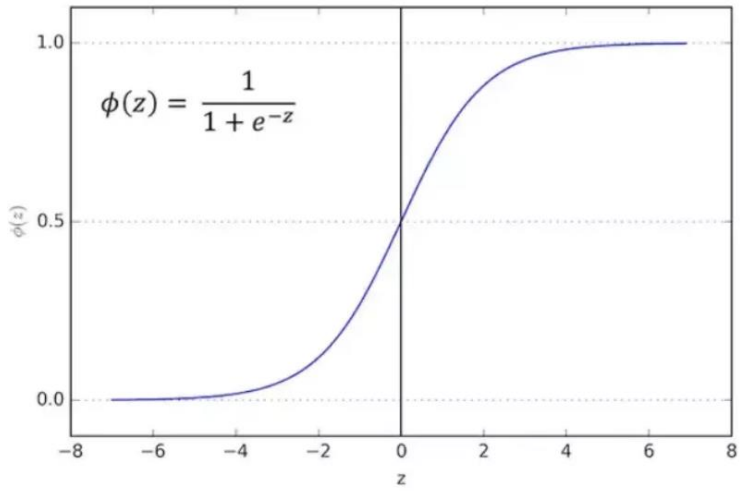
Khác với linear regression, giá trị đầu ra y là 1 số thực bất kì nên ta có thể biểu diễn mối quan hệ giữa X, y trong linear regression là:

$$Y = w^T X$$

$$Y^{(i)} = w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)} + .. + w_n * x_n^{(i)}$$

Trở lại với logicstic, giá trị đầu ra của y là có giá trị 0, hoặc 1. Mặt khác ta cũng phải biểu diễn sự liên quan giữa các giá trị của X để suy ra được giá trị y (ta dùng hàm tuyến tính lấy trong linear regression).

Hàm chuyển các giá trị liên tục trên tập R về khoảng [0,1] mà luôn có đạo hàm với mọi giá trị trên tập R và đơn giản chính là hàm Sigmoid:



Hình 8. Hàm Sigmoid

Như vậy y sẽ được biểu diễn bằng hàm sigmoid, kết quả cuối cùng của y sẽ là 1 số trong khoảng [0,1], ta sẽ làm tròn giá trị này để đưa ra kết quả cuối cùng.

$$z = w^T X$$

$$z^{(i)} = w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)} + .. + w_n * x_n^{(i)}$$

(Dòng thứ i)

Hàm z ta lấy như trên bởi vì ta còn cần phải biểu diễn mối quan hệ giữa y với các đại lượng trong X ,điều này khá giống trong linear regression

Như vậy ta sẽ có hàm dự đoán là:

$$\hat{y}_i = \Phi\left(w_0 + w_1 * x_1^{(i)} + .. + w_n * x_n^{(i)}\right) = \frac{1}{1 + e^{w_0 + w_1 * x_1^{(i)} + .. + w_n * x_n^{(i)}}}$$

Ký hiệu $z^{(i)} = z_i = f(w^T x_i) = \hat{y}_i$

Ta có thể giả sử rằng xác suất để một điểm dữ liệu X rơi vào class 1 là $f(w^T X)$ (f là hàm sigmoid) và rơi vào class 0 là $1 - (w^T X)$. Với mô hình được giả sử như vậy, với các điểm dữ liệu training (đã biết đầu ra y), ta có thể viết như sau:

$$P(y_i = 1 | x_i; w) = f(w^T x_i)$$

$$P(y_i = 0 | x_i; w) = 1 - f(w^T x_i)$$

Ký hiệu $z^{(i)} = z_i = f(w^T x_i) = \hat{y}_i$ và viết gộp lại hai biểu thức bên trên ta có:

$$P(y_i|x_i; w) = z_i^{y_i}(1 - z_i)^{1-y_i}$$

Chúng ta muốn mô hình gần với dữ liệu đã cho nhất, tức xác suất này đạt giá trị cao nhất hay là cần tìm max phương trình dưới đây:

$$P(y|X; w) = \prod_{i=1}^{D} P(y_i|x_i; w) = \prod_{i=1}^{D} z_i^{y_i}(1 - z_i)^{1-y_i}$$

Ta lấy log và thêm dấu – trước phương trình ta sẽ cần tìm giá trị nhỏ nhất cho hàm mất mát như dưới đây:

$$L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

Đến đây ta dùng gradient descent hay các biến thể của nó để tìm ra bộ tham số W tối ưu làm cho L nhỏ nhất có thể.

Ta tính đạo hàn cho 1 điểm dữ liệu

$$\frac{dL}{d\hat{y}_i} = -\frac{d\left(y_i * \log\left(\hat{y}_i\right) + \left(1 - y_i\right) * \log\left(1 - \hat{y}_i\right)\right)}{d\hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right)$$

$$\frac{d\hat{y}_i}{dw_0} = \left(\frac{d\,\phi(w_0 + w_1 * x_1^{(i)} + .. + w_n * x_n^{(i)})}{dw_0}\right) = \hat{y}_i(1 - \hat{y}_i)$$

$$\frac{dL}{dw_0} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{dw_0} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right) * \hat{y}_i\left(1 - \hat{y}_i\right) = -\left(y_i\left(1 - \hat{y}_i\right) - \left(1 - y_i\right) * \hat{y}_i\right)$$

$$= \hat{y}_i - y_i$$

Tương tự:

$$\frac{dL}{dw_1} = x_1^{(i)} * (\hat{y}_i - y_i)$$

$$\frac{dL}{dw_2} = x_2^{(i)} * (\hat{y}_i - y_i)$$

Tổng quát hóa lên ta được

$$\frac{dL}{dw_j} = x_j^{(i)} * (\hat{y}_i - y_i) \mid (x_0^0 = 1, 1 <= \text{j} <= \text{n}), \text{i là dòng dữ liệu thứ i}$$

Đấy là trên 1 điểm dữ liệu, còn trên toàn bộ dữ liệu:

$$\frac{dL}{dw_1} = \sum_{i=1}^{D} x_1^{(i)} * (\hat{y}_i - y_i)$$

$$\frac{dL}{dw_2} = \sum_{i=1}^{D} x_2^{(i)} * (\hat{y}_i - y_i)$$

.....

$$\frac{dL}{dw_n} = \sum_{i=1}^{D} x_n^{(i)} * (\hat{y}_i - y_i)$$

Tổng quát hóa lên ta được :

$$\frac{dL}{dw_j} = \sum_{i=1}^{D} \frac{dL}{dw_i} = x_j^{(i)} * (\hat{y}_i - y_i)$$ | (x_0^0=1 ,0<=j<=n),D là số dòng dữ liệu

$$(x_0^0 = 1 \text{ , } 1<=i<=D)$$

Đạo hàm thì ta cũng đã tính được, đến đây ta chỉ cần cập nhập giá trị bộ tham số W theo thuật toán gradient descent đến khi nào hàm độ lớn giữa L của 2 lần cập nhập liên tiếp giảm không đáng kể thì dừng ,ngoài ra còn có thế có các điều kiện dừng khác .

$$w_{t+1} = w_t - learning_{rate} * L'(w_t)$$

Sau khi tìm được bộ tham số w tối ưu ta tìm phương trình phần cách các điểm dữ liệu thành 2 lớp bằng 1 tham số t,thường là t=0.5

Trong trường hợp tổng quát t bất kì, $\hat{y}_i$> t thì:

$$w_0 + w_1 * x_1^{(i)} + .. + w_n * x_n^{(i)} > -Ln(\frac{1}{t} - 1)$$

1.4.3. Ưu và nhược điểm

Ưu điểm

- Đơn giản dễ hiểu, dễ thực hiện và đào tạo hiệu quả
- Hoạt động tốt khi tập dữ liệu có thể phân tách tuyến tính
- Độ chính xác tốt cho các tập dữ liệu nhỏ hơn
- Cung cấp xác suất được hiệu chỉnh tốt
- Ít bị overfitting trong các bộ dữ liệu có chiều thấp
- Có thể được mở rộng để phân loại nhiều lớp

Nhược điểm

- Chỉ có thể được sử dụng để dự đoán các chức năng rời rạc
- Không thể giải các bài toán phi tuyến tính
- Nhạy cảm với ngoại lệ

1.4.4. Ứng dụng

- Spam detection: Dự đoán mail gửi đến hòm thư của bạn có phải spam hay không.
- Credit card fraud: Dự đoán giao dịch ngân hàng có phải gian lận không.
- Health: Dự đoán 1 u là u lành hay u ác tính.
- Banking: Dự đoán khoản vay có trả được hay không.
- Investment: Dự đoán khoản đầu tư vào start-up có sinh lợi hay không

Tóm lại, thuật toán này sẽ dựa trên các dữ liệu trong quá khứ để phân lớp, đưa ra cho chúng ta 1 đáp án có khả năng chính xác cao để lựa chọn.

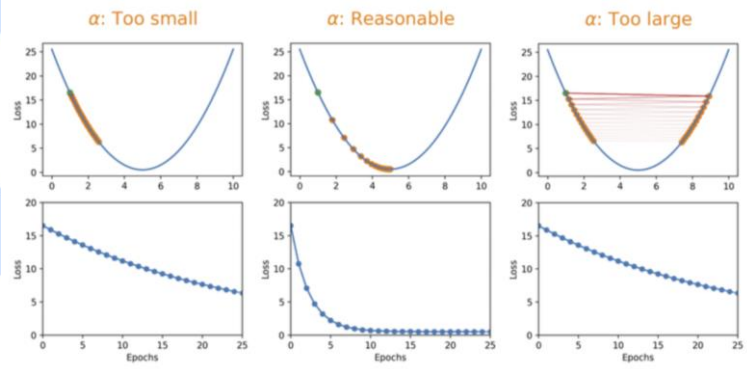
2. Phương pháp tìm kiếm tham số phù hợp GridSearch

2.1. Tại sao cần điều chỉnh các siêu tham số

Siêu tham số là tham số của mô hình mà giá trị được xác định trước khi huấn luyện. *Ví dụ:* số *neuron* của *feed-forward neural network*; số *tree* trong *random forest;* mức *penalty* của *hồi quy Lasso;...* [2]

Các số này đều được xác định trước khi thực hiện huấn luyện và có ảnh hưởng lớn tới hiệu quả của mô hình. Tuy nhiên, không có cách nào để biết chính xác đâu là giá trị tối ưu nhất.

- Trong *neural networks*, số *neuron* quá thấp có thể dẫn đến underfitting và ngược lại, overfitting. Ta cần tìm một số *neuron* phù hợp để mô hình cho ra kết quả tốt nhất.
- Trong *Gradient Descent*, *learning rate* lớn cho phép mô hình học nhanh hơn nhưng đồng thời có thể bỏ lỡ hàm mất mát tối tiểu. Mặt khác, chọn *learning rate* thấp hơn có thể mang lại cơ hội tốt hơn để tìm điểm cực tiểu cục bộ nhưng cần sự đánh đổi về số epochs lớn hơn hay tốn nhiều thời gian hơn.



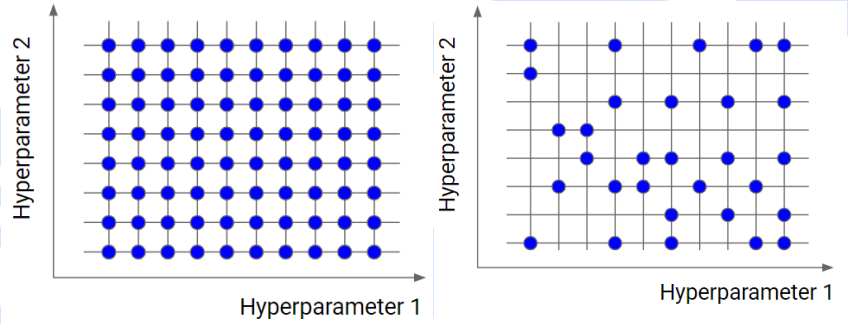
Hình 9. Ảnh hưởng của learning rate tới loss

Trong trường hợp mô hình có nhiều siêu tham số, ta cần tìm tổ hợp giá trị siêu tham số tối ưu nhất trong không gian nhiều chiều. Việc thực hiện thủ công rất phức tạp và tốn nhiều thời gian, vì vậy, cần sử dụng các phương pháp điều chỉnh siêu tham số tự động.

2.2. Phương pháp GridSearch

GridSearch là thuật toán đơn giản nhất để điều chỉnh siêu tham số:

- Chia miền của siêu tham số thành một lưới rời rạc
- Thử mọi kết hợp giá trị của lưới này, tính toán một vài đánh giá bằng Cross Validation
- Điểm của lưới mà tại đó giá trị trung bình của Cross Validation được tối đa hóa là tổ hợp tối ưu của siêu tham số



Hình 10. Minh họa GridSearch và Random Search

Như đã đề cập ở trên, các giá trị được định nghĩa trước cho siêu tham số sẽ được truyền cho hàm *GridSearchCV* [3]. Điều này được thực hiện bằng cách định nghĩa một từ điển mà mỗi siêu tham số cụ thể sẽ đi cùng với các giá trị mà nó có thể nhận. *Ví dụ:*

```
{ 'C': [0.1, 1, 10, 100, 1000],
   'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
   'kernel': ['rbf',’linear’,'sigmoid']  }
```

Với C, gamma và kernel là một vài siêu tham số của mô hình SVM, *GridSearchCV* sẽ thử tất cả các tổ hợp giá trị được truyền từ từ điển và đánh giá mô hình cho từng tổ hợp bằng phương pháp Cross Validation. Sau đó, ta nhận được accuracy/loss cho mọi tổ hợp siêu tham số và chọn ra tổ hợp mang lại hiệu quả tốt nhất.

Ưu điểm

GridSearch là một thuật toán toàn diện, mở rộng tất cả các kết hợp nên đảm bảo tìm thấy điểm tối ưu nhất trong miền.

Hạn chế

Hạn chế lớn của *GridSearch* là rất chậm. Việc kiểm tra mọi kết hợp của không gian đòi hỏi rất nhiều thời gian. Mọi điểm trong lưới đều cần thực hiện *k-fold Cross Validation*, tức cần *k* bước huấn luyện. Vì vậy, việc điều chỉnh các tham số của một mô hình theo cách này khá phức tạp và tốn chi phí.

➔ *Nếu cần xác định tổ hợp tối ưu nhất của các giá trị siêu tham số, GridSearch vẫn là một lựa chọn tốt.*

2.3. Sử dụng GridSearchCV trong Python

Install sklearn library

```
pip install sklearn
```

Import sklearn library

```
from sklearn.model_selection import GridSearchCV
```

Import model

```
from sklearn.svm import SVC
```

Tạo danh sách từ điển siêu tham số

```
parameters = [{
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
    'C': [1,2,3,300,500],
    'max_iter': [1000,100000]}]
```

Giả sử tìm các giá trị siêu tham số tối ưu cho:

- ***kernel:*** mô hình tự đào tạo trong các kernels sau và cung cấp giá trị tốt nhất của các giá trị `'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'`
- ***C:*** mô hình thử nghiệm các giá trị C `[1,2,3,300,500]`
- ***max_iter:*** mô hình sử dụng các giá trị max_iter `[1000,100000]` và đưa ra giá trị tốt nhất

Khởi tạo *GridSearchCV* và truyền các tham số

```
clf = GridSearchCV(
    SVC(), parameters, scoring='accuracy'
    )
clf.fit(X_train, y_train)
```

Ở đây, ta sử dụng *accuracy* để đánh giá hiệu quả.

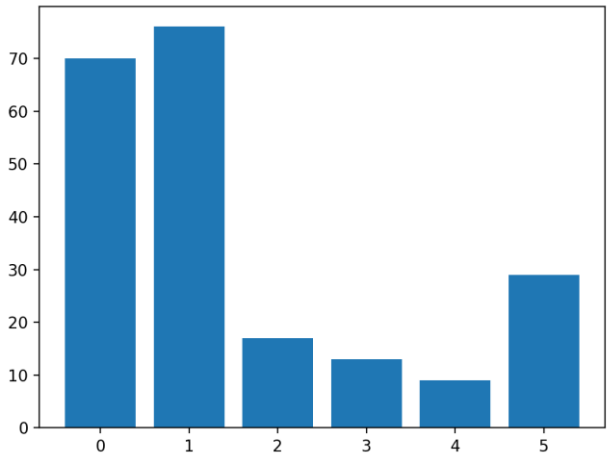
Cuối cùng, in ra các tham số tốt nhất (các tham số tối đa hóa scoring)

```
print(clf.best_params_)
```

3. Phương pháp cân bằng dữ liệu

3.1. Mất cân bằng dữ liệu

Mất cân bằng dữ liệu là hiện tượng những bộ dữ liệu có phân phối quan sát không đồng đều giữa các lớp, tức một vài nhãn lớp có số lượng quan sát rất cao trong khi có nhãn thì số lượng quan sát rất thấp. [4]



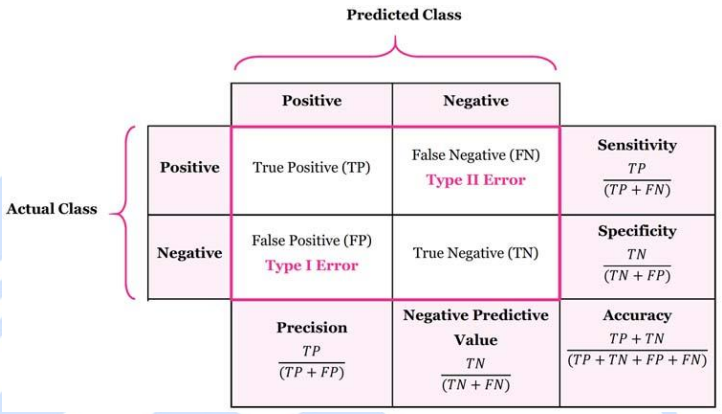
Hình 11. Biểu đồ số lượng mẫu của mỗi lớp trong bộ dữ liệu phân loại Glass

Mất cân bằng dữ liệu nghiêm trọng dẫn đến hiện tượng mô hình phân lớp có xu hướng thiên vị nhóm đa số và dự đoán kém chính xác trên nhóm thiểu số. Trong khi đó, việc dự đoán chính xác mẫu thuộc nhóm thiểu số thường quan trọng hơn rất nhiều. Để cải thiện kết quả dự đoán, ta cần những điều chỉnh thích hợp để mô hình đạt được độ chính xác cao trên nhóm thiểu số.

3.2. Các phương pháp khắc phục

3.2.1. Chọn độ đo đánh giá phù hợp

Confusion Matrix thường được sử dụng cho bài toán phân loại để cung cấp cái nhìn tổng quan về hiệu quả phân lớp của mô hình. Nó chứa thông tin phân loại trên thực tế và dự đoán.



Hình 12. Confusion Matrix và những độ đo liên quan

Giả sử trong 100 bệnh nhân chỉ có 5 bệnh nhân được chẩn đoán có bệnh. Lúc này, nhóm đa số là không có bệnh với 95% và nhóm thiểu số là có bệnh với chỉ 5%. Thông thường nếu chỉ sử dụng *Accuracy* để đánh giá $\left(\frac{0+95}{0+95+0+5} = 0.95 = 95\%\right)$, ta thấy độ chính xác của mô hình đạt 95% cho dù không xác định được nhóm thiểu số, điều này dẫn đến sự đánh giá sai lệch về độ hiệu quả của mô hình. Khi đó ta có thể cân nhắc tới một số metrics thay thế như:

- ***Precision:*** thể hiện độ tin cậy của kết quả khi mô hình xác định một mẫu thuộc về lớp đó (mức độ dự báo chính xác những trường hợp được dự báo là Positive)
- ***Recall:*** thể hiện khả năng mô hình có thể phát hiện được lớp đó (mức độ dự báo chính xác những trường hợp là Positive trong những trường hợp thực tế là Positive)
- ***F1-Score:*** cân bằng giữa *Precision* và *Recall*

Đây là chỉ số thay thế lý tưởng cho *Accuracy* khi mô hình có tỷ lệ mất cân bằng mẫu cao.

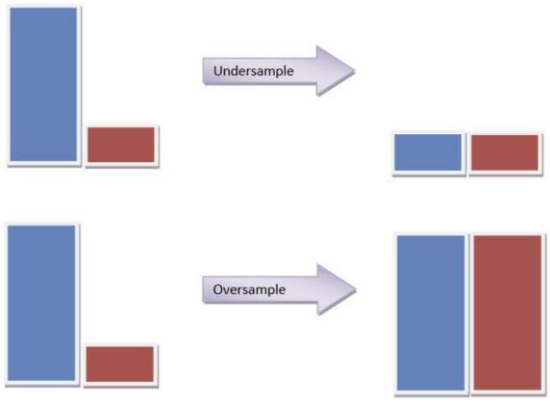
$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Nếu mô hình phân loại dự đoán sai nhóm thiểu số và số lượng False Positive tăng, *Precision* sẽ giảm và F_1 cũng giảm. Ngoài ra, nếu mô hình phân loại kém trong việc phát hiện ra các lớp thiểu số, tức nhiều lớp thiểu số này bị dự đoán sai thành lớp đa số, False Negative sẽ tăng lên và *Recall*, F_1 đều sẽ thấp. *F1-Score* chỉ tăng khi số lượng và chất lượng dự đoán cùng được cải thiện.

- ***Kappa-Score:*** chỉ số đo lường mức độ liên kết tin cậy (inter-rater reliability) cho các categories.
- ***Gini:*** đo lường sự bất bình đẳng trong phân phối giữa Positive và Negative được dự báo từ mô hình.
- ***AUC:*** biểu diễn mối quan hệ giữa độ nhạy (sensitivity) và độ đặc hiệu (specificity), đánh giá khả năng phân loại good và bad được dự báo từ mô hình.
- ***ROC:*** thể hiện mối quan hệ, sự đánh đổi giữa độ nhạy và tỷ lệ cảnh báo sai; là đường cong biểu diễn tỷ lệ dự báo True Positive Rate (TPR) dựa trên tỷ lệ dự báo False Positive Rate (FPR) tại các ngưỡng Threshold khác nhau.

3.2.2. Resampling (Oversampling và Undersampling)

Kỹ thuật làm giảm kích thước mẫu ở nhóm đa số hoặc gia tăng kích thước mẫu ở nhóm thiểu số để cân bằng dữ liệu và mô hình phân loại sẽ có tầm quan trọng như nhau ở cả hai nhóm.



Hình 13. Kỹ thuật Undersample và Oversample

Sklearn.utils resample có thể được áp dụng cho cả *Undersampling* và *Oversampling.*

```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df_train[(df_train['Is_Lead']==0)]
df_minority = df_train[(df_train['Is_Lead']==1)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
          replace=True,    # sample with replacement
          n_samples= 131177, # to match majority class
          random_state=42)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

BalancedBaggingClassifier tương tự như phân loại sklearn nhưng có thêm khả năng cân bằng. Phương pháp này bổ sung thêm một bước để cân bằng tập training tại một thời điểm phù hợp cho một bộ lấy mẫu nhất định. Mô hình phân loại này nhận hai tham số đặc biệt:

- *sampling_strategy:* quyết định loại sampling (‘majority’ – chỉ thực hiện resample ở nhóm đa số, ‘all’ – resample tất cả các lớp,…)
- *replacement:* quyết định mẫu có thay thế hay không

```
from imblearn.ensemble import BalancedBaggingClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
#Create an instance
classifier =
BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                  sampling_strategy='not majority',
                                  replacement=False,
                                  random_state=42)
classifier.fit(X_train, y_train)
preds = classifier.predict(X_test)
```

3.2.2.1. Undersampling

- **Ưu điểm:** nhanh chóng, dễ dàng thực hiện mà không cần đến thuật toán giả lập mẫu.
- **Nhược điểm:** kích thước mẫu bị giảm đáng kể. Giả sử nhóm thiểu số có kích thước là 500, như vậy để tạo ra sự cân bằng, cần giảm kích thước mẫu của nhóm đa số từ 10000 về 500. Tập huấn luyện mới khá nhỏ, không đại diện cho phân phối của toàn bộ tập dữ liệu và dễ dẫn đến hiện tượng overfitting.

Do đó, có thể không nhất thiết lựa chọn tỷ lệ mẫu giữa nhóm đa số và thiểu số là 50:50, có thể giảm dần về 80:20, 70:30 hoặc 60:40 và tìm ra phương án nào mang lại hiệu quả dự báo tốt nhất trên tập test.

3.2.2.2. Oversampling

Có hai phương pháp chính để thực hiện Oversampling:

- ***Lựa chọn mẫu có tái lập***

  Naive random Oversampling là phương pháp tái chọn mẫu dựa trên giả thuyết dữ liệu mẫu giả lập mới sẽ giống dữ liệu sẵn có. Do đó ta sẽ cân bằng mẫu bằng cách lựa chọn ngẫu nhiên có lặp lại các quan sát thuộc nhóm thiểu số.

- ***Mô phỏng mẫu mới dựa trên tổng hợp của các mẫu cũ***

SMOTE (Synthetic Minority Oversampling) và **ADASYN (Adaptive Synthetic Sampling)** là các phương pháp sinh mẫu nhằm gia tăng kích thước mẫu của nhóm thiểu số trong trường hợp mất cân bằng mẫu.

Để gia tăng kích thước mẫu, với mỗi một mẫu thuộc nhóm thiểu số ta sẽ lựa chọn ra mẫu láng giềng gần nhất và thực hiện tổ hợp tuyến tính để tạo ra mẫu giả lập. Phương pháp để lựa chọn ra các láng giềng của một quan sát có thể dựa trên thuật toán **kNN** hoặc **SVM**.

```
from imblearn.over_sampling import SMOTE

# Resampling the minority class. The strategy can be changed as
required.

sm = SMOTE(sampling_strategy='minority', random_state=42)

# Fit the model to generate the data.

oversampled_X, oversampled_Y =
sm.fit_sample(df_train.drop('Is_Lead', axis=1),
df_train['Is_Lead'])

oversampled = pd.concat([pd.DataFrame(oversampled_Y),
pd.DataFrame(oversampled_X)], axis=1)
```

3.2.3. Tối ưu hóa threshold

Thông thường, xác suất của các dự đoán được gán cho một lớp nhất định dựa trên ngưỡng mặc định (thường là 0.5). Để khắc phục vấn đề mất cân bằng dữ liệu, ta có thể tìm giá trị ngưỡng tối ưu để phân lớp hiệu quả bằng:

- Đường cong ROC hoặc Đường cong Precision-Recall
- Các phương pháp tìm kiếm tham số phù hợp như GridSearch

4. Thực nghiệm

Bộ dữ liệu thực hiện: **Social Network Ads**

4.1. Environment set up

Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import GridSearchCV

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.utils import resample
from imblearn.ensemble import BalancedBaggingClassifier
```

Đây là bộ dữ liệu nhằm phân khúc đối tượng trong quảng tiếp thị sản phẩm. Ta có:

- Các cột 'UserID', 'Gender', 'Age' và 'EstimatedSalary' là các thuộc tính thông tin của khách hàng được đưa vào xem xét.
- Cột 'Purchased' là nhãn của mẫu.
    - Nếu giá trị của 'Purchased' là 1 tức là người đó có khả năng mua hàng.
    - Còn 0 là ngược lại.

```
df = pd.read_csv('/content/Social_Network_Ads.csv')
df
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

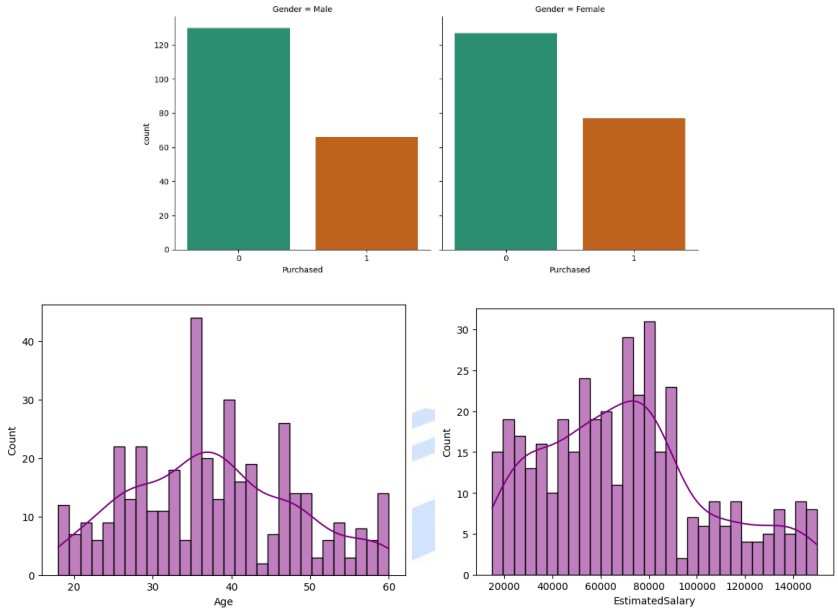
400 rows × 5 columns

Bộ dữ liệu có tổng cộng 400 mẫu và ta có các cột đều có kiểu dữ liệu int64 ngoài trừ 'Gender' mang kiểu dữ liệu object.

```
df.info()
```

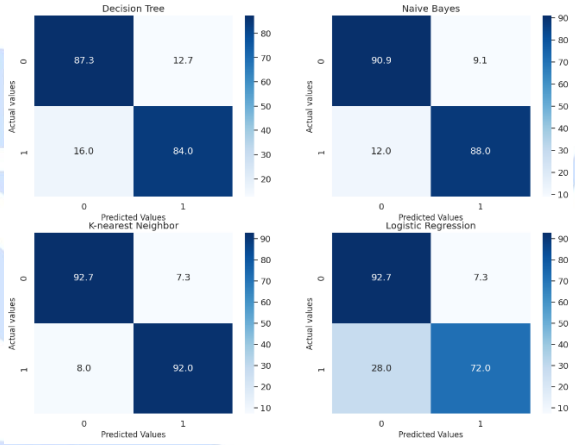
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

Trực quan dữ liệu



4.2. Thực nghiệm với các phương pháp

test_size = 0.2



Hình 14. Confusion Matrix của các thuật toán phân loại

Thực hiện resampling

```
pd.DataFrame(y_train).value_counts()

0    202
1    118
dtype: int64
```

Hình 15. Dữ liệu ban đầu

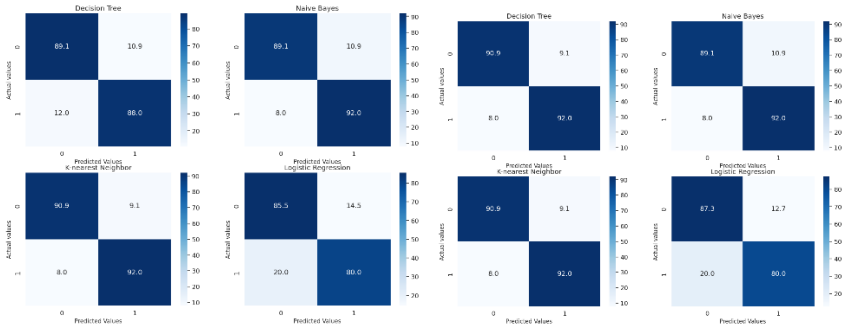
```
pd.DataFrame(y_undersampled).value_counts()

0    118
1    118
dtype: int64
```

```
pd.DataFrame(y_upsampled).value_counts()

0    202
1    202
dtype: int64
```

Hình 16. Dữ liệu sau khi thực hiện resample



Hình 17. Confusion Matrix của các thuật toán phân loại sau khi áp dụng undersample và upsample

Thực hiện GridSearch lần lượt cho các phương pháp:

- Decision Tree

```
parameters = {
    "criterion": ['gini', 'entropy'],
    "max_depth": [2, 3, 5, 10, 20],
    "min_samples_split": [5, 10, 20, 50, 100],
    "min_samples_leaf": range(1,5)
}
```

```
grid_dt.best_score_
```

```
0.90625
```

```
grid_dt.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 5,
 'min_samples_leaf': 1,
 'min_samples_split': 20}
```

- Naïve Bayes

```
params = {'var_smoothing': np.logspace(0,-9, num=100)}
```

```
grid_nb.best_score_
```

```
0.875
```

```
grid_nb.best_params_
```

```
{'var_smoothing': 0.008111308307896872}
```

- K-nearest Neighbor

```
knn = KNeighborsClassifier()
k_range = list(range(1, 31))
param_grid = dict(n_neighbors=k_range)
```

```
grid_knn.best_params_
```

```
{'n_neighbors': 11}
```

- Logistic Regression

```
params = {'C': np.logspace(-3, 3, 7), 'penalty': ['l1', 'l2']}
```

```
grid_lr.best_score_
```

```
0.840625
```

```
grid_lr.best_params_
```

```
{'C': 100.0, 'penalty': 'l2'}
```

4.3. Kết quả

Trên dữ liệu gốc:

	Accuracy	Precision	Recall	F1
Decision Tree	0.8625	0.750000	0.84	0.792453
Naive Bayes	0.9000	0.814815	0.88	0.846154
K-nearest Neighbor	0.9250	0.851852	0.92	0.884615
Logistic Regression	0.8625	0.818182	0.72	0.765957

Sau khi thực hiện undersample:

	Accuracy	Precision	Recall	F1
Decision Tree	0.8875	0.785714	0.88	0.830189
Naive Bayes	0.9000	0.793103	0.92	0.851852
K-nearest Neighbor	0.9125	0.821429	0.92	0.867925
Logistic Regression	0.8375	0.714286	0.80	0.754717

Sau khi thực hiện upsample:

	Accuracy	Precision	Recall	F1
Decision Tree	0.9125	0.821429	0.92	0.867925
Naive Bayes	0.9000	0.793103	0.92	0.851852
K-nearest Neighbor	0.9125	0.821429	0.92	0.867925
Logistic Regression	0.8500	0.740741	0.80	0.769231

Sau khi tối ưu tham số với GridSearch:

	Accuracy	Precision	Recall	F1
Decision Tree	0.9000	0.869565	0.80	0.833333
Naive Bayes	0.9000	0.814815	0.88	0.846154
K-nearest Neighbor	0.9250	0.851852	0.92	0.884615
Logistic Regression	0.8625	0.791667	0.76	0.775510

5. Tham khảo

[1] "Develop k-Nearest Neighbors in Python From Scratch," [Online]. Available: http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/.

[2] "Hyperparameter tuning. Grid search and random search," [Online]. Available: https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search/.

[3] "Hyperparameter Tuning with GridSearchCV," [Online]. Available: https://www.mygreatlearning.com/blog/gridsearchcv/.

[4] "5 Techniques to Handle Imbalanced Data For a Classification Problem," [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/.