ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

HỌC MÁY (CO3117)

Lớp: L01 - HK232

Đề tài:

Áp dụng các mô hình học máy cho bài toán Dự báo trời mưa tại Sydney

Giảng viên hướng dẫn: Nguyễn Đức Dũng

Sinh viên thực hiện: Nguyễn Thái Tân - 2112256

Thành phố Hồ Chí Minh, 06/2024

Mục lục

1 Giới thiệu

Dự báo thời tiết, đặc biệt là dự báo trời mưa, đóng vai trò quan trọng trong cuộc sống hàng ngày của con người và các hoạt động kinh tế xã hội. Quá trình này không chỉ mang lại những thay đổi đáng kể về kinh tế mà còn ảnh hưởng đến môi trường và các hoạt động sinh hoạt. Một trong những lợi ích đáng chú ý nhất từ quá trình này là giúp người dân chuẩn bị tốt hơn cho các hoạt động ngoài trời, từ đó giảm thiểu những sự bất tiện do mưa gây ra.

Công nghệ dự báo thời tiết đã được khuyến nghị là công cụ quan trọng trong nhiều lĩnh vực nhằm giảm thiểu tác động tiêu cực của các hiện tượng thời tiết khắc nghiệt và hỗ trợ các hoạt động kinh tế xã hội. Tuy nhiên, không dễ để đạt được độ chính xác cao trong dự báo mưa do nhiều yếu tố khác nhau có thể ảnh hưởng đến kết quả. Việc sử dụng các mô hình học máy như Logistic Regression, Support Vector Classifier, Random Forest,... đã được đề xuất nhằm cải thiện độ chính xác của dự báo mưa.

Chính vì vậy, nghiên cứu này đang hướng đến việc xây dựng một hệ thống dự báo mưa thông minh giúp tiếp cận người dân một cách hiệu quả và giúp họ có thêm sự chuẩn bị trước khi trời mưa. Trước hết là cung cấp một công cụ dự đoán rằng vào ngày mai trời có mưa hay không tại thành phố Sydney. Mục tiêu của nghiên cứu này là cố gắng phát triển các mô hình supervised learning có thể dự đoán trời mưa dựa trên dữ liệu khí tượng. Từ các mô hình trên, đưa ra những đánh giá, so sánh kết quả và kết luận.

2 Yêu cầu bài toán

Yêu cầu bài toán về dự báo trời mưa hay không là một trong những vấn đề quan trọng trong lĩnh vực thời tiết. Đối với người dân, việc biết được thời tiết có mưa hay không sẽ giúp họ sắp xếp được lịch trình và kế hoạch làm việc một cách hiệu quả.

2.1 Đầu vào

Các dữ liệu về thời tiết liên quan tại thành phố Sydney, được đo đạc theo từng ngày:

- Nhiệt độ
- Lượng mưa
- Lượng bốc hơi
- Lượng ánh sáng
- Gió: hướng gió, vận tốc gió
- Độ ẩm
- Áp suất không khí
- Tình trạng mây

2.2 Đầu ra

Dự đoán rằng ngày mai có mưa hay không.

3 Các thuật toán được áp dụng

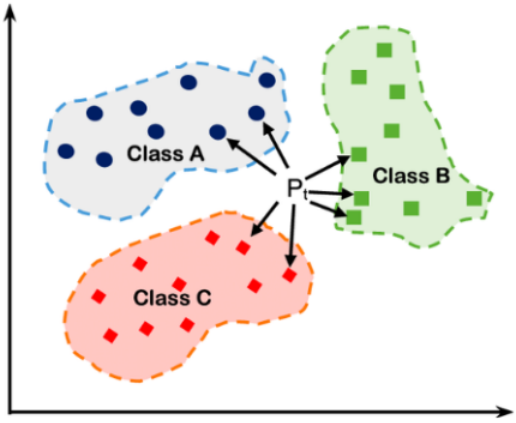
3.1 KNN - K-nearest neighbor ([8])

3.1.1 Tổng quan

KNN là một trong những thuật toán supervised-learning đơn giản nhất trong Machine Learning. KNN có thể áp dụng được vào cả bài toán phân loại và bài toán hồi quy.

Khi training, thuật toán này không học từ dữ liệu training, mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. KNN 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, không quan tâm đến việc có dữ liệu nhiễu hay không. Cụ thể hơn,

- Trong bài toán phân loại, nhãn của một điểm dữ liệu mới được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Nhãn của một dữ liệu có thể được quyết định bằng việc bầu chọn theo số lượng nhãn giữa các điểm gần nhất (có thể đánh trọng số).
- Trong bài toán hồi quy, đầu ra của một điểm dữ liệu sẽ bằng trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.



Hình 1: *Minh họa KNN, nguồn*

3.1.2 Xây dựng công thức

Giả sử S là tập các dữ liệu đã được phân loại thành 2 nhãn, một điểm dữ liệu mới x chưa biết nhãn và một số k chọn trước.

Như định nghĩa, với tập hợp K gồm k điểm có khoảng cách gần nhất so với x, đầu ra của điểm dữ liệu x này là

$$\frac{1}{k}\sum_{i\in K} d\left(x,i\right)$$

Trong đó, d chính là hàm đo khoảng cách giữa hai điểm x và y có n thuộc tính. Có nhiều cách đo khoảng cách, một số ví dụ tiêu biểu

- Euclid: $\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2}$
- Manhattan: $\sum_{i=1}^{n}\left|x_i-y_i\right|$

- Minkowski: $(\sum_{i=1}^{n}(|x_i - y_i|)^q)^{\frac{1}{q}}$

3.1.3 Ưu - nhược điểm

Ưu điểm

- Dễ hiểu và dễ triển khai. Nó không yêu cầu huấn luyện mô hình phức tạp mà chỉ cần lưu trữ các mẫu huấn luyện.
- KNN hoạt động tốt với các tập dữ liệu nhỏ và các ứng dụng mà dữ liệu không thay đổi thường xuyên.

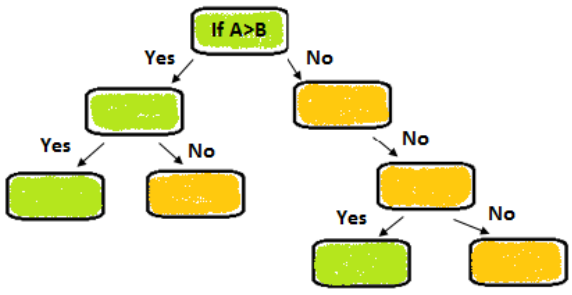
Nhược điểm

- KNN có thể trở nên chậm và không thực tế đối với các tập dữ liệu lớn vì phải tính khoảng cách từ điểm cần dự đoán tới tất cả các điểm trong tập huấn luyện.
- Hiệu suất của KNN phụ thuộc vào sự lựa chọn tham số K. K quá nhỏ có thể dẫn đến overfitting, trong khi K quá lớn có thể làm underfitting.
- KNN có thể bị ảnh hưởng bởi các giá trị ngoại lai (outliers) trong tập huấn luyện, làm cho dự đoán bị lệch.

3.2 Cây quyết định - Decision Tree ([7])

3.2.1 Tổng quan

Thuật toán Decision Tree là một phương pháp quan trọng được sử dụng cho cả bài toán phân loại và hồi quy. Với các bài toán có nhiều thuộc tính và mỗi thuộc tính có nhiều giá trị khác nhau, một phương pháp đơn giản thường được sử dụng là tại mỗi bước, một thuộc tính tốt nhất sẽ được chọn ra dựa trên một tiêu chuẩn nào đó. Với mỗi thuộc tính được chọn, ta chia dữ liệu vào các child node tương ứng với các giá trị của thuộc tính đó rồi tiếp tục áp dụng phương pháp này cho mỗi child node, cho đến khi phân loại được kết quả cuối cùng. Decision Tree là thuật toán như vậy.



Hình 2: *Minh họa Decision Tree, nguồn*

3.2.2 Xây dựng cây quyết định

Như đã trình bày, với mỗi thuộc tính được chọn, ta chia dữ liệu vào các child node tương ứng. Ta gọi một phép phân chia là “tốt" nếu dữ liệu trong mỗi child node hoàn toàn thuộc vào một class. Nếu dữ liệu trong các child node vẫn lẫn vào nhau theo tỉ lệ lớn, ta coi rằng phép phân chia đó chưa thực sự

tốt. Do đó ta cần có một hàm số đo độ tinh khiết của một phép phân chia. Hàm số thường dùng trong Decision Tree là Entropy.

Cho một phân phối xác suất của một biến rời rạc x có thể nhận n giá trị khác nhau $x_1, x_2, ..., x_n$. Giả sử rằng xác suất để x nhận các giá trị này là $p_i = p(x = x_i)$ với $0 \leq p_i \leq 1$ và $\sum_{i=1}^{n} p_i = 1$. Ký hiệu phân phối này là $\mathbf{p}$. Entropy của phân phối này được định nghĩa là

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

Xét bài toán với C class khác nhau. Giả sử ta đang làm việc với một node (không phải node lá) với các điểm dữ liệu tạo thành một tập S với số phần tử là $|S| = N$. Giả sử thêm rằng trong số N điểm dữ liệu này, N_c điểm thuộc vào class c. Xác suất để mỗi điểm dữ liệu rơi vào một class c được xấp xỉ bằng $\frac{N_c}{N}$. Như vậy, entropy tại node này được tính bởi

$$H(S) = -\sum \frac{N_c}{N} \log\left(\frac{N_c}{N}\right)$$

Tiếp theo, giả sử thuộc tính được chọn là x. Dựa trên x, các điểm dữ liệu trong S được phân ra thành 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 định nghĩa

$$H(x, S) = \sum_{k=1}^{K} \frac{m_k}{N} H(S_K)$$

Tiếp theo, ta định nghĩa information gain dựa trên thuộc tính x

$$G(x, S) = H(S) - H(x, S)$$

Tại mỗi node, thuộc tính được chọn được xác định dựa trên thuộc tính khiến cho information gain đạt giá trị lớn nhất

$$x^* = \underset{x}{\text{argmax}} G(x, S) = \underset{x}{\text{argmin}} H(x, S)$$

Tiếp tục làm như vậy cho đến khi các node được tinh khiết.

3.2.3 Ưu - nhược điểm

Ưu điểm

- Dễ hiểu và trực quan. Người dùng có thể dễ dàng diễn giải các quy tắc quyết định dưới dạng các điều kiện if-else.
- Decision Tree có khả năng xử lý các mối quan hệ phi tuyến tính giữa các đặc trưng và nhãn.
- Decision Tree có thể xử lý cả dữ liệu phân loại và dữ liệu liên tục mà không cần chuyển đổi chúng sang dạng số.

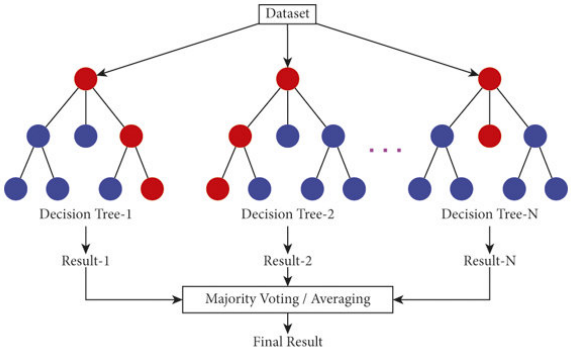
Nhược điểm

- Decision Tree dễ bị overfitting, đặc biệt là khi cây quá sâu hoặc có nhiều nhánh. Điều này có thể được khắc phục bằng cách cắt tỉa cây (pruning) hoặc đặt giới hạn độ sâu.
- Một thay đổi nhỏ trong dữ liệu có thể dẫn đến một cây quyết định hoàn toàn khác, do đó, mô hình không ổn định.

3.3 Random Forest ([2])

3.3.1 Tổng quan

Random Forest là một thuật toán học máy giám sát được sử dụng rộng rãi trong các bài toán phân loại và hồi quy. Thuật toán này xây dựng nhiều cây quyết định trên các mẫu khác nhau và lấy phiếu đa số của chúng để phân loại và lấy trung bình trong trường hợp hồi quy.



Hình 3: *Minh họa Random Forest, nguồn*

3.3.2 Xây dựng Random Forest

Giả sử bộ dữ liệu có n dữ liệu và mỗi dữ liệu có d thuộc tính. Lấy ngẫu nhiên n dữ liệu từ bộ dữ liệu với kĩ thuật Bootstrapping, nghĩa là từ n dữ liệu ban đầu, lấy ngẫu nhiên 1 dữ liệu, sau đó không bỏ dữ liệu đấy ra mà vẫn giữ lại trong tập dữ liệu ban đầu, rồi tiếp tục lấy 1 dữ liệu cho tới khi đủ n dữ liệu. Như vậy, bộ dữ liệu mới có thể có những dữ liệu trùng nhau.

Sau khi chọn được bộ dữ liệu như trên thì chọn ngẫu nhiên $k < d$ thuộc tính. Dùng thuật toán Decision Tree để xây dựng cây quyết định với bộ dữ liệu này. Do quá trình xây dựng mỗi cây quyết định đều có yếu tố ngẫu nhiên nên kết quả là các cây quyết định trong thuật toán Random Forest có thể khác nhau.

Thuật toán Random Forest sẽ bao gồm nhiều cây quyết định, mỗi cây được xây dựng dùng thuật toán Decision Tree trên tập dữ liệu khác nhau và dùng tập thuộc tính khác nhau. Sau đó kết quả dự đoán của thuật toán Random Forest sẽ được tổng hợp từ các cây quyết định.

3.3.3 Ưu - nhược điểm

Ưu điểm

- Random Forest thường có độ chính xác cao hơn so với một cây quyết định đơn lẻ do sự kết hợp của nhiều cây, giảm thiểu lỗi dự đoán.
- Random Forest ít nhạy cảm với thay đổi trong dữ liệu so với cây quyết định đơn lẻ. Một thay đổi nhỏ trong dữ liệu ít có khả năng làm thay đổi kết quả cuối cùng.

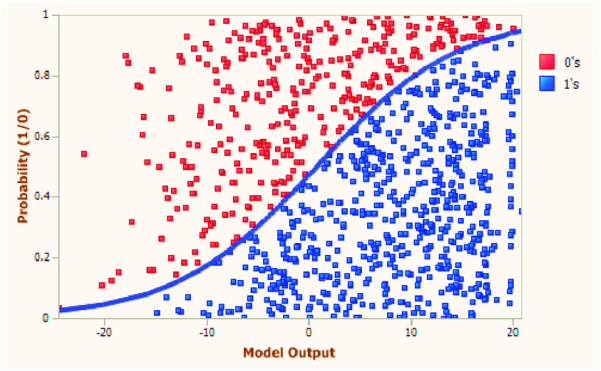
Nhược điểm

- Random Forest yêu cầu nhiều tài nguyên tính toán hơn so với một cây quyết định đơn lẻ, đặc biệt là với số lượng cây lớn hoặc dữ liệu lớn.
- Việc dự đoán của Random Forest có thể chậm hơn vì cần phải tính toán kết quả từ nhiều cây.

3.4 Logistic Regression ([4])

3.4.1 Tổng quan

Mặc dù có tên gọi là Regression hay hồi quy, nhưng Logistic Regression là một mô hình phổ biến được sử dụng để phân loại nhị phân. Vì Logistic Regression làm việc dựa trên nguyên tắc của hàm sigmoid, giá trị đầu ra của hàm này nằm trong khoảng (0;1) nên nó sẽ thể hiện được xác suất của việc phân loại thuộc về một trong hai lớp nhị phân.



Hình 4: *Minh họa Logistic Regression, nguồn*

3.4.2 Xây dựng hàm mất mát

Giả sử rằng xác suất để một điểm dữ liệu x rơi vào class 1 là $f\left(w^T x\right)$ và rơi vào class 0 là $1-f\left(w^T x\right)$. Ký hiệu $z_i = f\left(w^T x\right)$. Với giả sử như vậy, với các điểm dữ liệu training (đã biết đầu ra y), ta có thể viết như sau:

$$P\left(y_i|x_i, w\right) = z_i^{y_i}\left(1 - z_i\right)^{1-y_i}$$

Mục đích cuối cùng là tìm được hệ số w sao cho z_i càng gần với 1 càng tốt với các điểm dữ liệu thuộc class 1 và càng gần với 0 càng tốt với những điểm thuộc class 0. Chúng ta muốn mô hình gần với dữ liệu đã cho nhất, tức xác suất vừa nêu trên phải đạt giá trị cao nhất. Nói cách khác, với toàn bộ training set $X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{d\times N}$ và $y = [y_1, y_2, ..., y_N]$, cần tìm w sao cho

$$w = \underset{w}{\operatorname{argmax}} P\left(y|X, w\right)$$

Giả sử thêm rằng các điểm dữ liệu được sinh ra một cách ngẫu nhiên độc lập với nhau,

$$P\left(y|X, w\right) = \prod_{i=1}^{N} P\left(y_i|x_i, w\right) = \prod_{i=1}^{N} z_i^{y_i}\left(1 - z_i\right)^{1-y_i}$$

Việc tối ưu hàm số vừa nêu trên là rất khó, do đó ta sử dụng phương pháp lấy logarit tự nhiên biến phép nhân thành phép cộng, sau đó lấy ngược dấu để được một hàm và coi nó là hàm mất mát. Lúc này bài toán tìm giá trị lớn nhất trở thành bài toán tìm giá trị nhỏ nhất của hàm mất mát.

$$J\left(w\right) = -\log P\left(y|X, w\right) = -\sum_{i=1}^{N}\left(y_i \log z_i + \left(1 - y_i\right)\log(1 - z_i)\right)$$

Biểu thức vế phải có tên gọi là cross entropy, thường được sử dụng để đo khoảng cách giữa hai phân phối (distributions).

Để tối ưu hàm mất mát trên, ta dùng phương pháp gradient descent. Ta có hàm mất mát với một điểm dữ liệu (x_i, y_i) là

$$J(w; x_i, y_i) = -(y_i \log z_i + (1 - y_i) \log(1 - z_i))$$

và đạo hàm

$$\frac{\partial J(w; x_i, y_i)}{\partial w} = -\left(\frac{y_i}{z_i} - \frac{1 - y_i}{1 - z_i}\right) \frac{\partial z_i}{\partial w} = \frac{z_i - y_i}{z_i(1 - z_i)} \frac{\partial z_i}{\partial w}$$

Chọn hàm $z = f\left(w^T x\right) = \dfrac{e^x}{1 + e^x}$ (còn gọi là hàm sigmoid), khi đó

$$\frac{\partial J(w; x_i, y_i)}{\partial w} = (z_i - y_i) x_i$$

.

Như vậy công thức cập nhật gradient là

$$w = w + \eta (y_i - z_i) x_i$$

3.4.3 Ưu - nhược điểm

Ưu điểm

- Logistic Regression dễ hiểu và dễ triển khai, phù hợp cho các bài toán phân loại cơ bản.
- Có hiệu suất tính toán cao và hoạt động tốt với các tập dữ liệu vừa và nhỏ.

Nhược điểm

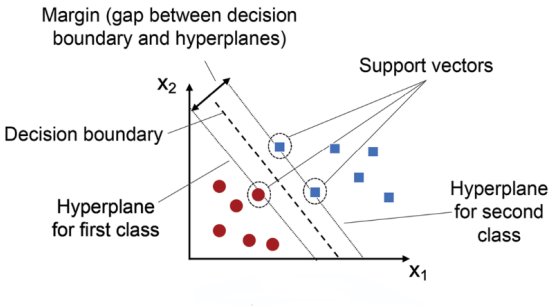
- Logistic Regression giả định mối quan hệ tuyến tính giữa các đặc trưng và biến mục tiêu. Điều này có thể không phù hợp với các dữ liệu có mối quan hệ phi tuyến tính.
- Nó có thể gặp khó khăn với các tập dữ liệu rất lớn hoặc rất phức tạp.
- Logistic Regression có thể gặp vấn đề khi các đặc trưng có mối tương quan cao với nhau, dẫn đến các hệ số ước lượng không ổn định.

3.5 SVM - Support Vector Machine ([5])

3.5.1 Tổng quan

Trước hết, giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai classes này phân chia tuyến tính, tức tồn tại một siêu phẳng phân chia chính xác hai classes đó.

Thuật toán SVM xây dựng một siêu mặt phẳng (hyperplane) phân chia hai classes đó, tức tất cả các điểm thuộc một class nằm về cùng một phía của siêu mặt phẳng đó và ngược phía với toàn bộ các điểm thuộc class còn lại. Siêu mặt phẳng này đóng vai trò như một ranh giới giữa hai lớp sao cho khoảng cách tối thiểu giữa các mẫu thuộc hai lớp khác nhau là lớn nhất. Khoảng cách này còn được gọi là lề (margin), và các điểm dữ liệu có khoảng cách gần lề nhất được gọi là các support vectors.



Hình 5: *Minh họa SVM, nguồn*

3.5.2 Xây dựng bài toán tối ưu cho SVM

Giả sử các cặp dữ liệu của ***training set*** là (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) với các vector $x_i \in R^d$ thể hiện đầu vào của một điểm dữ liệu và y_i là nhãn của dữ liệu đó, d là số chiều của dữ liệu và n là số điểm dữ liệu. Để dễ hình dung, ta xét bài toán trên không gian hai chiều. Bài toán hoàn toàn có thể được tổng quát lên không gian nhiều chiều.

Giả sử hyperplane tối ưu của bài toán là $w^T x + b = 0$. Với mọi cặp dữ liệu (x_i, y_i), khoảng cách từ nó đến mặt phân chia là:

$$\frac{|w^T x_i + b|}{||w||}$$

Tổng quát hơn, khoảng cách từ cặp dữ liệu (x_i, y_i) đến mặt phân chia là:

$$\frac{y_i(w^T x_i + b)}{||w||}$$

Như đã đề cập, hyperplane được xây dựng sao cho margin đạt giá trị lớn nhất. Do đó, bài toán SVM chính là bài toán tìm w và b thỏa điều kiện:

$$(w, b) = arg\max_{w,b} \left\{ \min_i \frac{y_i(w^T x_i + b)}{||w||} \right\} = arg\max_{w,b} \left\{ \frac{1}{||w||} \min_i y_i(w^T x_i + b) \right\} \quad (1)$$

Dễ thấy, nếu thay w và b bằng kw và kb thì giá trị biểu thức không thay đổi. Vì thế không mất tính tổng quát, giả sử với cặp (x_i, y_i) nằm gần mặt phân chia nhất thì $y_i(w^T x_i + b) = 1$. Như vậy, $\forall i \in (1, n)$ ta có:

$$y_i(w^T x_i + b) \geq 1$$

Vậy bài toán (1) có thể được viết lại như sau:

$$(w, b) = arg\max_{w,b} \frac{1}{||w||} = arg\min_{w,b} ||w|| \quad (2)$$

với $y_i(w^T x_i + b) \geq 1 \ \forall i \in (1, n)$

Việc thay $||w||$ ở bài toán (2) bằng $\frac{1}{2}||w||^2$ sẽ không làm thay đổi kết quả của bài toán, khi đó ta

đưa bài toán về

$$(w, b) = arg\min_{w,b} \frac{1}{2}||w||^2 \tag{3}$$

với $y_i(w^T x_i + b) \geq 1 \ \forall i \in (1, n)$

Đây là một bài toán tối ưu bậc hai có ràng buộc và có thể được giải quyết bằng cách thêm các *nhân tử Lagrange* α để tạo ra một bài toán mới

$$\min L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1], a_i \geq 0 \tag{4}$$

Ta sẽ gán giá trị $\alpha = 0$ cho các điểm không phải là support vector (dựa theo điều kiện KKT)
Giải bài toán Lagrange, ta thu được

$$\begin{cases} \dfrac{\partial L}{\partial w} = w - \sum\limits_{i=1}^{n} \alpha_i y_i x_i = 0 \rightarrow w = \sum\limits_{i=1}^{n} \alpha_i y_i x_i \\ \dfrac{\partial L}{\partial b} = \sum\limits_{i=1}^{n} \alpha_i y_i = 0 \end{cases}$$

3.5.3 Support Vector Classifier (SVC)

Support Vector Classifier là phương pháp phân loại dựa trên SVM. Sau khi thực hiện SVM để tìm được siêu mặt phẳng phân chia hai lớp, SVC được sử dụng một cách đơn giản là phân loại dựa trên siêu mặt phẳng này. Các dữ liệu nằm về cùng một phía so với siêu mặt phẳng vừa tìm được thì thuộc về cùng một lớp.

3.5.4 Ưu - nhược điểm

Ưu điểm

- SVM có thể hoạt động tốt trong không gian nhiều chiều. Điều này làm cho SVM trở nên mạnh mẽ đối với dữ liệu có nhiều đặc trưng.
- SVM có thể sử dụng các kernel trick để biến đổi dữ liệu phi tuyến tính thành không gian cao hơn, nơi mà nó có thể được phân loại tuyến tính.
- SVM có thể xử lý dữ liệu nhiễu tốt hơn nhiều thuật toán khác, nhờ vào việc tối ưu hóa biên phân cách.

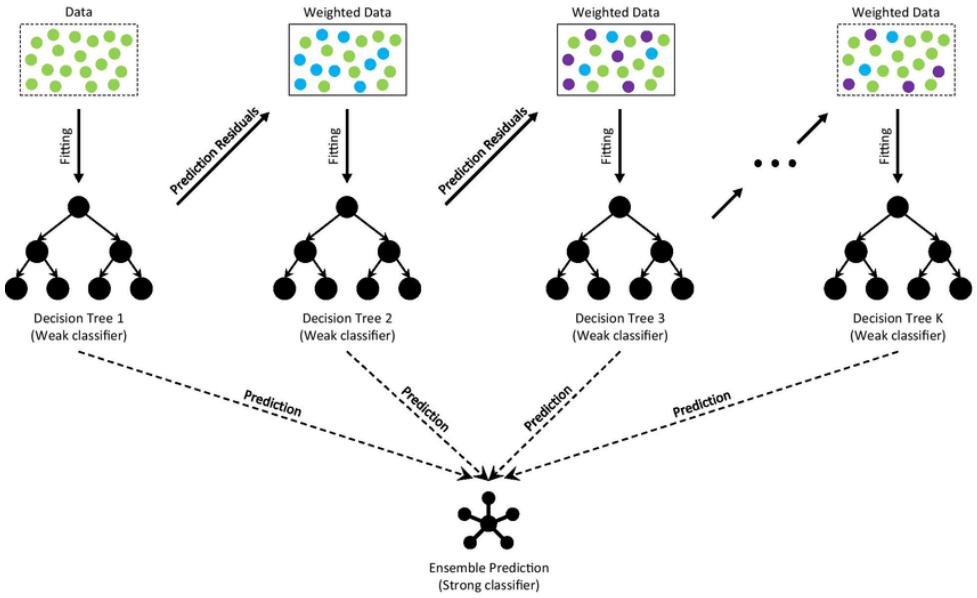
Nhược điểm

- Việc chọn kernel và điều chỉnh các tham số (chẳng hạn như C) là một nhiệm vụ không dễ dàng.
- SVM yêu cầu bộ nhớ và thời gian tính toán tăng lên đáng kể khi số lượng mẫu tăng.

3.6 Gradient Boosting

3.6.1 Tổng quan

Gradient Boosting hoạt động dựa trên ý tưởng của Ensemble Learning, nghĩa là thay vì cố gắng xây dựng một mô hình tốt duy nhất, chúng ta sẽ xây dựng một họ các mô hình yếu hơn (weak learner), nhưng khi kết hợp chúng lại, sẽ thu được một mô hình còn vượt trội hơn cả.



Hình 6: *Minh họa Gradient Boosting, nguồn*

3.6.2 Xây dựng công thức

Gradient Boosting xây dựng thuật toán để giải quyết bài toán tối ưu hàm mất mát như sau

$$min_{c_m, w_m} L(y, F_{m-1} + c_m f_m)$$

Trong đó:

- L: Giá trị loss function
- y: giá trị thực tế
- c_m: Trọng số (confidence score) của weak learner thứ m (hay lần học thứ m)
- f_m: Giá trị dự đoán của weak learner thứ m

Như vậy nếu coi chuỗi các weak learner là một hàm số F thi mỗi hàm dự đoán f là một tham số. Do đó để tìm min chúng ta sẽ áp dụng Gradient Descent để tìm lời giải:

$$F_m = F_{m-1} - \eta \frac{\partial L(y, F_{m-1})}{\partial F_{m-1}}$$

Đến đây ta có thể thấy mối quan hệ liên quan như sau:

$$c_m f_m \approx -\eta \frac{\partial L(y, F_{m-1})}{\partial F_{m-1}}$$

Khởi tạo mô hình với giá trị khởi tạo (giá trị dự đoán ban đầu) là:

$$F_0(x) = arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

Trong đó:

- x: là input data - các feature để dự đoán y
- y_i: là giá trị thực tế

- γ : là giá trị dự đoán
- L: là hàm mất mát

Tùy vào hàm loss function mà giá trị dự đoán ban đầu sẽ khác nhau.
Với mỗi lần học m:

- Tính toán giá trị pseudo-residuals $r_{i,m} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$ cho mỗi dòng dữ liệu thứ i $(1 \leq i \leq N)$
- Giả sử ta có J node lá, vậy ta sẽ có được tập hợp pseudo-residuals $R_{j,m}$ $(1 \leq j \leq J)$. Với mỗi j $(1 \leq j \leq J)$, tính toán giá trị dự đoán

$$\gamma_{j,m} = arg\min_{\gamma} \sum_{x_i \in R_{i,j}} L(y_i, F_{m-1}(x_i) + \gamma)$$

- Cập nhật giá trị của weak learner hiện tại với learning rate η:

$$F_m(x) = F_{m-1}(x) + \eta \sum_{j=1}^{J} \gamma_{j,m}(x \in R_{j,m})$$

$F_m(x)$ chính là output của giá trị dự đoán cho bài toán

3.6.3 Ưu - nhược điểm

Ưu điểm

- Gradient Boosting thường cho kết quả chính xác cao và có thể cạnh tranh với các thuật toán học máy tiên tiến khác như Random Forest và SVM.
- Gradient Boosting có thể tự động xác định và chọn ra các đặc trưng quan trọng nhất.
- Nhờ vào việc xây dựng các weak learner tuần tự, Gradient Boosting có khả năng nắm bắt và mô hình hóa các mối quan hệ phi tuyến trong dữ liệu.

Nhược điểm

- Gradient Boosting đòi hỏi nhiều tài nguyên tính toán và thời gian để huấn luyện, đặc biệt là với các tập dữ liệu lớn.
- Gradient Boosting yêu cầu điều chỉnh nhiều siêu tham số (như tốc độ học, số lượng cây, độ sâu tối đa của cây), điều này có thể phức tạp và tốn thời gian.
- Nếu không được điều chỉnh cẩn thận, Gradient Boosting dễ bị overfitting.

4 Các phương pháp đánh giá ([6])

Khi xây dựng một mô hình Machine Learning, chúng ta cần những 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).

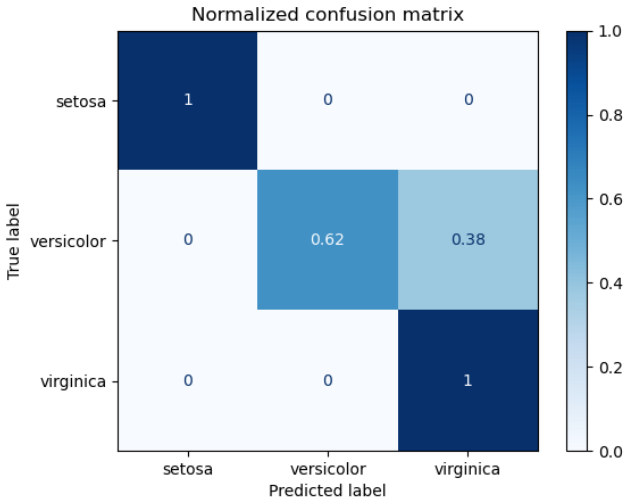
Với một mô hình phân lớp, có rất nhiều cách đánh giá. Tuỳ vào những bài toán khác nhau mà chúng ta sử dụng các phương pháp khác nhau. Các phương pháp được sử dụng là accuracy score, confusion matrix, ROC curve, Area Under the Curve, Precision and Recall, F1 score.

4.1 Accuracy Score

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

4.2 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.



Hình 7: *Minh họa Confusion Matrix, nguồn*

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.

True/False Positive/Negative: Đây cũng là một confusion matrix với bài toán phân loại chỉ có hai lớp. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ, trong bài toán dự báo ngày mai có mưa hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm trời không mưa thành trời mưa.

Trong những bài toán này, người ta thường định nghĩa lớp dữ liệu quan trọng hơn cần được xác định đúng là lớp Positive (P-dương tính), lớp còn lại được gọi là Negative (N-âm tính).

Người ta thường quan tâm đến TPR, FNR, FPR, TNR (R - Rate), cụ thể

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{TNR} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

False Positive Rate (FPR) còn được gọi là False Alarm Rate (tỉ lệ báo động nhầm), False Negative Rate (FNR) còn được gọi là Miss Detection Rate (tỉ lệ bỏ sót). Trong những bài toán dự báo như vậy, ta có thể chấp nhận False Alarm Rate cao để đạt được Miss Detection Rate thấp.

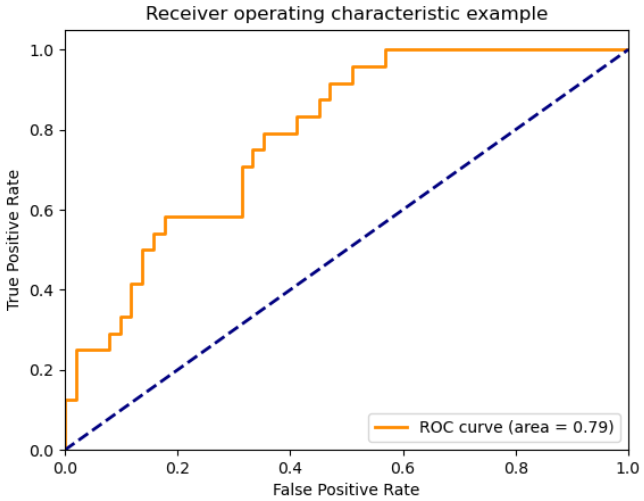
Với bài toán nhiều lớp dữ liệu, vẫn có thể xây dựng bảng True/False Positive/Negative cho mỗi lớp nếu coi lớp đó là lớp Positive, các lớp còn lại gộp chung thành lớp Negative.

4.3 Receiver Operating Characteristic curve - ROC curve

Trong một số bài toán, việc tăng hay giảm FNR, FPR có thể được thực hiện bằng việc thay đổi một ngưỡng (threshold) nào đó. Nếu bây giờ coi lớp 1 là lớp Positive, lớp 0 là lớp Negative, ta cần tăng FPR, và vì FPR + FNR = 1, nên việc này cũng giảm FNR.

Một kỹ thuật đơn giản là ta thay giá trị threshold xuống một số nhỏ hơn. Khi đó, tỉ lệ các điểm được phân loại là Positive sẽ tăng lên, kéo theo cả FPR và TPR cùng tăng lên, suy ra cả FNR và TNR đều giảm.

Như vậy, ứng với mỗi giá trị của threshold, ta sẽ thu được một cặp (FPR, TPR). Biểu diễn các điểm này trên đồ thị khi thay đổi threshold từ 0 tới 1 (hoặc trong một khoảng khác) ta sẽ thu được một đường được gọi là Receiver Operating Characteristic curve hay ROC curve.



Hình 8: *Minh họa ROC curve, nguồn*

Dựa trên ROC curve, ta có thể chỉ ra rằng một mô hình có hiệu quả hay không. Một mô hình hiệu quả khi có FPR thấp và TPR cao, tức tồn tại một điểm trên ROC curve gần với điểm có toạ độ (0, 1)

trên đồ thị (góc trên bên trái). Curve càng gần thì mô hình càng hiệu quả. Từ đó, ta cũng thu được threshold tối ưu cho mô hình của mình.

Có một thông số nữa dùng để đánh giá gọi là Area Under the Curve hay AUC. Đại lượng này chính là diện tích nằm dưới ROC curve. Giá trị này là một số dương nhỏ hơn hoặc bằng 1. Giá trị này càng lớn thì mô hình càng tốt.

4.4 Precision-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.

Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive.

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

Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive.

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

Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. 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.

Recall cao đồng nghĩa tỉ lệ bỏ sót các điểm thực sự positive là thấp. 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. Một cách đo chất lượng của bộ phân lớp dựa vào Precision và Recall phổ biến là F-score.

F-score F1-score, là harmonic mean của precision và recall, với giả sử chúng đều khác 0,

$$\text{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

F1-score có giá trị nằm trong nửa khoảng $(0; 1]$. F1-score càng cao, bộ phân lớp càng tốt.

Trường hợp tổng quát của F1-score là Fβ-score

$$\text{F}\beta\text{-score} = \frac{(1+\beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$$

4.5 Jaccard Index

Jaccard Index là 1 giải thuật được đưa ra bởi nhà Toán học Pháp Paul Jaccard. Giải thuật này là kết quả tính toán độ tương đồng giữa giao của 2 tập hợp và hợp của 2 tập hợp,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Đây là thước đo mức độ tương tự của hai bộ dữ liệu, với phạm vi từ 0 đến 1. Giá trị càng cao thì hai bộ dữ liệu càng giống nhau. Jaccard Index cực kỳ nhạy cảm với kích thước mẫu nhỏ.

5 Hiện thực mô hình dự đoán

Đường dẫn đến toàn bộ source code và data: github

5.1 Giới thiệu tập dữ liệu

Bài viết này sử dụng hai tập dữ liệu, cả hai cùng được cung cấp bởi Cục Khí tượng của Chính phủ Úc (Australian Government's Bureau of Meteorology).

- https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package: một file *.csv* về dữ liệu lượng mưa hằng ngày ở tất cả các thành phố của nước Úc từ năm 2008 đến năm 2017. Ta gọi là **tập dữ liệu 1**.
- http://www.bom.gov.au/climate/dwo/IDCJDW2124.latest.shtml: 14 file *.csv* về dữ liệu lượng mưa hằng ngày ở thành phố Sydney trong 13 tháng gần nhất, nghĩa là từ 5/2023 đến 5/2024. Ta gọi là **tập dữ liệu 2**.

Tuy có khác biệt về tên gọi của các cột, nhưng cả hai tập dữ liệu này đều có các cột mang ý nghĩa ([1]) như sau:

Tên cột	Ý nghĩa	Đơn vị đo
Date	Ngày cụ thể	
MinTemp	Nhiệt độ thấp nhất trong ngày	Độ Celsius
MaxTemp	Nhiệt độ cao nhất trong ngày	Độ Celsius
Rainfall	Lượng mưa trong ngày	mm
Evaporation	Lượng bốc hơi	mm
Sunshine	Lượng ánh sáng	Giờ
WindGustDir	Hướng gió mạnh nhất	Hướng trên la bàn
WindGustSpeed	Vận tốc của cơn gió mạnh nhất	km/h
WindDir9am	Hướng gió trung bình 10 phút trước 9h	Hướng trên la bàn
WindDir3pm	Hướng gió trung bình 10 phút trước 15h	Hướng trên la bàn
WindSpeed9am	Vận tốc gió trung bình 10 phút trước 9h	km/h
WindSpeed3pm	Vận tốc gió trung bình 10 phút trước 15h	km/h
Humidity9am	Độ ẩm lúc 9h	%
Humidity3pm	Độ ẩm lúc 15h	%
Pressure9am	Áp suất khí quyển trung bình so với mực nước biển lúc 9h	hPa
Pressure3pm	Áp suất khí quyển trung bình so với mực nước biển lúc 15h	hPa
Cloud9am	Phần bầu trời bị mây che khuất lúc 9h	1/8
Cloud3pm	Phần bầu trời bị mây che khuất lúc 15h	1/8
Temp9am	Nhiệt độ lúc 9h	Độ Celsius
Temp3pm	Nhiệt độ lúc 15h	Độ Celsius

Bảng 1: *Ý nghĩa các cột trong tập dữ liệu*

Ngoài ra, mỗi tập dữ liệu có các cột riêng như sau:

- Tập dữ liệu 1:

Tên cột	Ý nghĩa	Đơn vị đo
Location	Địa điểm	
RainToday	Hôm nay có mưa hay không	Yes/No
RainTomorrow	Ngày mai có mưa hay không	Yes/No

Bảng 2: *Ý nghĩa các cột trong tập dữ liệu 1*

- Tập dữ liệu 2:

Tên cột	Ý nghĩa	Đơn vị đo
Time of maximum wind gust	Thời gian xảy ra cơn gió mạnh nhất	giờ:phút

Bảng 3: *Ý nghĩa các cột trong tập dữ liệu 2*

5.2 Tiền xử lý dữ liệu

5.2.1 Đọc tập dữ liệu

Để huấn luyện mô hình một cách dễ dàng về sau, nhiệm vụ đầu tiên là phải kết hợp được tất cả các file *.csv* lại với nhau và đọc vào một biến duy nhất, gọi là *output_df*.

Như đã đề cập, hai tập dữ liệu có một ít sự khác biệt về tên gọi của các cột, vì thế vấn đề này được xử lý như sau.

Import thư viện, khai báo các biến chứa tên file, chứa các cột cần thay đổi tên.

```
import pandas as pd

input_file_names = [
    "IDCJDW2124.202305", "IDCJDW2124.202306", "IDCJDW2124.202307", "IDCJDW2124.202308",
        "IDCJDW2124.202309",
    "IDCJDW2124.202310", "IDCJDW2124.202311", "IDCJDW2124.202312", "IDCJDW2124.202401",
        "IDCJDW2124.202402",
    "IDCJDW2124.202403", "IDCJDW2124.202404", "IDCJDW2124.202405"
]
base_path = "data/"
rename_dict = {
          'Date': 'Date',
          'Time of maximum wind gust': 'TimeMaxWindGust',
          'Minimum temperature (C)': 'MinTemp',
          'Maximum temperature (C)': 'MaxTemp',
          'Rainfall (mm)': 'Rainfall',
          'Evaporation (mm)': 'Evaporation',
          'Sunshine (hours)': 'Sunshine',
          'Direction of maximum wind gust ': 'WindGustDir',
          'Speed of maximum wind gust (km/h)': 'WindGustSpeed',
          '9am Temperature (C)': 'Temp9am',
          '9am relative humidity (%)': 'Humidity9am',
          '9am cloud amount (oktas)': 'Cloud9am',
          '9am wind direction': 'WindDir9am',
          '9am wind speed (km/h)': 'WindSpeed9am',
          '9am MSL pressure (hPa)': 'Pressure9am',
          '3pm Temperature (C)': 'Temp3pm',
          '3pm relative humidity (%)': 'Humidity3pm',
          '3pm cloud amount (oktas)': 'Cloud3pm',
          '3pm wind direction': 'WindDir3pm',
          '3pm wind speed (km/h)': 'WindSpeed3pm',
          '3pm MSL pressure (hPa)': 'Pressure3pm'
}
```

Đọc dữ liệu từ các file (có xử lý ngoại lệ về vấn đề encoding), tiến hành đổi tên các cột, và lưu vào file *.csv* mới. Đây chính là **tập dữ liệu 2**.

```
combined_df = pd.DataFrame()

for file_name in input_file_names:
    input_file = f"{base_path}{file_name}.csv"
    temp_output_file = f"{base_path}{file_name}.cleaned.csv"

    try:
        with open(input_file, 'r', encoding='ISO-8859-1') as infile, open(temp_output_file, 'w',
             encoding='utf-8') as outfile:
            for line in infile:
                fields = line.strip().split(',')
                if len(fields) == 22:
                    outfile.write(','.join(fields[1:]) + '\n')
    except UnicodeDecodeError as e:
        print(f"UnicodeDecodeError: {e}")
        try:
            with open(input_file, 'r', encoding='cp1252') as infile, open(temp_output_file, 'w',
                 encoding='utf-8') as outfile:
                for line in infile:
                    fields = line.strip().split(',')
                    if len(fields) == 22:
                        outfile.write(','.join(fields[1:]) + '\n')
        except UnicodeDecodeError as e:
            print(f"UnicodeDecodeError: {e}")

    df = pd.read_csv(temp_output_file, encoding='utf-8')

    for old_name, new_name in rename_dict.items():
        df.rename(columns={old_name: new_name}, inplace=True)

    df.drop(columns=['TimeMaxWindGust'], inplace=True)

    df.to_csv(temp_output_file, index=False)

    combined_df = pd.concat([combined_df, df], ignore_index=True)

combined_df['RainToday'] = combined_df['Rainfall'].apply(lambda x: 'Yes' if x > 0 else 'No')
combined_df['RainTomorrow'] = combined_df['Rainfall'].shift(-1).apply(lambda x: 'Yes' if x > 0
     else 'No')
combined_df.to_csv("data/combined.csv", index=False)
```

Đối với **tập dữ liệu 1**, tập dữ liệu này chứa nhiều dữ liệu ở các thành phố khác nhau, nhưng như đã nói, ta chỉ quan tâm đến thành phố Sydney.

```
other_df = pd.read_csv("data/weatherAUS.csv")
other_df = other_df[other_df['Location'] == 'Sydney']
other_df.drop(columns=['Location'], inplace=True)
```

Cuối cùng, kết hợp 2 tập dữ liệu lại, đồng thời cũng ghi vào một file *.csv* để lưu trữ.

```
output_df = pd.concat([other_df, combined_df], ignore_index=True)
output_file = "data/output.csv"
output_df.to_csv(output_file, index=False)
```

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Humidity9am	Humidity3pm	Pre
0	2008-02-01	19.5	22.4	15.6	6.2	0.0	NaN	NaN	S	SSW	...	92.0	84.0	
1	2008-02-02	19.5	25.6	6.0	3.4	2.7	NaN	NaN	W	E	...	83.0	73.0	
2	2008-02-03	21.6	24.5	6.6	2.4	0.1	NaN	NaN	ESE	ESE	...	88.0	86.0	
3	2008-02-04	20.2	22.8	18.8	2.2	0.0	NaN	NaN	NNE	E	...	83.0	90.0	
4	2008-02-05	19.7	25.7	77.4	NaN	0.0	NaN	NaN	NNE	W	...	88.0	74.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3736	2024-05-27	10.2	21.3	0.0	3.0	9.5	W	30.0	W	S	...	76.0	51.0	
3737	2024-05-28	9.7	22.4	0.0	2.4	9.1	WNW	26.0	W	E	...	82.0	58.0	
3738	2024-05-29	9.4	21.3	0.0	1.4	9.3	WNW	22.0	WNW	ENE	...	87.0	68.0	
3739	2024-05-30	10.8	24.2	0.0	2.6	8.3	NNW	31.0	WNW	NNE	...	90.0	48.0	
3740	2024-05-31	14.2	20.8	0.0	4.4	0.0	NNW	43.0	N	N	...	60.0	65.0	

3741 rows × 22 columns

Hình 9: *Tập dữ liệu cuối cùng cần xử lý*

5.2.2 Tổng quát dữ liệu

Về kích thước

```
print(combined_df.shape)
print(other_df.shape)
print(output_df.shape)
```

Kết quả:

```
(397, 22)
(3344, 22)
(3741, 22)
```

Về các giá trị categorical

```
data_types = output_df.dtypes
categorical_columns = data_types[data_types == 'object'].index
for col in categorical_columns:
    unique_values = output_df[col].unique()
    unique_values_count = len(unique_values)
    print(f"Column '{col}' is categorical with {unique_values_count} unique values.")
    if col != 'Date':
        for val in unique_values:
            print(val, end=", ")
```

```
    print("\n")
```

Kết quả:

```
Column 'Date' is categorical with 3741 unique values.


Column 'WindGustDir' is categorical with 17 unique values.
nan, SSE, ESE, ENE, NNE, NNW, WNW, W, NW, SSW, NE, WSW, SE, SW, N, E, S,

Column 'WindDir9am' is categorical with 17 unique values.
S, W, ESE, NNE, SSW, WNW, N, SW, SE, SSE, WSW, E, nan, ENE, NW, NNW, NE,

Column 'WindDir3pm' is categorical with 18 unique values.
SSW, E, ESE, W, ENE, S, SE, SSE, NE, NNE, nan, NNW, NW, WNW, N, WSW, SW, ,

Column 'WindSpeed3pm' is categorical with 45 unique values.
20.0, 13.0, 2.0, 6.0, 22.0, 15.0, 9.0, 26.0, 24.0, 30.0, 19.0, 17.0, 28.0, 31.0, nan, 4.0, 7.0,
    11.0, 41.0, 39.0, 37.0, 35.0, 44.0, 33.0, 50.0, 48.0, 43.0, 0.0, 46.0, 57.0, 11, 17, 35,
    13, 22, 30, 24, 20, 19, 15, 9, Calm, 39, 7, 2,

Column 'RainToday' is categorical with 3 unique values.
Yes, No, nan,

Column 'RainTomorrow' is categorical with 3 unique values.
Yes, No, nan,
```

Về các giá trị khuyết

```
print(output_df.isna().sum())
```

Kết quả:

```
Date              0
MinTemp           5
MaxTemp           7
Rainfall         12
Evaporation      72
Sunshine         21
WindGustDir    1045
WindGustSpeed  1045
WindDir9am       56
WindDir3pm       33
WindSpeed9am     26
WindSpeed3pm     25
Humidity9am      16
Humidity3pm      14
Pressure9am      21
Pressure3pm      19
Cloud9am        568
```

```
Cloud3pm        563
Temp9am           5
Temp3pm           5
RainToday         7
RainTomorrow      7
dtype: int64
```

5.2.3 Làm sạch dữ liệu

Trước hết, tạo một bản sao cho *output_ df* để thuận tiện hơn trong quá trình xử lý (nếu gặp lỗi có thể quay về bước này để tránh việc đọc lại tập dữ liệu ở bước trên).

```
df = output_df.copy()
```

Có 2 vấn đề cần quan tâm với tập dữ liệu trong bài toán này, là xử lý giá trị khuyết và xử lý giá trị categorical. Hai vấn đề này cũng có sự liên quan với nhau.

- Với giá trị khuyết, nhận thấy số lượng khuyết của cột WindGustDir, WindGustSpeed ($1045/3741 \approx 28\%$), Cloud9am ($568/3741 \approx 15\%$) và Cloud3pm ($563/3741 \approx 15\%$) là đáng kể so với tổng số lượng dữ liệu, vì thế giải pháp loại bỏ tất cả dữ liệu khuyết sẽ không phù hợp. Do đó, giá trị khuyết cần được xem xét kỹ càng hơn ở từng đặc trưng.
- Giá trị categorical chỉ có ở các cột WindGustDir, WindDir9am, WindDir3pm, WindSpeed3pm, RainToday, RainTomorrow. Trong đó:
  - cột WindGustDir chứa rất nhiều khuyết như đã đề cập
  - cột WindSpeed3pm chứa rất nhiều giá trị số thực nhưng lại có một giá trị 'Calm'
  - cột RainToday và RainTomorrow ngoài số ít giá trị bị khuyết thì chỉ có giá trị Yes hoặc No

Vì hai vấn đề vừa đề cập có liên quan đến nhau, nên chúng không thể được xử lý riêng lẻ từng bước một. Bài viết sẽ trình bày theo từng cột, từ đơn giản đến phức tạp:

1. RainToday, RainTomorrow, Rainfall

Như đã đề cập, vấn đề cần quan tâm trong bài viết này là dựa vào thông tin trước đó để dự đoán ngày mai có mưa hay không. Do đó, để quá trình huấn luyện có kết quả chính xác hơn, các dữ liệu nếu bị khuyết ở đặc trưng RainToday hoặc RainTomorrow hoặc Rainfall sẽ bị loại bỏ.

```
df.dropna(subset=['RainToday', 'RainTomorrow', 'Rainfall'], inplace=True)
```

Đồng thời cũng chuyển các giá trị No và Yes ở cột RainToday, RainTomorrow về số 0 và 1 tương ứng.

```
df.replace(['No', 'Yes'], [0,1], inplace=True)
```

2. WindSpeed3pm

Theo National Weather Service ([3]), 'Calm' là điều kiện thời tiết khi không phát hiện thấy chuyển động của không khí. Vì vậy, có thể hiểu khi thời tiết 'Calm' tức là vận tốc của gió là 0 km/h. Do đó, ta thay thế 'Calm' bởi giá trị 0.0, đồng thời chuyển cột này thành dạng *float*, vì trước đó cột này đã bị định dạng là categorical.

```
df['WindSpeed3pm'].replace('Calm', 0.0, inplace=True)
df['WindSpeed3pm'] = df['WindSpeed3pm'].astype(float)
```

3. Cloud9am, Cloud3pm, WindGustSpeed

Đây là các cột có giá trị số thực và chứa nhiều khuyết. Vì các cột này chứa nhiều khuyết, nên cách xử lý đầu tiên có thể được xem xét là loại bỏ luôn cột này.

Để quyết định xem cột có bị loại bỏ hay không, hệ số tương quan Pearson được áp dụng, với ngưỡng loại bỏ được chọn ngẫu nhiên là dưới 0.3.

```
cloud9am_correlation_coefficient = df['Cloud9am'].corr(df['RainTomorrow'])
print(f"Correlation between Cloud9am and RainTomorrow: {correlation}")

cloud3pm_correlation_coefficient = df['Cloud3pm'].corr(df['RainTomorrow'])
print(f"Correlation between Cloud3pm and RainTomorrow: {cloud3pm_correlation_coefficient}")

windgustspeed_correlation_coefficient = df['WindGustSpeed'].corr(df['RainTomorrow'])
print(f"Correlation between WindGustSpeed and RainTomorrow:
    {windgustspeed_correlation_coefficient}")
```

Kết quả:

```
Correlation between Cloud9am and RainTomorrow: 0.3727035136751936
Correlation between Cloud3pm and RainTomorrow: 0.45764003475226217
Correlation between WindGustSpeed and RainTomorrow: 0.18442526352429942
```

Như vậy, chỉ có cột WindGustSpeed là bị loại bỏ. Hai cột còn lại sẽ được xử lý ở bước tiếp theo.

```
df.drop(columns=['WindGustSpeed'], inplace=True)
```

4. MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm, Temp9am, Temp3pm

Đây là các cột có giá trị số thực và chứa ít khuyết. Với các giá trị như thế này, cách phổ biến và đơn giản nhất là điền các giá trị bị khuyết bằng trung bình cộng của các giá trị không bị khuyết. Hơn nữa, vì đây là dữ liệu đã được phân chia rõ ràng về ngày tháng nên để chi tiết hơn, các giá trị bị khuyết sẽ được tính bằng **trung bình cộng** của các giá trị không bị khuyết *trong cùng một tháng*. Riêng đối với **MinTemp** và **MaxTemp**, như tên gọi của cột, các giá trị bị khuyết thay vì được tính bằng trung bình cộng sẽ được tính bằng **giá trị nhỏ nhất và lớn nhất** của các giá trị không bị khuyết **trong cùng một tháng**.

```
df['Date'] = pd.to_datetime(df['Date'])

df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year

max_temp_by_month_year = df.groupby(['Year', 'Month'])[['MaxTemp']].max()
min_temp_by_month_year = df.groupby(['Year', 'Month'])[['MinTemp']].min()
mean_evaporation_by_month_year = df.groupby(['Year', 'Month'])[['Evaporation']].mean()
```

```
# rest of the code

def fillna_with_group_month(row, column_name, groupby_values):
    if pd.isna(row[column_name]):
        row[column_name] = groupby_values.loc[row['Year'], row['Month']][column_name]
    return row

df[['MaxTemp']] = df.apply(fillna_with_group_month, args=('MaxTemp', max_temp_by_month_year),
    axis=1)[['MaxTemp']]
df[['MinTemp']] = df.apply(fillna_with_group_month, args=('MinTemp', min_temp_by_month_year),
    axis=1)[['MinTemp']]
df[['Evaporation']] = df.apply(fillna_with_group_month, args=('Evaporation',
    mean_evaporation_by_month_year), axis=1)[['Evaporation']]
# rest of the code

df.drop(columns=['Month', 'Year'], inplace=True)
```

Đến đây, kiểm tra thì lại có vấn đề phát sinh

```
print(df.isna().sum())
```

Kết quả

```
Date             0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustDir   1041
WindDir9am      56
WindDir3pm      33
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am       517
Cloud3pm       517
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

Cột Cloud9am và Cloud3pm vẫn chưa được xử lý hoàn chỉnh. Một lần nữa xem xét lại hai cột này.

```
df_cloud9am_na = df[df['Cloud9am'].isna()]
print(df_cloud9am_na)
```

Kết quả cho thấy, không thể điền khuyết được là vì trong cùng một tháng đó, không có giá trị nào của Cloud9am và Cloud3pm được đo đạc. Vì vậy riêng với các giá trị này, không thể điền khuyết theo cách tính trung bình cộng của các giá trị không bị khuyết trong cùng một tháng, phải điền theo cách tính trung bình cộng của tất cả các giá trị không bị khuyết.

```
df['Cloud9am'] = df['Cloud9am'].fillna(df['Cloud9am'].mean())
df['Cloud3pm'] = df['Cloud3pm'].fillna(df['Cloud3pm'].mean())
```

5. WindGustDir, WindDir9am, WindDir3pm

Đây là các cột có giá trị categorical và chứa nhiều khuyết. Vì các cột này chứa nhiều khuyết, nên cách xử lý đầu tiên có thể được xem xét là loại bỏ luôn cột này.

Để quyết định xem cột có bị loại bỏ hay không, Cramer's V được áp dụng, với ngưỡng loại bỏ được chọn ngẫu nhiên là dưới 0.1.

```
import scipy.stats as ss
import numpy as np

def cramers_v(contingency_table):
    chi2 = ss.chi2_contingency(contingency_table)[0]
    n = contingency_table.sum().sum()
    min_dim = min(contingency_table.shape) - 1
    return np.sqrt(chi2 / (n * min_dim))

print(f"Cramer's V between WindGustDir and RainTomorrow:
    {cramers_v(pd.crosstab(df['WindGustDir'], df['RainTomorrow']))}")
print(f"Cramer's V between WindDir9am and RainTomorrow:
    {cramers_v(pd.crosstab(df['WindDir9am'], df['RainTomorrow']))}")
print(f"Cramer's V between WindDir3pm and RainTomorrow:
    {cramers_v(pd.crosstab(df['WindDir3pm'], df['RainTomorrow']))}")
```

Kết quả

```
Cramer's V between WindGustDir and RainTomorrow: 0.3288554557950443
Cramer's V between WindDir9am and RainTomorrow: 0.23749997584723245
Cramer's V between WindDir3pm and RainTomorrow: 0.3170455218876473
```

Như vậy, không có cột nào bị loại bỏ.

Do đó, với các cột có giá trị categorical này, có hai hướng xử lý.

- **One-Hot Encoding**: chuyển đổi mỗi giá trị trong biến categorical thành một vector đặc biệt. Mỗi giá trị được biểu diễn dưới dạng một vector có chiều dài bằng số lượng các giá trị unique trong biến. Trong vector này, chỉ có một phần tử có giá trị 1 (biểu thị giá trị đó), các phần tử còn lại có giá trị 0. Thông thường, One-Hot Encoding được sử dụng khi đặc trưng không có tính chất thứ tự hoặc thứ hạng.
- **Label Encoding**: là quá trình chuyển đổi các giá trị categorical thành các giá trị số tương ứng. Mỗi giá trị trong biến categorical được gán một số nguyên duy nhất. Thông thường, Label Encoding được sử dụng khi đặc trưng có tính chất thứ tự hoặc thứ hạng.

Hiện tại đang có 3 đặc trưng categorical là WindGustDir, WindDir9am và WindDir3pm. Nhưng

như theo mô tả dữ liệu, chúng đều biểu thị hướng gió, nên việc xử lý 3 đặc trưng này tương tự nhau. Trong bài toán này, one-hot encoding được chọn vì các tính chất về hướng gió là như nhau. Trong Pandas, hàm *pd.get_ dummies* thường được sử dụng để thực hiện one-hot encoding.

```
df = pd.get_dummies(data=df, columns=['RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm'])
```

6. Kiểm tra lần cuối

```
print(df.isna().sum())
data_types = df.dtypes
categorical_columns = data_types[data_types == 'object'].index
if len(categorical_columns) == 0:
    print("\nNo categorical anymore")
```

Kết quả

```
Date            0
MinTemp         0
MaxTemp         0
Rainfall        0
Evaporation     0
               ..
WindDir3pm_SSW  0
WindDir3pm_SW   0
WindDir3pm_W    0
WindDir3pm_WNW  0
WindDir3pm_WSW  0
Length: 68, dtype: int64

No categorical anymore
```

5.2.4 Loại bỏ outliers với Z-score

Tính Z-score là một phương pháp thống kê được sử dụng để đo lường khoảng cách của một giá trị dữ liệu đến giá trị trung bình của dữ liệu, được đo lường theo đơn vị độ lệch chuẩn. Sử dụng Z-score, chúng ta có thể xác định được xem một giá trị dữ liệu cụ thể có nằm ngoài phạm vi bình thường hay không.

- **Bước 1: Tính Giá Trị Trung Bình và Độ Lệch Chuẩn**

  Trước tiên, ta tính giá trị trung bình (μ) và độ lệch chuẩn (σ) của dữ liệu trong một cột cụ thể.

- **Bước 2: Tính Z-score Cho Mỗi Giá Trị Dữ Liệu:**

  Sau đó, ta tính Z-score cho mỗi giá trị dữ liệu (x_i) trong cột bằng cách sử dụng công thức:

$$Z = \frac{|x_i - \mu|}{\sigma}$$

  Giá trị Z-score thể hiện khoảng cách của giá trị x_i so với giá trị trung bình của cột, được đo lường trong đơn vị độ lệch chuẩn.

- **Bước 3: Xác Định Ngưỡng Z-score Để Xác Định Outlier**

  Ngưỡng Z-score được sử dụng để quyết định liệu một giá trị dữ liệu có được coi là outlier hay không. Nếu giá trị Z-score của một điểm dữ liệu vượt quá ngưỡng, điểm dữ liệu đó được xem xét là outlier.

- **Bước 4: Loại Bỏ Các Dòng Chứa Outlier:**

  Việc này giúp loại bỏ các điểm dữ liệu không thường xuyên và không đại diện cho xu hướng chung của dữ liệu, giúp cải thiện chất lượng của dữ liệu trong quá trình xử lý và phân tích.

```
from scipy.stats import zscore
for column in temp.select_dtypes(include='number').columns:
    if column != 'RainTomorrow':
        z_scores = zscore(df[column])
        threshold = 6
        df = df[(abs(z_scores) <= threshold)]
df.reset_index(drop=True, inplace=True)
df.shape
```

Kết quả

```
(3698, 68)
```

Như vậy, có thể kết luận có $3723 - 3698 = 25$ outlier, và chúng đã bị loại bỏ.

5.2.5 Các bước khác

1. Loại bỏ cột Date

Dễ thấy đặc trưng Date không có ý nghĩa trong việc dự đoán ngày mai trời có mưa hay không, vì vậy việc loại bỏ đặc trưng này là cần thiết.

```
df.drop('Date',axis=1,inplace=True)
```

2. Chuyển về dạng số thực

Vì hiện tại trong dữ liệu vẫn còn dạng boolean, không thể sử dụng cho việc huấn luyện, việc đưa chúng về giá trị 0 và 1 là cần thiết.

```
df = df.astype(float)
```

5.2.6 Chia tách dữ liệu huấn luyện và kiểm thử

1. Trích xuất các features vào hai biến X và y

Đầu tiên, trích xuất các features vào hai biến features và y.

- X là ma trận các features có liên quan đến mô hình gồm 66 đặc trưng.
- y là biến phụ thuộc cần dự đoán trong bài toán này, là RainTomorrow.

```
X = df.drop(columns='RainTomorrow', axis=1)
y = df['RainTomorrow']
```

```
print(features.shape) # (3698, 66)
print(y.shape) # (3698,)
```

2. Chia tách dữ liệu huấn luyện và kiểm thử

Ta sử dụng hàm train_test_split từ thư viện scikit-learn để chia dữ liệu thành hai phần: một tập dữ liệu huấn luyện (X_train, y_train) và một tập dữ liệu kiểm tra (X_test, y_test).

- X là ma trận chứa các features của dữ liệu.
- y là vector chứa các nhãn hoặc giá trị mục tiêu tương ứng với từng mẫu dữ liệu trong X.
- Tham số test_size=0.2 xác định tỷ lệ của dữ liệu được sử dụng cho việc kiểm tra.

  Trong trường hợp này, 20% dữ liệu được lưu vào X_test và y_test sử dụng cho việc kiểm tra; 80% được lưu vào X_train và y_train sử dụng cho việc huấn luyện.
- Tham số random_state=10 đặt giá trị seed cho việc tạo ngẫu nhiên các tỷ lệ chia dữ liệu. Việc này đảm bảo rằng mỗi khi chạy đoạn code, việc chia dữ liệu sẽ được thực hiện theo cùng một cách, giúp tái lập được kết quả và so sánh hiệu suất giữa các lần chạy.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
```

5.3 Hiện thực áp dụng mô hình

5.3.1 Chiến lược chung

Thực hiện theo từng thuật toán đã đề cập ở các phần trước, theo các bước như sau

1. Hiển nhiên bước đầu tiên là import thư viện. Trong bài viết này, ta sử dụng thư viện *scikit-learn* là chủ yếu.
2. Sử dụng thư viện để khởi tạo mô hình theo thuật toán đang chạy.
3. Sử dụng mô hình đã khởi tạo để huấn luyện.
4. Đối với những thuật toán dự đoán dựa trên xác suất (logistic regression, KNN,...), sử dụng thêm ROC curve để tìm kiếm được threshold dự báo tốt nhất (threshold tốt nhất là giá trị làm cho (FPR; TPR) gần với điểm (0;1) nhất).

   Đối với những thuật toán chỉ dự đoán nhị phân (decision tree, SVM,...), không làm gì thêm ở bước này.
5. Dùng threshold vừa tìm được (hoặc 0.5 theo mặc định) để đưa ra dự đoán
6. Tính các metrics cần thiết
7. Dựa trên metrics vừa tìm được, đưa ra những nhận xét và đánh giá về thuật toán được áp dụng

Sau khi đã xác định được các bước, ta bắt đầu thực hiện import một số thư viện cần thiết

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

Để tìm được threshold tốt nhất, ta cũng định nghĩa một phương thức để tìm threshold này

```
def plot_roc_curve(y_true, y_probabilities, model_name='Model'):
    fpr, tpr, thresholds = roc_curve(y_true, y_probabilities)
    roc_auc = auc(fpr, tpr)
    distances_to_corner = np.sqrt((fpr - 0)**2 + (tpr - 1)**2)
    optimal_idx = np.argmin(distances_to_corner)
    optimal_threshold = thresholds[optimal_idx]

    plt.figure(figsize=(5, 4))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'{model_name} ROC curve (area =
        {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.scatter(fpr[optimal_idx], tpr[optimal_idx], marker='o', color='red', label=f'Optimal
        Threshold = {optimal_threshold:.2f}')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{model_name} Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.show()

    print(f'Optimal threshold that minimizes distance to (0, 1) in ROC space:
        {optimal_threshold}')

    return optimal_threshold
```

5.3.2 KNN

Khởi tạo và huấn luyện

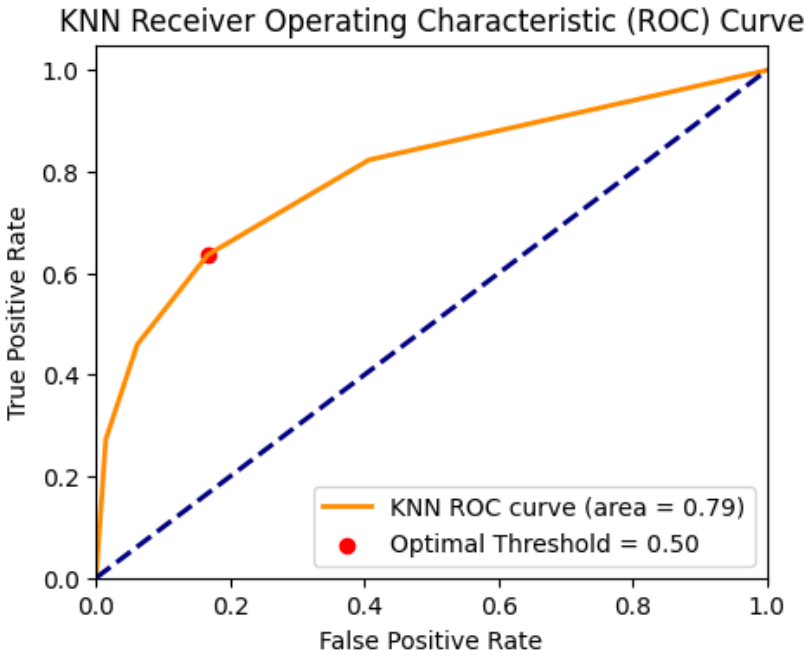
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n\_neighbors=4)
KNN.fit(X_train, y_train)
```

Sử dụng *scikit-learn* để khởi tạo mô hình KNN với

- Siêu tham số **n_neighbors**, biểu diễn số lượng neighbors của một điểm bất kỳ.
- Siêu tham số này được chọn là **4** theo một cách ngẫu nhiên.

Dự đoán xác suất để xác định threshold tốt nhất

```
KNN_probabilities = KNN.predict_proba(np.array(X_test))[:, 1]
optimal_threshold = plot_roc_curve(y_test, KNN_probabilities, model_name='KNN')
```



Hình 10: *Kết quả threshold tốt nhất cho KNN*

Đưa ra dự đoán cuối cùng

```
KNN_predictions = KNN.predict(np.array(X_test))

KNN_Accuracy_Score = accuracy_score(y_test, KNN_predictions)
KNN_JaccardIndex = jaccard_score(y_test, KNN_predictions, average='weighted')
KNN_F1_Score = f1_score(y_test, KNN_predictions, average='weighted')

print(f"KNN_Accuracy_Score: {KNN_Accuracy_Score}")
print(f"KNN_JaccardIndex: {KNN_JaccardIndex}")
print(f"KNN_F1_Score: {KNN_F1_Score}")
```

Kết quả

```
KNN_Accuracy_Score: 0.8175675675675675
KNN_JaccardIndex: 0.6913754792195159
KNN_F1_Score: 0.8028705788688126
```

Nhận xét: Mô hình KNN này thể hiện hiệu suất tốt với các chỉ số đánh giá cao, cho thấy nó có khả năng phân loại chính xác và cân bằng giữa độ chính xác và độ nhạy.

5.3.3 Decision Tree

Khởi tạo và huấn luyện

```
from sklearn.tree import DecisionTreeClassifier
```

```
Tree = DecisionTreeClassifier()
Tree.fit(X_train, y_train)
```

Đưa ra dự đoán cuối cùng

```
Tree_predictions = Tree.predict(X_test)

Tree_Accuracy_Score = accuracy_score(y_test, Tree_predictions)
Tree_JaccardIndex = jaccard_score(y_test, Tree_predictions, average='weighted')
Tree_F1_Score = f1_score(y_test, Tree_predictions, average='weighted')

print(f"Tree_Accuracy_Score: {Tree_Accuracy_Score}")
print(f"Tree_JaccardIndex: {Tree_JaccardIndex}")
print(f"Tree_F1_Score: {Tree_F1_Score}")
```

Kết quả

```
Tree_Accuracy_Score: 0.7391891891891892
Tree_JaccardIndex: 0.6107380124189712
Tree_F1_Score: 0.7479599259046547
```

Nhận xét: Mô hình cây quyết định này có hiệu suất ở mức khá, khả năng phân loại chính xác và sự cân bằng giữa độ chính xác và độ nhạy là chưa cao.

5.3.4 Random Forest

Khởi tạo và huấn luyện

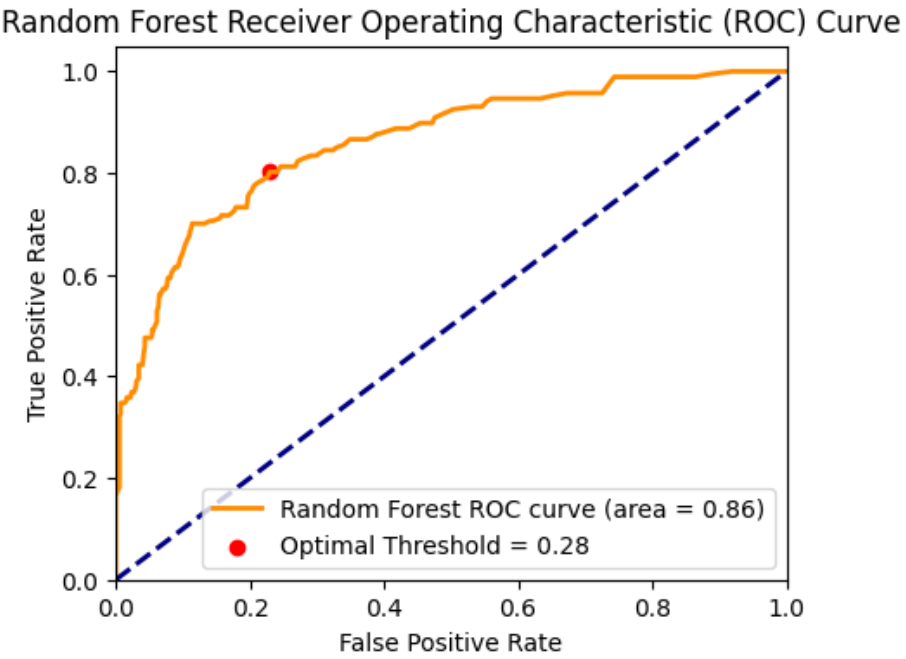
```
from sklearn.ensemble import RandomForestClassifier
Forest = RandomForestClassifier(n_estimators=200, random_state=24)
Forest.fit(X_train, y_train)
```

Sử dụng *scikit-learn* để khởi tạo mô hình RandomForestClassifier với

- Siêu tham số **n_estimators** biểu diễn số lượng cây quyết định trong ensemble. Việc tăng số lượng cây thường dẫn đến một mô hình mạnh mẽ hơn, nhưng cũng tăng độ phức tạp và thời gian huấn luyện.
- Siêu tham số **random_state** biểu diễn việc đặt seed cho việc tạo ra các số ngẫu nhiên trong quá trình xây dựng mô hình. Điều này đảm bảo rằng mỗi khi bạn chạy mã này, các kết quả được sinh ra sẽ giống nhau, giúp việc tái lập được.
- Hai siêu tham số này được chọn giá trị lần lượt là **200** và **24** một cách ngẫu nhiên.

Dự đoán xác suất để xác định threshold tốt nhất

```
Forest_probabilities = Forest.predict_proba(X_test)[:, 1]
optimal_threshold = plot_roc_curve(y_test, Forest_probabilities, model_name='Random Forest')
```



Hình 11: *Kết quả threshold tốt nhất cho Random Forest*

Đưa ra dự đoán cuối cùng

```
Forest_predictions = (Forest_probabilities > optimal_threshold).astype(int)

Forest_Accuracy_Score = accuracy_score(y_test, Forest_predictions)
Forest_JaccardIndex = jaccard_score(y_test, Forest_predictions, average='weighted')
Forest_F1_Score = f1_score(y_test, Forest_predictions, average='weighted')

print(f"Forest_Accuracy_Score: {Forest_Accuracy_Score}")
print(f"Forest_JaccardIndex: {Forest_JaccardIndex}")
print(f"Forest_F1_Score: {Forest_F1_Score}")
```

Kết quả

```
Forest_Accuracy_Score: 0.7783783783783784
Forest_JaccardIndex: 0.6601475435934895
Forest_F1_Score: 0.7897523098546116
```

Nhận xét: Mô hình Random Forest này thể hiện hiệu suất tốt với các chỉ số đánh giá cao, cho thấy nó có khả năng phân loại chính xác và cân bằng giữa độ chính xác và độ nhạy.

5.3.5 Logistic Regression

Khởi tạo và huấn luyện

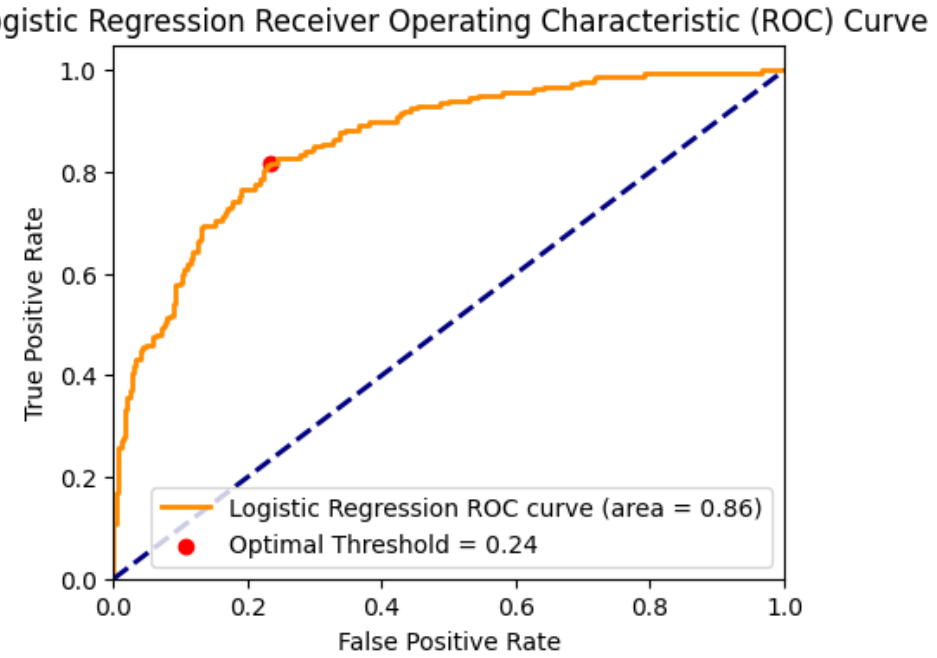
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
```

Sử dụng *scikit-learn* để khởi tạo mô hình Logistic Regression với

- Siêu tham số **solver**, biểu diễn phương pháp để giải bài toán tối ưu đối với cross entropy. Đối với dữ liệu kích thước nhỏ thì liblinear sử dụng sẽ phù hợp hơn; các lựa chọn như sag, saga có tốc độ huấn luyện nhanh hơn cho dữ liệu lớn.
- Siêu tham số được chọn là **liblinear**.

Dự đoán xác suất để xác định threshold tốt nhất

```
LR_probabilities = LR.predict_proba(X_test)[:,1]
optimal_threshold = plot_roc_curve(y_test, LR_probabilities, model_name='Logistic Regression')
```



Hình 12: *Kết quả threshold tốt nhất cho Logistic Regression*

Đưa ra dự đoán cuối cùng

```
LR_predictions = (LR_probabilities > optimal_threshold).astype(int)

LR_Accuracy_Score = accuracy_score(y_test, LR_predictions)
LR_JaccardIndex = jaccard_score(y_test, LR_predictions, average='weighted')
LR_F1_Score = f1_score(y_test, LR_predictions, average='weighted')

print(f"LR_Accuracy_Score: {LR_Accuracy_Score}")
print(f"LR_JaccardIndex: {LR_JaccardIndex}")
print(f"LR_F1_Score: {LR_F1_Score}")
```

Kết quả

```
LR_Accuracy_Score: 0.7783783783783784
LR_JaccardIndex: 0.6604206376358276
LR_F1_Score: 0.790342546864286
```

Nhận xét: Mô hình Logistic Regression này thể hiện hiệu suất tốt với các chỉ số đánh giá cao, cho thấy nó có khả năng phân loại chính xác và cân bằng giữa độ chính xác và độ nhạy.

5.3.6 SVM

Khởi tạo và huấn luyện

```
from sklearn import svm
SVM = svm.SVC(kernel='rbf', C=10)
SVM.fit(X_train, y_train)
```

Sử dụng *scikit-learn* để khởi tạo mô hình SVM với

- Siêu tham số **kernel**, biểu diễn hàm kernel được sử dụng. Ở đây dùng radial basis function (RBF), một loại kernel thường được sử dụng để ánh xạ dữ liệu vào một không gian chiều cao hơn để giải quyết các vấn đề phi tuyến tính.
- Siêu tham số **C**, là tham số điều chuẩn (regularization parameter) ở mức độ độ cứng của biên. Giá trị C càng cao, mô hình sẽ cố gắng giảm thiểu sai số trên tập huấn luyện, nhưng có nguy cơ bị overfitting.
- Siêu tham số **kernel** được chọn là **rbf** và **C** được chọn là **10** theo một cách ngẫu nhiên.

Đưa ra dự đoán cuối cùng

```
SVM_predictions = SVM.predict(X_test)

SVM_Accuracy_Score = accuracy_score(y_test, SVM_predictions)
SVM_JaccardIndex = jaccard_score(y_test, SVM_predictions, average='weighted')
SVM_F1_Score = f1_score(y_test, SVM_predictions, average='weighted')

print(f"SVM_Accuracy_Score: {SVM_Accuracy_Score}")
print(f"SVM_JaccardIndex: {SVM_JaccardIndex}")
print(f"SVM_F1_Score: {SVM_F1_Score}")
```

Kết quả

```
SVM_Accuracy_Score: 0.8216216216216217
SVM_JaccardIndex: 0.6805067170175457
SVM_F1_Score: 0.7863752142386289
```

Nhận xét: Mô hình SVM này thể hiện hiệu suất tốt với các chỉ số đánh giá cao, cho thấy nó có khả năng phân loại chính xác và cân bằng giữa độ chính xác và độ nhạy.

5.3.7 Gradient Boosting

Khởi tạo và huấn luyện

```
from sklearn.ensemble import GradientBoostingClassifier
params = {
    "n_estimators": 500,
    "max_depth": 4,
    "min_samples_split": 5,
```

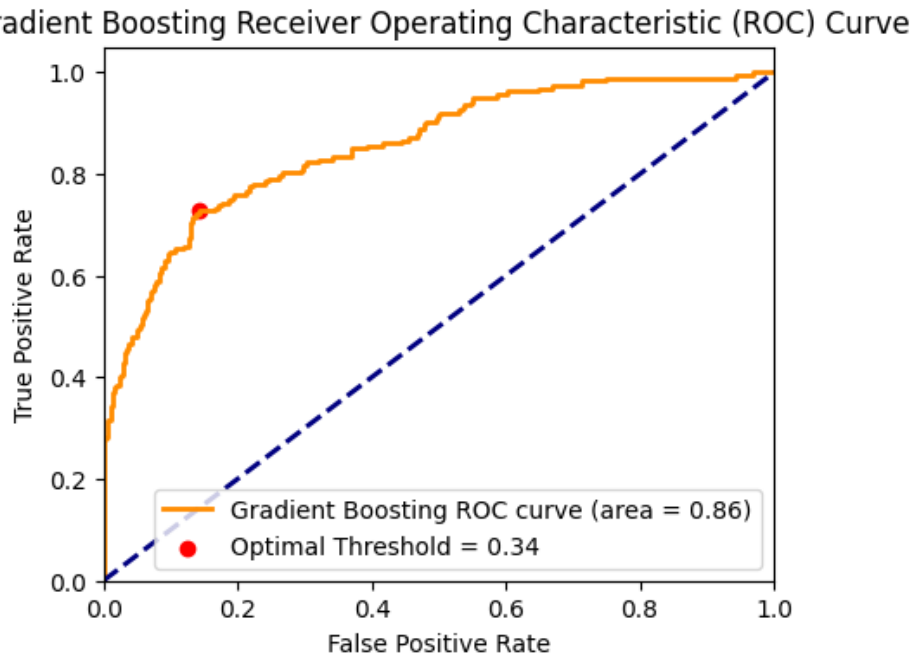
```
    "learning_rate": 0.01,
    "loss": "log_loss",
}
GBC = GradientBoostingClassifier(**params)
GBC.fit(X_train, y_train)
```

Sử dụng *scikit-learn* để khởi tạo mô hình Gradient Boosting với

- Siêu tham số **n_estimators**, biểu diễn số lượng giai đoạn tăng cường (cũng được gọi là số lượng cây quyết định) mà thuật toán sẽ hoàn thành. Số lượng lớn sẽ giúp mô hình mạnh mẽ hơn nhưng cũng tăng độ phức tạp và thời gian.
- Siêu tham số **max_depth**, biểu diễn độ sâu tối đa của cây quyết định được sinh ra trong mỗi giai đoạn.
- Siêu tham số **min_samples_split**, biểu diễn số lượng mẫu tối thiểu cần thiết để chia một nút nội bộ.
- Siêu tham số **learning_rate**, biểu diễn tốc độ học. Điều này kiểm soát mức độ mà mỗi cây quyết định góp phần vào dự đoán cuối cùng. Một tốc độ học nhỏ hơn có thể yêu cầu nhiều giai đoạn tăng cường hơn nhưng có thể tạo ra mô hình chính xác hơn.
- Siêu tham số **loss**, biểu diễn cho hàm mất mát được dùng trong tối ưu hóa.
- Các siêu tham số trên được chọn một cách ngẫu nhiên.

Dự đoán xác suất để xác định threshold tốt nhất

```
GBC_probabilities = GBC.predict_proba(X_test)[:,1]
optimal_threshold = plot_roc_curve(y_test, GBC_probabilities, model_name='Gradient Boosting')
```



Hình 13: *Kết quả threshold tốt nhất cho Gradient Boosting*

Đưa ra dự đoán cuối cùng

```
GBC_predictions = (GBC_probabilities > optimal_threshold).astype(int)

GBC_Accuracy_Score = accuracy_score(y_test, GBC_predictions)
GBC_JaccardIndex = jaccard_score(y_test, GBC_predictions, average='weighted')
GBC_F1_Score = f1_score(y_test, GBC_predictions, average='weighted')

print(f"GBC_Accuracy_Score: {GBC_Accuracy_Score}")
print(f"GBC_JaccardIndex: {GBC_JaccardIndex}")
print(f"GBC_F1_Score: {GBC_F1_Score}")
```

Kết quả

```
GBC_Accuracy_Score: 0.822972972972973
GBC_JaccardIndex: 0.7137371880673314
GBC_F1_Score: 0.8267178470675752
```

Nhận xét: Mô hình Gradient Boosting này thể hiện hiệu suất tốt với các chỉ số đánh giá rất cao, cho thấy nó có khả năng phân loại rất chính xác và cân bằng giữa độ chính xác và độ nhạy.

6 Đánh giá các mô hình

Sau khi đã có những đánh giá sơ bộ về từng thuật toán được áp dụng, ta sẽ thực hiện việc đánh giá và so sánh những thuật toán với nhau.

6.1 Đánh giá bằng chỉ số đánh giá (Evaluation Metrics)

Ta đánh giá các mô hình theo các chỉ số sau (đã được giải thích rõ hơn ở phần **Các phương pháp đánh giá**)

- Accuracy
- Jaccard Index
- F1 Score

Tạo một DataFrame để thuận tiện hơn trong việc theo dõi

```
Report = pd.DataFrame({
    'Model': ['KNN', 'Decision Tree', 'Random Forest', 'Logistic Regression', 'SVM', 'Gradient
        Boosting'],
    'Accuracy': [KNN_Accuracy_Score, Tree_Accuracy_Score, Forest_Accuracy_Score,
        LR_Accuracy_Score, SVM_Accuracy_Score, GBC_Accuracy_Score],
    'Jaccard Index': [KNN_JaccardIndex, Tree_JaccardIndex, Forest_JaccardIndex,
        LR_JaccardIndex, SVM_JaccardIndex, GBC_JaccardIndex],
    'F1 Score': [KNN_F1_Score, Tree_F1_Score, Forest_F1_Score, LR_F1_Score, SVM_F1_Score,
        GBC_F1_Score]
})

Report
```

Kết quả:

	Model	Accuracy	Jaccard Index	F1 Score
0	KNN	0.817568	0.691375	0.802871
1	Decision Tree	0.739189	0.610738	0.747960
2	Random Forest	0.778378	0.660148	0.789752
3	Logistic Regression	0.778378	0.660421	0.790343
4	SVM	0.821622	0.680507	0.786375
5	Gradient Boosting	0.822973	0.713737	0.826718

Hình 14: *Metrics*

Nhận xét:

- Accuracy:

  Độ chính xác cao nhất nằm ở Gradient Boosting.

  Độ chính xác thấp nhất là Decision Tree.

Trong giai đoạn xử lý dữ liệu, ta thay các dữ liệu khuyết bởi trung bình các giá trị trong cùng tháng, có thể điều đó ảnh hưởng tốt đến KNN khi thuật toán này cũng có độ chính xác cao.

- Jaccard Index:

  Chỉ số cao nhất nằm ở Gradient Boosting.

  Chỉ số thấp nhất là Decision Tree.

- F1 Score:

  Chỉ số cao nhất nằm ở Gradient Boosting.

  Chỉ số thấp nhất là Decision Tree.

- Gradient Boosting cho thấy hiệu suất tốt hơn rõ rệt so với tất cả các thuật toán còn lại. Điều này có thể được giải thích do Gradient Boosting là một phương pháp học ensemble, tận dụng sức mạnh của nhiều mô hình yếu kết hợp lại để cải thiện hiệu suất tổng thể.

  Decision Tree cho thấy hiệu suất kém hơn rõ rệt so với tất cả các thuật toán còn lại. Điều này có thể do xu hướng quá khớp hoặc không đủ phức tạp để phân biệt các mẫu dữ liệu một cách hiệu quả. Decision Tree thường gặp khó khăn với các tập dữ liệu phức tạp hoặc có nhiều nhiễu.

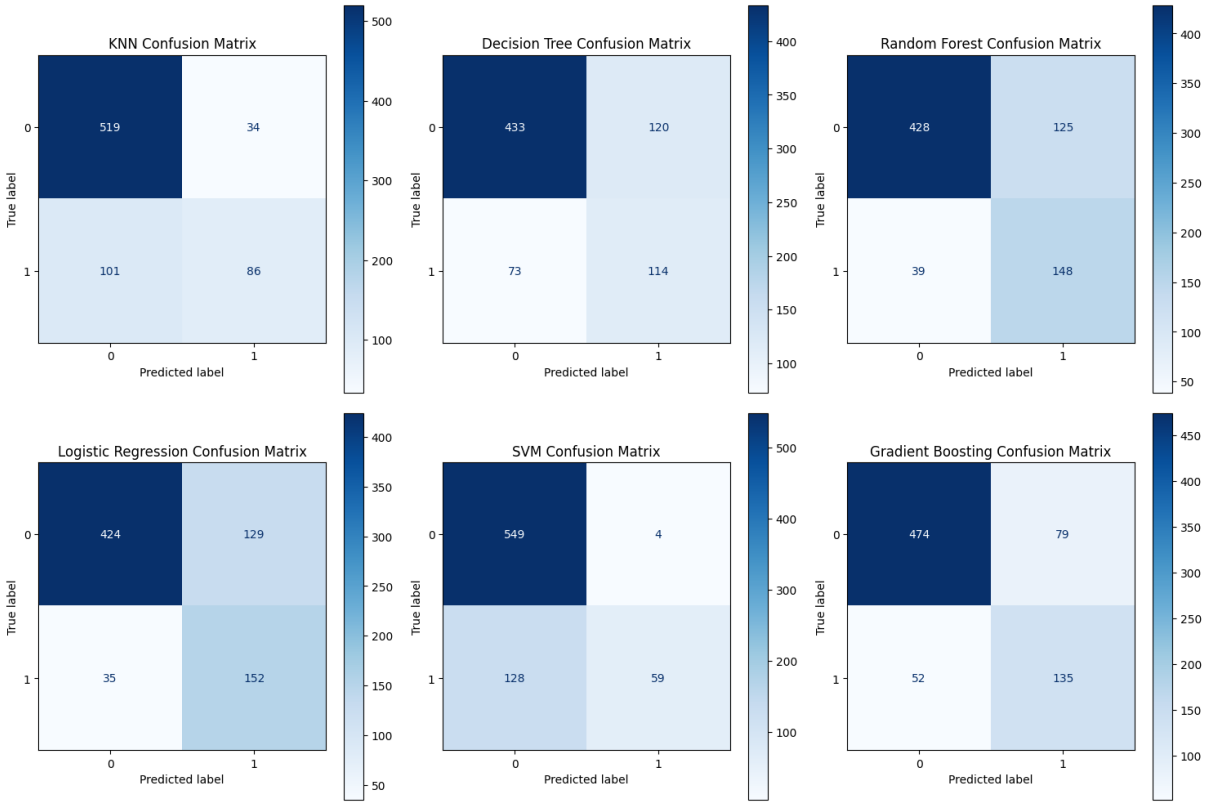
- Decision Tree & Random Forest: Random Forest sử dụng nhiều cây quyết định để giảm thiểu vấn đề quá khớp, nên việc các Decision Tree thành phần không tốt cũng dẫn đến Random Forest không thật sự ấn tượng, mặc dù đã có cải thiện so với Decision Tree.

- Random Forest & Logistic Regression: các thông số của hai thuật toán này gần giống nhau. Có thể đây chỉ là một sự trùng hợp, do Logistic Regression thường hoạt động tốt với dữ liệu có quan hệ tuyến tính giữa các đặc trưng và kết quả, trong khi Random Forest hoạt động tốt hơn với các dữ liệu phi tuyến tính.

6.2 Đánh giá bằng Confusion matrix

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score

classifiers = {
    "KNN": KNN_predictions,
    "Decision Tree": Tree_predictions,
    "Random Forest": Forest_predictions,
    "Logistic Regression": LR_predictions,
    "SVM": SVM_predictions,
    "Gradient Boosting": GBC_predictions
}
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axes = axes.flatten()

for ax, (name, predictions) in zip(axes, classifiers.items()):
    cm = confusion_matrix(y_test, predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
    disp.plot(ax=ax, cmap=plt.cm.Blues)
    ax.set_title(f"{name} Confusion Matrix")

for ax in axes[len(classifiers):]:
    fig.delaxes(ax)

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



Hình 15: *Các confusion matrix*

Với cách vẽ confusion matrix như trên thì True/False Positive/Negative tương ứng là

True Negative	False Positive
False Negative	True Positive

Nhận xét:

- Với True Negative (TN): tất cả thuật toán đều đưa ra TN là lớn nhất so với bộ dữ liệu.

  SVM đưa ra nhiều TN nhất.

  Logistic đưa ra ít TN nhất.

- Với False Negative (FN):

  Logistic đưa ra ít FN nhất.

  SVM đưa ra nhiều FN nhất.

- Với Positive (False Positive và True Positive):

  Logistic đưa ra nhiều Positive nhất (129 + 152 = 281).

  SVM đưa ra ít Positive nhất (4 + 59 = 63).

- Logistic Regression cho thấy hiệu suất tốt hơn rõ rệt so với tất cả các thuật toán còn lại. Mô hình này có độ nhạy cao, khả năng nhận diện đúng các mẫu dương tính tốt, và mặc dù có số lượng dự đoán sai (false positive) cao, nó vẫn duy trì được hiệu suất tổng thể tốt.

  SVM cho thấy hiệu suất kém hơn rõ rệt so với tất cả các thuật toán còn lại. Mặc dù SVM có khả năng phân loại tốt các mẫu âm tính, nhưng gặp khó khăn lớn trong việc nhận diện các mẫu dương tính, dẫn đến số lượng lớn các dự đoán sai (false negative).

- Random Forest & Logistic Regression: Thông số của hai thuật toán này xấp xỉ nhau.

6.3 Đánh giá tổng quan

Sau khi nhận xét theo chỉ số đánh giá và theo Confusion matrix, ta có nhận xét tổng quan như sau:

- **KNN:** Mô hình được hưởng lợi khi ta thay các dữ liệu khuyết bởi trung bình các giá trị trong cùng tháng, kết quả là thuật toán này có độ chính xác cao.
- **Decision Tree:** Mô hình cho hiệu suất kém nhất nếu đánh giá bằng các chỉ số, cho hiệu suất ổn nếu đánh giá bằng confusion matrix.
- **Random Forest:** Với cả hai phương pháp đánh giá, mô hình đều chứng tỏ được sự cải thiện hơn Decision Tree khi Random Forest sử dụng nhiều cây quyết định để đưa ra dự đoán.
- **Logistic Regression:** Mô hình cho hiệu suất tốt nhất khi đánh giá bằng confusion matrix, cho hiệu suất ổn nếu đánh giá bằng chỉ số.
- **SVM:** Mô hình cho hiệu suất kém nhất khi đánh giá bằng confusion matrix, cho hiệu suất ổn nếu đánh giá bằng chỉ số.
- **Gradient Boosting:** Mô hình cho thấy hiệu suất tốt nhất nếu đánh giá bằng chỉ số, hiệu suất tốt nếu đánh giá bằng confusion matrix. Nó chứng tỏ được sự hiệu quả của phương pháp học ensemble, tận dụng sức mạnh của nhiều mô hình yếu kết hợp lại để cải thiện hiệu suất tổng thể.

7 Kết luận

Trong nghiên cứu này, ta đã tập trung vào bài toán dự đoán trời mưa, một thách thức quan trọng trong dự báo thời tiết và cải thiện đời sống người dân. Ta đã áp dụng và so sánh các mô hình supervised learning, bao gồm KNN, Decision Tree, Random Forest, Logistic Regression, Support Vector Machine (SVM), và Gradient Boosting (GB), để xem xét khả năng của chúng trong việc dự đoán trời mưa.

Để đánh giá hiệu suất của mô hình học máy phân loại, ta đã áp dụng hai phương pháp đánh giá khác nhau. Với phương pháp đánh giá bằng chỉ số đánh giá (Evaluation Metrics), các chỉ số thường dùng là độ chính xác (Accuracy), Jaccard Index, F1 Score. Với phương pháp đánh giá bằng confusion matrix, ta sử dụng một ma trận là một bảng thể hiện các kết quả dự đoán của mô hình (True/False Positive/Negative), cung cấp cái nhìn chi tiết hơn về cách mô hình hoạt động đối với từng loại dự đoán (dương tính và âm tính) và giúp xác định rõ ràng hơn các sai sót mà mô hình đang gặp phải.

Phương pháp đánh giá bằng các chỉ số đánh giá chứng tỏ rằng mô hình Gradient Boosting đã mang lại hiệu suất tốt nhất so với các mô hình khác, với độ chính xác và khả năng dự đoán cao. Sự mạnh mẽ của Gradient Boosting trong việc xử lý các mối quan hệ phi tuyến tính và khả năng tự điều chỉnh tốt đã giúp nó vượt trội trong bài toán dự đoán trời mưa. Trong khi đó mô hình Decision Tree mang lại hiệu suất kém nhất, với độ chính xác không cao.

Phương pháp đánh giá bằng confusion matrix chứng tỏ rằng mô hình Logistic Regression có hiệu suất tốt nhất so với các mô hình khác, với lượng dự đoán dương tính là nhiều nhất. Random Forest cũng rất tốt, với các thông số xấp xỉ Logistic Regression. Trong khi đó, SVM mang lại hiệu suất kém nhất, với lượng dự đoán âm tính là ít nhất.

Những kết quả này không chỉ có ý nghĩa trong việc cải thiện đời sống người dân mà còn đề xuất những hướng phát triển mới trong nghiên cứu và ứng dụng trong lĩnh vực này. Các mô hình supervised learning có thể được kết hợp để tạo ra một hệ thống dự đoán thời tiết linh hoạt, đáp ứng hiệu quả cho sự biến động của môi trường.

References

[1] Australian Government's Bureau of Meteorology. *Notes to accompany Daily Weather Observations*. http://www.bom.gov.au/climate/dwo/IDCJDW0000.shtml.

[2] Tuấn Nguyễn. *Machine Learning cho dữ liệu dạng bảng - Random Forest algorithm*. https://machinelearningcoban.com/tabml_book/ch_model/random_forest.html.

[3] National Weather Service. *Glossary*. https://forecast.weather.gov/glossary.php?word=calm.

[4] Vũ Hữu Tiệp. *Machine Learning cơ bản - Bài 10: Logistic Regression*. https://machinelearningcoban.com/2017/01/27/logisticregression/.

[5] Vũ Hữu Tiệp. *Machine Learning cơ bản - Bài 19: Support Vector Machine*. https://machinelearningcoban.com/2017/04/09/smv/.

[6] Vũ Hữu Tiệp. *Machine Learning cơ bản - Bài 33: Các phương pháp đánh giá một hệ thống phân lớp*. https://machinelearningcoban.com/2017/08/31/evaluation/.

[7] Vũ Hữu Tiệp. *Machine Learning cơ bản - Bài 34: Decision Trees (1): Iterative Dichotomiser 3*. https://machinelearningcoban.com/2018/01/14/id3/.

[8] Vũ Hữu Tiệp. *Machine Learning cơ bản - Bài 6: K-nearest neighbors*. https://machinelearningcoban.com/2017/01/08/knn/.