**CÁC THUẬT TOÁN ÁP DỤNG ĐỂ DỰ ĐOÁN VÀ PHÂN TÍCH**

**Máy vector hỗ trợ (SVM)**

Máy vector hỗ trợ (Support Vector Machine - SVM) là một thuật toán học máy được sử dụng cho các bài toán phân loại và hồi quy. SVM tìm ra một ranh giới phân chia giữa các lớp sao cho khoảng cách giữa các điểm dữ liệu và ranh giới phân chia là lớn nhất. SVM được áp dụng trong nhiều lĩnh vực, chẳng hạn như nhận dạng ký tự, dự đoán chuỗi gen, phân loại ảnh, v.v.

Các điểm dữ liệu được biểu diễn trong không gian nhiều chiều, mỗi chiều là một đặc trưng (feature). Với bài toán phân loại, SVM tìm ra một siêu phẳng (hyperplane) để phân tách các điểm thuộc lớp khác nhau. Nếu không thể tìm được siêu phẳng phân chia tuyến tính thì SVM sử dụng kernel trick để chuyển đổi không gian ban đầu sang một không gian khác có thể phân chia tuyến tính.

Sau khi tìm được siêu phẳng phân chia, SVM tìm ra các vector hỗ trợ (support vectors) là các điểm nằm gần siêu phẳng nhất. Điểm mạnh của SVM là nó chỉ cần lưu trữ các vector hỗ trợ này để đưa ra dự đoán trên dữ liệu mới.



Hàm SVM trong đoạn mã trên được sử dụng để huấn luyện mô hình máy học SVM (Support Vector Machine) trên dữ liệu trong file XSMB.csv, và tính toán độ chính xác trên tập kiểm tra.

Đầu tiên, đoạn mã sử dụng thư viện pandas để đọc dữ liệu từ file CSV vào trong một DataFrame. Tham số parse\_dates được sử dụng để chuyển đổi cột "Ngày" sang định dạng ngày tháng.

Tiếp theo, đoạn mã tách dữ liệu thành hai phần: một phần là các cột "Đặc biệt" và "Giải nhất", được lưu trong biến X, và phần còn lại là cột "Ngày", được chuyển đổi sang dạng số nguyên và lưu trong biến y.

Sau đó, đoạn mã sử dụng hàm train\_test\_split trong thư viện sklearn để chia dữ liệu thành hai tập huấn luyện và kiểm tra. Tham số test\_size xác định tỉ lệ dữ liệu được chia vào tập kiểm tra.

Tiếp theo, đoạn mã khởi tạo một mô hình SVM với kernel tuyến tính và tham số C=1. Tham số C được sử dụng để điều chỉnh mức độ lỏng lẻo của ranh giới phân loại, giá trị càng cao thì mô hình càng khắt khe.

Sau đó, đoạn mã huấn luyện mô hình SVM trên tập huấn luyện bằng cách gọi phương thức fit trên mô hình SVM với các tham số X\_train và y\_train.

Tiếp theo, đoạn mã dự đoán giá trị trên tập kiểm tra bằng cách gọi phương thức predict trên mô hình SVM với tham số X\_test.

Cuối cùng, đoạn mã tính toán độ chính xác trên tập kiểm tra bằng cách sử dụng hàm accuracy\_score trong thư viện sklearn. Kết quả được in ra màn hình bằng lệnh print.

Tóm lại, hàm SVM trong đoạn mã trên là một ví dụ về việc sử dụng thuật toán SVM để huấn luyện mô hình máy học trên dữ liệu có cấu trúc. Quá trình huấn luyện bao gồm các bước như chia dữ liệu thành tập huấn luyện và kiểm tra, khởi tạo mô hình SVM, huấn luyện mô hình trên tập huấn luyện và dự đoán giá

*Chu Kỳ*

Chu kỳ của thuật toán SVM (Support Vector Machine) trong trường hợp này là:

Đọc dữ liệu từ file CSV và chuyển đổi các giá trị "Ngày" thành định dạng số nguyên.

Tách dữ liệu thành tập huấn luyện và tập kiểm tra sử dụng hàm train\_test\_split().

Khởi tạo mô hình SVM với các tham số như kernel và C.

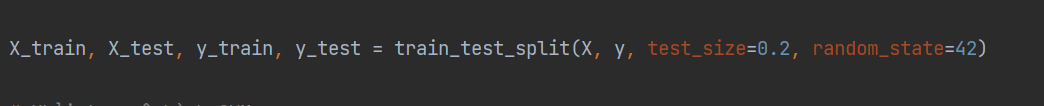
Huấn luyện mô hình trên tập huấn luyện sử dụng phương thức fit().

Sử dụng mô hình đã huấn luyện để dự đoán giá trị trên tập kiểm tra sử dụng phương thức predict().

Tính toán độ chính xác trên tập kiểm tra sử dụng hàm accuracy\_score().

Trả về kết quả độ chính xác trên tập kiểm tra.

Sau khi kết thúc chu kỳ, ta có kết quả độ chính xác của mô hình SVM trên tập kiểm tra.

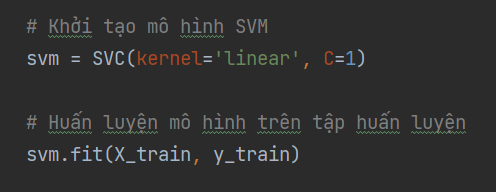


Hàm train\_test\_split trong thư viện sklearn được sử dụng để chia dữ liệu thành hai tập huấn luyện và kiểm tra, thông thường để đánh giá mô hình máy học.

Cụ thể, với tham số test\_size=0.2, hàm này sẽ chia dữ liệu ngẫu nhiên thành hai tập con là X\_train và X\_test, với tỉ lệ tương ứng là 80% cho tập huấn luyện và 20% cho tập kiểm tra. Đồng thời, nó sẽ chia cả nhãn y (trong trường hợp này là ngày) tương ứng cho từng tập con.

Tham số random\_state được sử dụng để thiết lập một số nguyên ngẫu nhiên để việc chia dữ liệu trở nên xác định. Nếu không thiết lập tham số này, kết quả chia dữ liệu sẽ khác nhau mỗi lần chạy.

Sau khi thực hiện lệnh trên, X\_train và y\_train sẽ chứa 80% dữ liệu ban đầu được sử dụng để huấn luyện mô hình, trong khi X\_test và y\_test sẽ chứa 20% dữ liệu còn lại được sử dụng để kiểm tra độ chính xác của mô hình.



Lệnh này sẽ khởi tạo một đối tượng mô hình SVM với các tham số được thiết lập. Trong đó:

kernel='linear' thiết lập kernel function cho mô hình SVM là linear (tức là hàm tuyến tính), giúp giải quyết bài toán phân loại tuyến tính.

C=1 thiết lập hằng số regularization trong mô hình SVM, giúp kiểm soát độ phức tạp của mô hình. Tham số này càng lớn, mô hình càng phức tạp và có khả năng bị overfitting.

Sau đó, mô hình SVM sẽ được huấn luyện trên tập huấn luyện X\_train và y\_train bằng cách gọi phương thức fit().

Kết quả sau khi huấn luyện mô hình là một mô hình SVM được đào tạo, đã học được các hệ số và các siêu mặt phân cách tương ứng với các giá trị của dữ liệu huấn luyện. Mô hình SVM này sẽ được sử dụng để dự đoán các giá trị của tập kiểm tra X\_test.

Hàm fit() trong mô hình SVM sử dụng phương pháp tối ưu hóa Quadratic Programming (QP) để tìm giá trị của các hệ số và các siêu mặt phân cách tối ưu, từ đó xây dựng mô hình SVM. Cụ thể, công thức tối ưu hóa của mô hình SVM được biểu diễn như sau:

minimize:

$\frac{1}{2} ||\mathbf{w}||^2 + C \sum\_{i=1}^n \xi\_i$

subject to:

$y\_i(\mathbf{w}^T \mathbf{x}\_i + b) \geq 1 - \xi\_i$

$\xi\_i \geq 0, i = 1, ..., n$

Trong đó:

$\mathbf{x}\_i$ là một vectơ đặc trưng (feature vector) của một điểm dữ liệu, có chiều dài bằng số lượng các đặc trưng của dữ liệu.

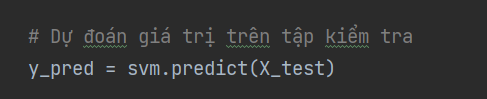
$y\_i$ là nhãn của điểm dữ liệu thứ $i$, có giá trị bằng 1 hoặc -1, tương ứng với hai lớp dữ liệu khác nhau trong bài toán phân loại.

$\mathbf{w}$ là một vectơ trọng số (weight vector) có chiều dài bằng số lượng đặc trưng của dữ liệu, được tối ưu để tạo ra siêu mặt phân cách tốt nhất giữa hai lớp dữ liệu.

$b$ là một hằng số đệm (bias), được tối ưu để điều chỉnh vị trí của siêu mặt phân cách.

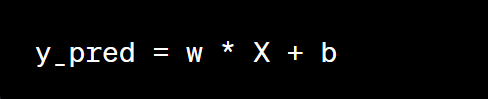
$\xi\_i$ là biến lỏng (slack variable) cho phép mô hình SVM mềm, có giá trị lớn hơn 0 nếu điểm dữ liệu thứ $i$ bị phân loại sai, giúp cho mô hình SVM có thể tìm được một siêu mặt phân cách tốt hơn trong trường hợp các điểm dữ liệu không phân biệt được hoàn toàn bằng siêu mặt phân cách tuyến tính.

Mục tiêu của hàm tối ưu trên là tìm giá trị của vectơ trọng số $\mathbf{w}$, hằng số đệm $b$ và biến lỏng $\xi\_i$ sao cho mô hình SVM đáp ứng các ràng buộc điều kiện phân loại đúng toàn bộ các điểm dữ liệu huấn luyện, đồng thời đạt được độ tối ưu hóa cao nhất của hàm mất mát (loss function) được xác định bởi công thức trên.



Hàm predict() trong mô hình SVM được sử dụng để dự đoán nhãn của các mẫu trong tập dữ liệu kiểm tra sau khi mô hình đã được huấn luyện trên tập dữ liệu huấn luyện. Cụ thể, hàm predict() nhận đầu vào là tập dữ liệu kiểm tra và trả về một mảng các dự đoán nhãn tương ứng với mỗi mẫu trong tập kiểm tra.

Trong trường hợp của thuật toán SVM với kernel tuyến tính, hàm predict() sử dụng công thức sau để tính toán giá trị dự đoán của một mẫu:



Trong đó:

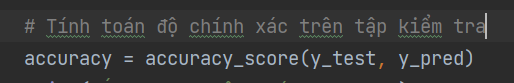
w là vector trọng số của mô hình đã được học trong quá trình huấn luyện

b là hệ số điều chỉnh đường biên của mô hình đã được học trong quá trình huấn luyện

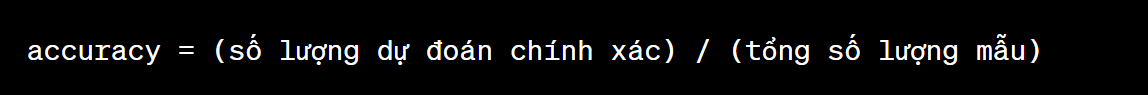
X là vector đặc trưng của mẫu cần được dự đoán

y\_pred là giá trị dự đoán của mẫu.

Các giá trị này được tính toán bằng cách sử dụng thông tin đã học được từ quá trình huấn luyện để xác định vị trí của mẫu đó trên đường biên phân chia các lớp của bài toán phân loại. Nếu giá trị dự đoán lớn hơn hoặc bằng một ngưỡng xác định trước, mẫu sẽ được dự đoán thuộc lớp 1, ngược lại nếu giá trị dự đoán nhỏ hơn ngưỡng thì mẫu sẽ được dự đoán thuộc lớp 0.



Hàm accuracy\_score được sử dụng để tính toán độ chính xác của mô hình dựa trên dự đoán (y\_pred) và nhãn thực tế (y\_test) trên tập kiểm tra. Công thức tính toán độ chính xác là:



Trong đó, số lượng dự đoán chính xác là số lượng mẫu mà mô hình đã dự đoán đúng trên tập kiểm tra, và tổng số lượng mẫu là tổng số lượng mẫu trên tập kiểm tra.

Hàm accuracy\_score trong thư viện sklearn.metrics sẽ tính toán số lượng dự đoán chính xác và tổng số lượng mẫu dựa trên các giá trị truyền vào và trả về giá trị độ chính xác.

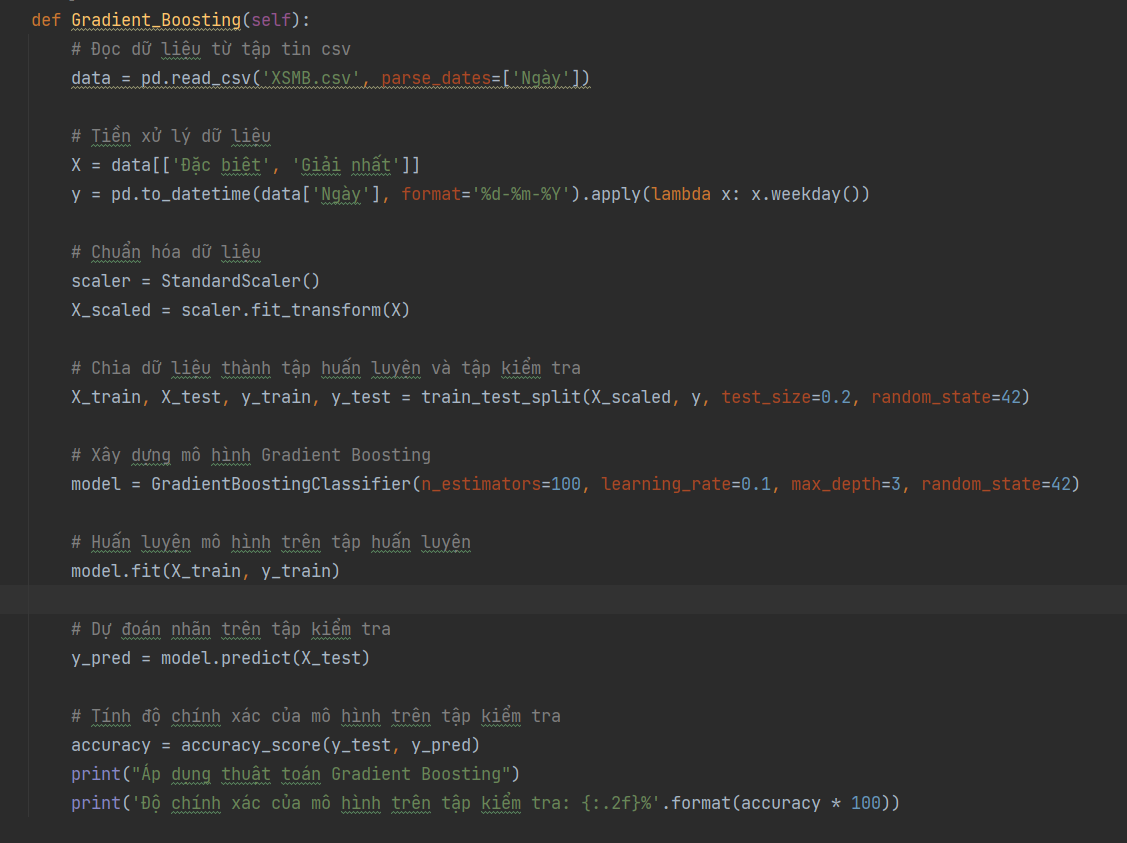
Ví dụ, nếu có 100 mẫu trên tập kiểm tra, và mô hình đã dự đoán chính xác cho 80 mẫu, thì độ chính xác sẽ là 80/100 = 0.8.

**Gradient Boosting**

Gradient Boosting là một thuật toán học máy trong đó tập hợp các mô hình dự đoán yếu được kết hợp để tạo ra một mô hình dự đoán mạnh hơn. Thuật toán Gradient Boosting đặt ra một hàm mất mát (loss function) để đo lường sự khác biệt giữa giá trị dự đoán và giá trị thực tế và cố gắng tìm cách giảm thiểu hàm mất mát này bằng cách tối ưu hóa các tham số của mô hình.

Thuật toán Gradient Boosting hoạt động bằng cách tạo ra một chuỗi các mô hình dự đoán (ví dụ: cây quyết định) liên tiếp, mỗi mô hình sẽ tập trung vào các dữ liệu bị sai lệch (residuals) của mô hình trước đó. Sau khi tạo ra một mô hình dự đoán, các trọng số được tính toán lại để tối thiểu hoá hàm mất mát của mô hình mới được tạo ra. Việc lặp lại quá trình này cho đến khi hàm mất mát đạt được giá trị tối thiểu hoặc đạt đến số lượng mô hình cần thiết.

Thuật toán Gradient Boosting là một trong những phương pháp học máy hiệu quả nhất trong các bài toán dự đoán và phân loại, đặc biệt là trong các trường hợp dữ liệu lớn và phức tạp. Tuy nhiên, việc xây dựng một mô hình Gradient Boosting thường yêu cầu thời gian và công sức đầu tư hơn so với các thuật toán khác.



Đoạn code trên sử dụng thuật toán Gradient Boosting để xử lý dữ liệu từ file XSMB.csv. Cụ thể, quá trình thực thi của đoạn code được thực hiện như sau:

Đọc dữ liệu từ tập tin XSMB.csv bằng hàm pd.read\_csv('XSMB.csv', parse\_dates=['Ngày']), dữ liệu được lưu vào biến data.

Tiền xử lý dữ liệu bằng cách chọn các cột "Đặc biệt" và "Giải nhất" làm đặc trưng X, cột "Ngày" làm nhãn y, và chuẩn hóa dữ liệu bằng StandardScaler. Kết quả được lưu vào biến X\_scaled.

Chia dữ liệu thành 2 tập là tập huấn luyện và tập kiểm tra với tỉ lệ 80:20 bằng hàm train\_test\_split.

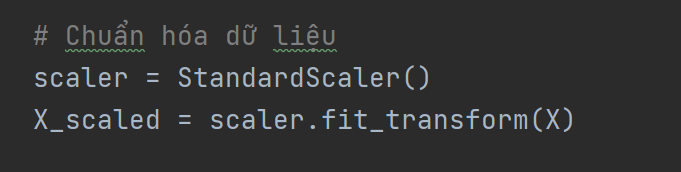
Xây dựng mô hình Gradient Boosting với 100 cây, learning rate là 0.1 và độ sâu tối đa của các cây là 3 bằng hàm GradientBoostingClassifier.

Huấn luyện mô hình trên tập huấn luyện với hàm fit.

Dự đoán nhãn trên tập kiểm tra bằng hàm predict.

Tính độ chính xác của mô hình trên tập kiểm tra bằng hàm accuracy\_score.

In ra kết quả độ chính xác của mô hình trên tập kiểm tra.



Dòng lệnh này sử dụng phương pháp chuẩn hóa StandardScaler để chuyển đổi dữ liệu trong X về dạng chuẩn hóa. Việc chuẩn hóa sẽ giúp cho các đặc trưng trong X có cùng độ lớn và cùng phân bố đồng đều giúp cho việc huấn luyện mô hình đạt hiệu quả cao hơn.

Kết quả trả về là một mảng numpy X\_scaled có cùng kích thước với X, nhưng dữ liệu trong đó đã được chuẩn hóa theo phương pháp StandardScaler.

StandardScaler là một kỹ thuật chuẩn hóa dữ liệu phổ biến trong Machine Learning. Nó thực hiện việc chuẩn hóa dữ liệu bằng cách biến đổi dữ liệu sao cho các giá trị trong dữ liệu có giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1. Cụ thể, phương pháp chuẩn hóa theo StandardScaler sử dụng công thức sau:

z = (x - u) / s

Trong đó:

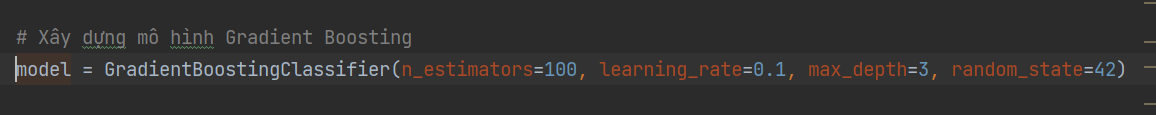
x: giá trị của một điểm dữ liệu trong tập dữ liệu ban đầu.

u: giá trị trung bình của tất cả các điểm dữ liệu trong tập dữ liệu ban đầu.

s: độ lệch chuẩn của tất cả các điểm dữ liệu trong tập dữ liệu ban đầu.

z: giá trị của một điểm dữ liệu đã được chuẩn hóa.

Việc chuẩn hóa dữ liệu giúp tăng tính ổn định và hiệu quả của các thuật toán Machine Learning, đặc biệt là những thuật toán dựa trên đạo hàm như Gradient Descent.



Dòng lệnh này không làm thay đổi dữ liệu, nó chỉ khởi tạo một đối tượng mô hình Gradient Boosting với các tham số đã cho.

Trong đoạn code trên, các giá trị được truyền vào cho GradientBoostingClassifier() là:

n\_estimators: số lượng cây trong ensemble. Trong trường hợp này, ta chọn 100 cây.

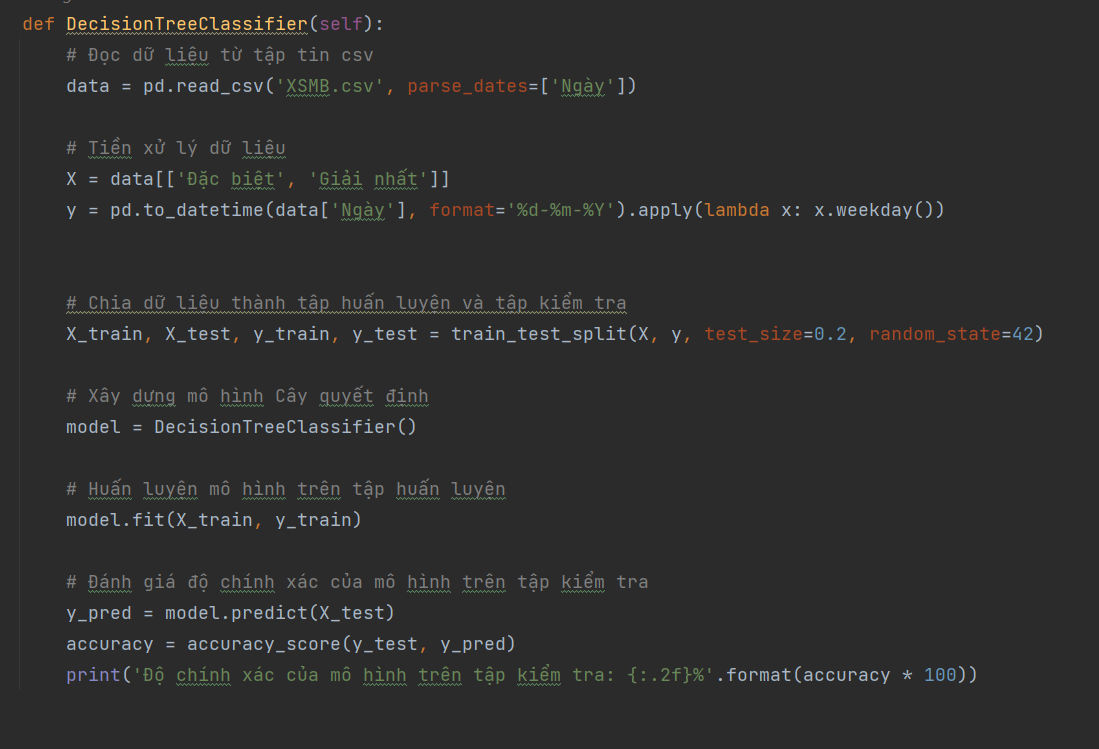
learning\_rate: tỉ lệ học của mô hình. Tức là, cách mà các cây trong ensemble được điều chỉnh để thích nghi với dữ liệu. Tỉ lệ học quá cao có thể dẫn đến quá khớp và tỉ lệ học quá thấp có thể dẫn đến khả năng dự đoán yếu. Trong trường hợp này, ta chọn tỉ lệ học là 0.1.

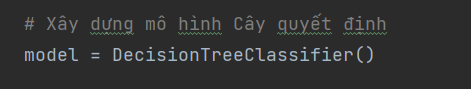
max\_depth: độ sâu tối đa của mỗi cây. Đây là một giới hạn cho số lượng các quyết định sẽ được thực hiện trong mỗi cây. Trong trường hợp này, ta chọn độ sâu tối đa của mỗi cây là 3.

random\_state: giá trị này được sử dụng để giữ cho kết quả của mô hình được tái lập khi chạy nhiều lần. Nếu ta không thiết lập giá trị này, các kết quả có thể khác nhau khi chạy nhiều lần. Trong trường hợp này, ta chọn giá trị random\_state là 42.

**Cây quyết định**

Cây quyết định (Decision tree) là một mô hình học máy phi tuyến tính được sử dụng cho bài toán phân loại (classification) và hồi quy (regression). Cây quyết định thường được sử dụng để dự đoán giá trị của một biến mục tiêu (target variable) bằng cách xây dựng một cây nhị phân hoặc đa nhánh với mỗi nút cây là một câu hỏi dựa trên giá trị của các biến đầu vào (predictor variables) và có hai hay nhiều nhánh dẫn đến các nút con. Cây quyết định được sử dụng phổ biến do tính đơn giản và dễ hiểu của nó, có thể giải thích kết quả một cách rõ ràng, và có khả năng xử lý các biến đầu vào liên tục và rời rạc.





Sau khi dòng lệnh này được thực thi, biến model sẽ chứa một mô hình Cây quyết định được khởi tạo với các tham số mặc định của lớp DecisionTreeClassifier. Mô hình này sẽ được sử dụng để huấn luyện và dự đoán trên dữ liệu sau đó.

**Mạng neural**

Mạng neural (neural network) là một mô hình toán học được lấy cảm hứng từ cách hoạt động của hệ thống thần kinh sinh học trong não người và động vật. Nó được sử dụng để giải quyết nhiều bài toán khác nhau như phân loại, dự đoán, nhận dạng hình ảnh, giọng nói, xử lý ngôn ngữ tự nhiên, v.v.

Mạng neural bao gồm một số lượng lớn các nơ-ron nhân tạo (artificial neuron) được tổ chức thành các lớp, trong đó các nơ-ron trong lớp thứ i được kết nối với các nơ-ron trong lớp thứ i-1. Các kết nối này có trọng số được điều chỉnh trong quá trình huấn luyện, giúp mô hình học được các mối quan hệ và đặc trưng của dữ liệu đầu vào để có thể dự đoán đầu ra chính xác.

Quá trình huấn luyện mạng neural thường được thực hiện thông qua thuật toán lan truyền ngược (backpropagation), trong đó độ lỗi (loss) được tính toán dựa trên đầu ra thực tế và đầu ra được dự đoán, sau đó độ lỗi này được lan truyền ngược từ lớp cuối cùng đến lớp đầu tiên để cập nhật trọng số và cải thiện độ chính xác của mô hình.



Hàm Neural trong đoạn code trên thực hiện các bước để huấn luyện một mô hình mạng neural để dự đoán ngày trong tuần dựa trên kết quả xổ số (đặc biệt và giải nhất) của các ngày trong tập dữ liệu XSMB.csv. Cụ thể:

Đọc dữ liệu từ tập tin XSMB.csv và tiền xử lý dữ liệu bằng cách chọn các cột Đặc biệt và Giải nhất làm đặc trưng và chuyển đổi cột Ngày sang dạng số nguyên biểu diễn ngày trong tuần (từ 0 đến 6, với 0 là Chủ nhật).

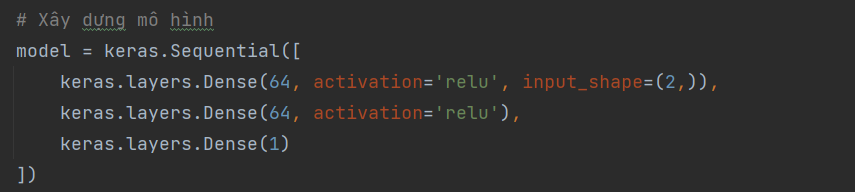
Chuẩn hóa dữ liệu đặc trưng bằng phương pháp StandardScaler để đưa các giá trị đặc trưng về cùng phạm vi.

Chia dữ liệu thành hai tập huấn luyện và kiểm tra với tỷ lệ 80:20.

Xây dựng mô hình mạng neural với 3 lớp: hai lớp ẩn với 64 nơ-ron sử dụng hàm kích hoạt ReLU và một lớp đầu ra với 1 nơ-ron để dự đoán ngày trong tuần. Hàm mất mát được sử dụng là mse và thuật toán tối ưu hóa là adam.

Huấn luyện mô hình với 100 epoch và batch size là 32, với 20% dữ liệu huấn luyện được sử dụng để đánh giá mô hình sau mỗi epoch.

Đánh giá mô hình trên tập kiểm tra bằng hàm mất mát và dự đoán giá trị ngày trong tuần với tập kiểm tra.



Đoạn code này đã khởi tạo một mô hình neural network sử dụng thư viện Keras. Mô hình được định nghĩa gồm 3 lớp: hai lớp ẩn (hidden layer) với 64 node ở mỗi lớp và hàm kích hoạt relu, và một lớp đầu ra (output layer) với 1 node. Các lớp ẩn được kết nối đầy đủ với lớp trước đó. Input shape của mô hình được đặt là (2,), tương ứng với 2 features đầu vào của dữ liệu.

Sau khi khởi tạo, mô hình chưa được huấn luyện nên các trọng số (weights) của các node trong mạng neural chưa được xác định. Các trọng số này sẽ được cập nhật trong quá trình huấn luyện để tối ưu hóa mô hình.



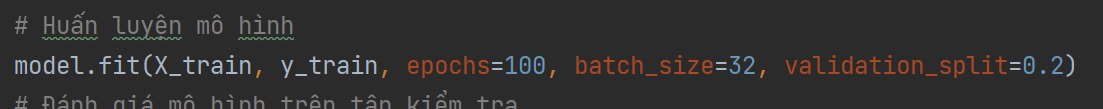
model.compile() được sử dụng để cấu hình quá trình huấn luyện mô hình trên dữ liệu đầu vào. Hàm này có các tham số quan trọng sau:

optimizer: Xác định phương thức tối ưu hoá mô hình trong quá trình huấn luyện. Adam là một phương thức tối ưu phổ biến trong Deep Learning.

loss: Xác định hàm mất mát (loss function) sử dụng để đánh giá mức độ sai lệch giữa dự đoán và kết quả thực tế trong quá trình huấn luyện. Mất mát được tính theo phương pháp Mean Squared Error (MSE).

metrics: Sử dụng để đánh giá hiệu suất của mô hình trên tập dữ liệu huấn luyện và kiểm tra. Trong ví dụ này, không có metrics nào được định nghĩa.

Sau khi cấu hình, model.compile() trả về một mô hình đã được cấu hình sẵn để được huấn luyện trên dữ liệu đầu vào đã được chuẩn bị.



Dòng code này thực hiện huấn luyện mô hình với các tham số như sau:

X\_train: Tập dữ liệu đầu vào cho huấn luyện.

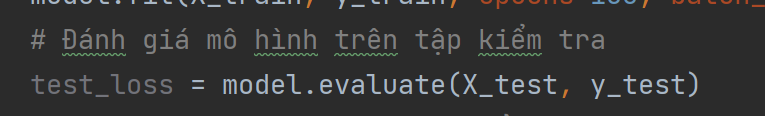
y\_train: Tập dữ liệu đầu ra cho huấn luyện.

epochs: Số lần duyệt qua toàn bộ tập dữ liệu huấn luyện.

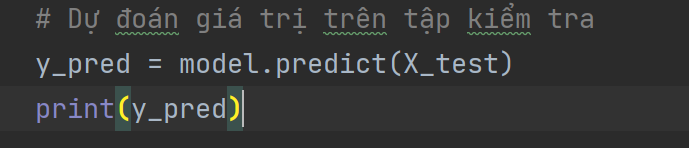
batch\_size: Số lượng mẫu dữ liệu được sử dụng để cập nhật trọng số của mô hình trong mỗi lần huấn luyện.

validation\_split: Tỷ lệ dữ liệu được sử dụng cho việc kiểm tra mô hình trong quá trình huấn luyện.

Trong quá trình huấn luyện, mô hình sẽ sử dụng giải thuật lan truyền ngược để tính toán gradient của hàm mất mát (mse) và cập nhật trọng số của các tầng trong mô hình. Mỗi lần duyệt qua toàn bộ tập dữ liệu huấn luyện (1 epoch), mô hình sẽ được kiểm tra trên tập dữ liệu kiểm tra (được chọn từ tập dữ liệu huấn luyện dựa trên validation\_split) để đánh giá hiệu suất của nó. Quá trình huấn luyện sẽ dừng lại sau khi hoàn thành số lượng epoch được chỉ định.



Sau khi đánh giá mô hình trên tập kiểm tra, biến test\_loss sẽ chứa giá trị loss trung bình trên tập kiểm tra. Giá trị loss được tính dựa trên hàm mất mát (loss function) đã được định nghĩa trong bước model.compile(). Trong trường hợp này, hàm mất mát là "mse" (mean squared error), tức là trung bình của bình phương sai số giữa giá trị dự đoán và giá trị thực tế trên tập kiểm tra. Các giá trị loss thấp hơn cho thấy mô hình đang dự đoán tốt trên tập kiểm tra.

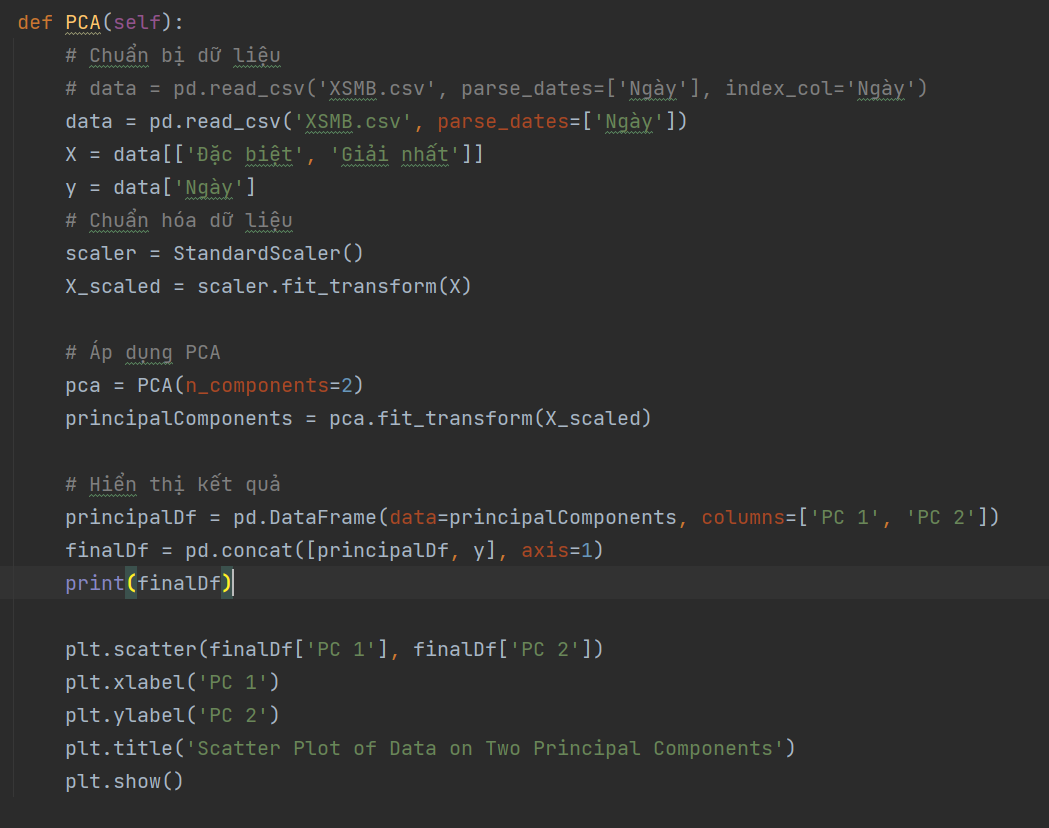


Đoạn mã trên sử dụng mô hình mạng neural đã được huấn luyện để dự đoán giá trị trên tập kiểm tra X\_test. Sau khi dự đoán xong, giá trị được lưu trong biến y\_pred, có cùng kích thước với y\_test.

Ở đây, đầu ra của mạng neural được thiết lập là một giá trị duy nhất (Dense(1)), do đó y\_pred là một mảng 1 chiều chứa các giá trị dự đoán tương ứng cho từng mẫu trong tập kiểm tra. Cụ thể, nếu tập kiểm tra có n mẫu, thì y\_pred sẽ là một mảng có kích thước (n,).

**Phân tích thành phần chính (PCA)**

Phân tích thành phần chính (PCA) là một phương pháp thống kê dùng để giảm số chiều của dữ liệu bằng cách tìm kiếm các thành phần chính của dữ liệu. Thành phần chính là một tổ hợp tuyến tính của các biến đầu vào, với mục tiêu tối đa hoá phương sai của dữ liệu. PCA có thể được sử dụng để giảm số chiều của dữ liệu bằng cách chọn một số lượng thành phần chính nhất định mà vẫn giữ được độ phân biệt của dữ liệu. Điều này giúp cho việc xử lý dữ liệu trở nên dễ dàng hơn, đồng thời giảm thiểu sự mất mát thông tin và tăng độ chính xác của các mô hình dữ liệu được xây dựng trên các dữ liệu đã được giảm số chiều.



Đoạn code sử dụng thuật toán PCA (Principal Component Analysis) để giảm số chiều của dữ liệu từ 2 chiều xuống còn 2 chiều mới (2 thành phần chính) sao cho giữ lại được thông tin quan trọng nhất của dữ liệu.

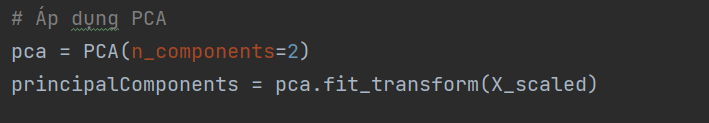
Chu kỳ của thuật toán gồm các bước:

Chuẩn bị dữ liệu bao gồm đọc dữ liệu từ file csv, chọn ra các features cần xử lý, chuẩn hóa dữ liệu.

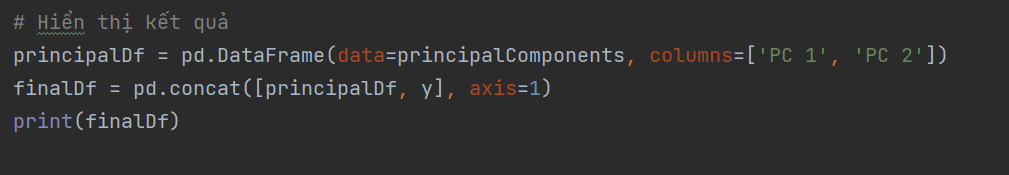
Áp dụng PCA với số lượng thành phần chính là 2.

Hiển thị kết quả bằng cách trực quan hóa dữ liệu trên 2 thành phần chính.

Tóm lại, thuật toán PCA được sử dụng để giảm số chiều của dữ liệu, giúp dễ dàng quan sát và trực quan hóa dữ liệu.



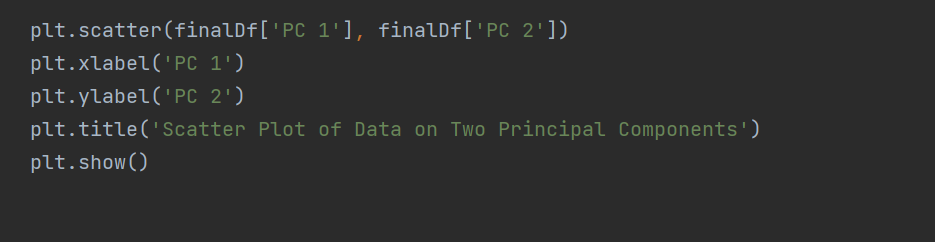
Sau khi thực hiện đoạn code này, các giá trị của dữ liệu ban đầu được biến đổi thành các thành phần chính mới, tương ứng với các giá trị được lưu trữ trong biến principalComponents. Cụ thể, n\_components=2 trong PCA nghĩa là thuật toán sẽ giữ lại 2 thành phần chính của dữ liệu, do đó principalComponents sẽ là một mảng numpy 2 chiều, trong đó mỗi hàng đại diện cho một quan sát trong dữ liệu ban đầu và mỗi cột đại diện cho một thành phần chính.



Sau khi thực hiện phân tích thành phần chính (PCA) trên dữ liệu, đoạn code này sẽ tạo ra một DataFrame principalDf chứa hai cột mới là 'PC 1' và 'PC 2', đại diện cho hai thành phần chính của dữ liệu.

Sau đó, nó sử dụng hàm pd.concat() để nối cột 'PC 1' và 'PC 2' với cột 'Ngày' của dữ liệu gốc X, để tạo ra một DataFrame mới finalDf.

Kết quả được in ra là finalDf với các cột 'PC 1', 'PC 2', và 'Ngày'.



Đoạn code này sử dụng thư viện matplotlib để vẽ một biểu đồ phân tán (scatter plot) của dữ liệu đã được giảm chiều trên hai thành phần chính (principal components). Cụ thể, trục x và y của biểu đồ tương ứng với hai thành phần chính (PC 1 và PC 2). Các điểm trên biểu đồ thể hiện các mẫu dữ liệu (sample), mỗi điểm tương ứng với một mẫu dữ liệu và vị trí của điểm trên biểu đồ là giá trị của mẫu dữ liệu trên hai thành phần chính. Biểu đồ giúp chúng ta có cái nhìn tổng quan về cách dữ liệu phân bố trên hai thành phần chính và có thể sử dụng để phát hiện các cụm (cluster) dữ liệu hoặc các quan hệ giữa các mẫu dữ liệu.

**Association Rule Mining**

Association Rule Mining (khai thác luật kết hợp) là một kỹ thuật phân tích dữ liệu để tìm kiếm các quy luật tương quan giữa các mục trong tập dữ liệu. Ví dụ, khi một khách hàng mua một sản phẩm, họ có thể cũng sẽ mua các sản phẩm khác. Từ đó, Association Rule Mining có thể giúp phát hiện các quy luật như "Nếu khách hàng mua sản phẩm A thì họ cũng có thể mua sản phẩm B". Kỹ thuật này có thể được sử dụng trong nhiều lĩnh vực như marketing, bán lẻ, y tế và khoa học dữ liệu.



Chu kỳ của đoạn code này là:

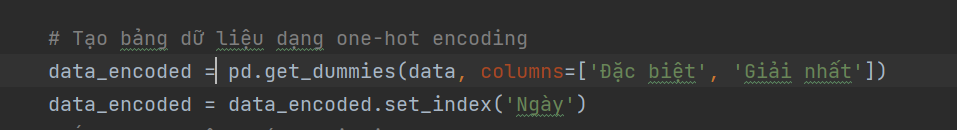
Load dữ liệu

Chuyển đổi dữ liệu thành dạng one-hot encoding

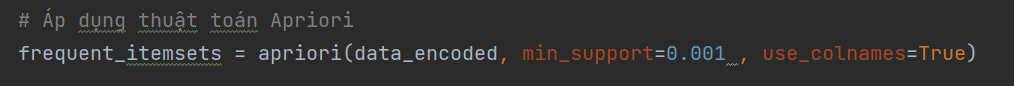
Áp dụng thuật toán Apriori để tìm frequent itemsets

Sử dụng frequent itemsets để tìm các luật kết hợp

In kết quả.



Đoạn code này sử dụng phương pháp one-hot encoding để chuyển đổi các giá trị của cột 'Đặc biệt' và 'Giải nhất' sang dạng các cột nhị phân (0 hoặc 1) tương ứng với mỗi giá trị trong cột ban đầu. Sau đó, nó đặt cột 'Ngày' làm chỉ mục cho bảng dữ liệu và lưu trữ bảng dữ liệu đã chuyển đổi vào biến data\_encoded.



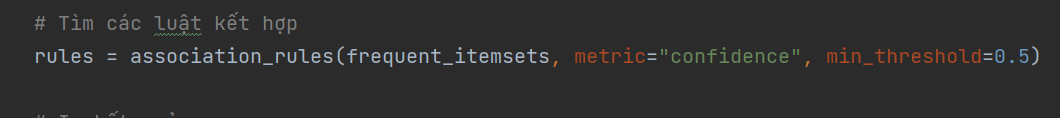
Đoạn code này sử dụng hàm apriori từ thư viện mlxtend.frequent\_patterns để áp dụng thuật toán Apriori trên bảng dữ liệu đã được mã hóa dạng one-hot từ trước. Các tham số của hàm bao gồm:

data\_encoded: bảng dữ liệu đã được mã hóa dạng one-hot.

min\_support: giá trị support tối thiểu của các tập phổ biến, được sử dụng để loại bỏ các tập ít phổ biến. Ở đây, giá trị là 0.001, nghĩa là các tập xuất hiện ít nhất 0.1% trong các giao dịch sẽ không được tính.

use\_colnames: giá trị True để sử dụng tên cột thay vì chỉ số cột.

Kết quả trả về là bảng chứa các tập phổ biến cùng với giá trị support của chúng.



Đoạn code này sử dụng kết quả frequent\_itemsets từ thuật toán Apriori để tìm các luật kết hợp, với các ràng buộc như chỉ chọn các luật có độ tin cậy (confidence) lớn hơn hoặc bằng 0.5. Cụ thể, confidence là một phép đo độ chắc chắn của một luật, nó được tính bằng tỉ lệ giữa số lần xuất hiện của cả itemset và của itemset phía trước của luật trong tập dữ liệu.

Trong đoạn code trên, ta cần cung cấp các tham số cho thuật toán Apriori để có thể tìm ra các mục phổ biến và tạo ra các luật kết hợp từ chúng. Các tham số đó là:

data\_encoded: dữ liệu đã được chuyển đổi dưới dạng one-hot encoding

min\_support: mức hỗ trợ tối thiểu của các mục. Giá trị này được đặt là 0.001, có nghĩa là các mục xuất hiện trong ít nhất 0.1% các giao dịch sẽ được xem là phổ biến.

use\_colnames: sử dụng tên cột thay vì số để đại diện cho các mục.

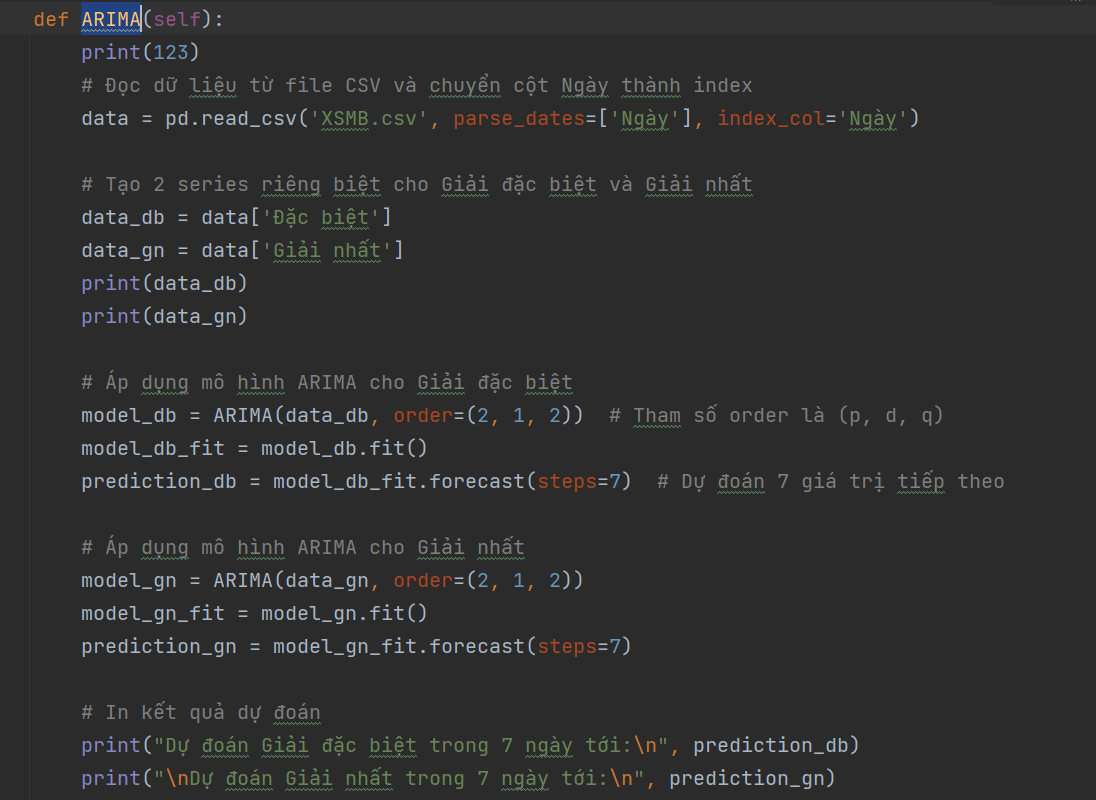
metric: chỉ số được sử dụng để đánh giá chất lượng của các luật kết hợp. Ở đây ta sử dụng chỉ số confidence.

min\_threshold: ngưỡng tối thiểu của chỉ số metric để một luật kết hợp được xem là có ý nghĩa. Giá trị này được đặt là 0.5, có nghĩa là các luật kết hợp với confidence ít nhất bằng 0.5 sẽ được giữ lại.

**ARIMA**

ARIMA (Autoregressive Integrated Moving Average) là một mô hình dự báo chuỗi thời gian (time series) phổ biến trong lĩnh vực kinh tế, tài chính và dự báo thị trường. Mô hình này được sử dụng để mô hình hóa các chuỗi thời gian không có xu hướng (stationary time series) bằng cách kết hợp giữa các thành phần autoregressive (AR), moving average (MA) và integration (I).

ARIMA có thể giúp dự báo giá trị của một biến dự báo (dependent variable) trong tương lai dựa trên các giá trị của biến này và các biến độc lập (independent variables) trong quá khứ. Các thông số của mô hình ARIMA được ước tính thông qua quá trình phân tích chuỗi thời gian, đánh giá mô hình và điều chỉnh các thông số cho đến khi đạt được mô hình tốt nhất.



Chu kỳ đoạn code của bạn là sử dụng mô hình ARIMA để dự đoán giá trị của Giải đặc biệt và Giải nhất trong 7 ngày tiếp theo dựa trên dữ liệu trong file CSV. Cụ thể, đoạn code sẽ thực hiện các bước sau:

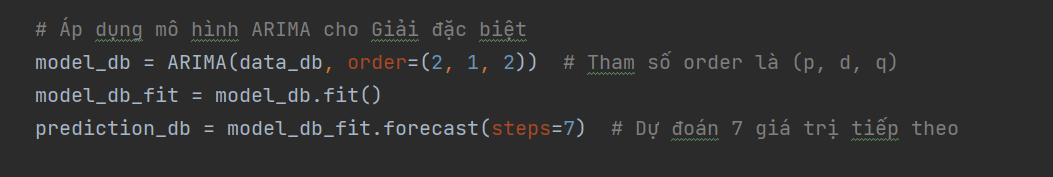
Đọc dữ liệu từ file CSV và chuyển cột Ngày thành index.

Tạo 2 series riêng biệt cho Giải đặc biệt và Giải nhất.

Áp dụng mô hình ARIMA cho Giải đặc biệt và Giải nhất.

Dự đoán giá trị của Giải đặc biệt và Giải nhất trong 7 ngày tiếp theo.

In kết quả dự đoán.



Đoạn code này khởi tạo một mô hình ARIMA với dữ liệu data\_db là chuỗi dữ liệu cần dự đoán, và tham số order=(2,1,2) được sử dụng để xác định mô hình ARIMA, trong đó:

p là số lượng lags (độ trễ) của biến phụ thuộc được bao gồm trong mô hình

d là độ cao của việc khác biệt giữa các giá trị trước và sau, để giúp ổn định phương sai

q là số lượng lags (độ trễ) của sai số mô hình được bao gồm trong mô hình.

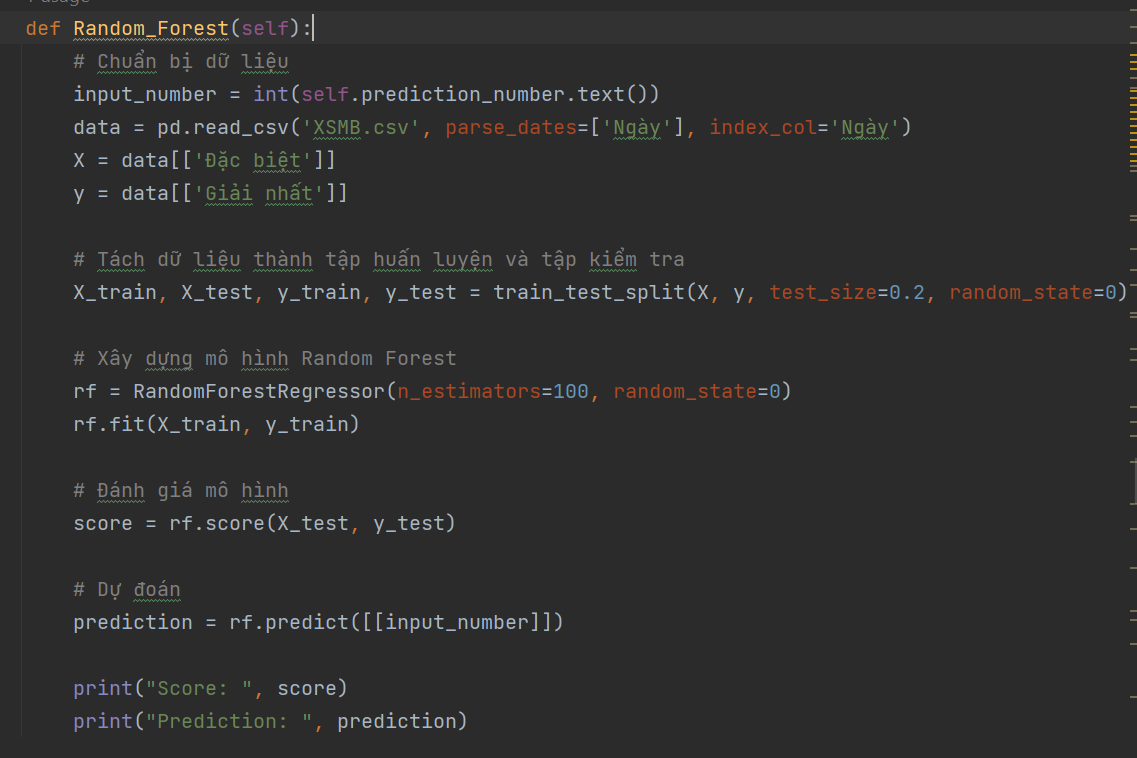
Trong trường hợp này, order=(2,1,2) có nghĩa là mô hình sẽ bao gồm 2 lags của biến phụ thuộc, có độ cao của việc khác biệt giữa các giá trị trước và sau là 1 (first difference), và 2 lags của sai số mô hình.

áp dụng mô hình ARIMA cho dữ liệu của Giải nhất (data\_gn), sử dụng tham số order=(2,1,2), tức là mô hình ARIMA với 2 lags của dữ liệu đã sai phân và 2 lags của sai số dự báo. Sau đó, chúng ta khớp mô hình với dữ liệu bằng phương pháp fit() và dự đoán 7 giá trị tiếp theo của dữ liệu bằng phương pháp forecast(steps=7).

**Random Forest**

Random Forest là một thuật toán học máy dựa trên phương pháp Ensemble Learning, nó kết hợp nhiều cây quyết định (decision tree) để tạo ra một mô hình phân loại hoặc dự đoán chính xác hơn. Mỗi cây quyết định được xây dựng trên một phần của dữ liệu được lấy ngẫu nhiên và với một số đặc trưng (features) được chọn ngẫu nhiên. Khi tạo cây quyết định, tại mỗi nút, các đặc trưng được chọn sẽ được sử dụng để tạo ra một quyết định phân nhánh. Quyết định phân nhánh sẽ được dựa trên việc giảm giá trị của một hàm mất mát được chọn.

Kết quả dự đoán của mô hình Random Forest được xác định bằng cách lấy trung bình các kết quả dự đoán của từng cây quyết định. Điều này giúp giảm thiểu overfitting, tức là mô hình tập trung quá nhiều vào dữ liệu huấn luyện và không thể áp dụng được cho các dữ liệu mới. Random Forest cũng cho phép đánh giá độ quan trọng của các đặc trưng được sử dụng trong việc phân loại hoặc dự đoán.



Đoạn mã trên thực hiện các bước sau:

Đọc dữ liệu từ file csv và chọn cột "Đặc biệt" làm input và cột "Giải nhất" làm output.

Tách dữ liệu thành tập huấn luyện và tập kiểm tra.

Xây dựng mô hình Random Forest với 100 cây quyết định.

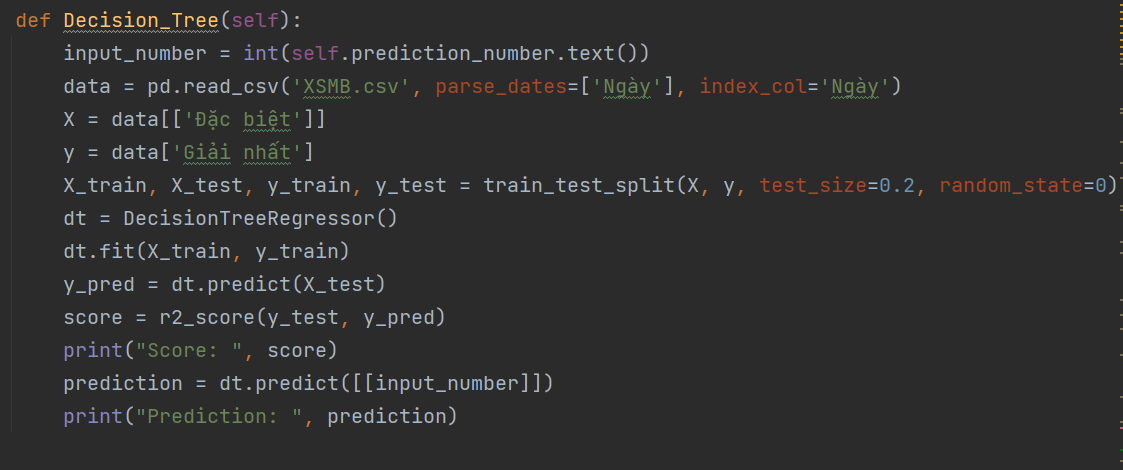
Đánh giá mô hình trên tập kiểm tra và tính toán điểm số.

Dự đoán giải nhất với giá trị input nhập vào.

In kết quả dự đoán và điểm số của mô hình.

**Decision Tree**

Decision Tree (Cây quyết định) là một trong những phương pháp học có giám sát được sử dụng trong Machine Learning, trong đó mô hình được xây dựng dưới dạng cây phân nhánh để dự đoán giá trị đầu ra của một quan sát dựa trên giá trị của các biến đầu vào. Cây quyết định được xây dựng bằng cách chia tập dữ liệu huấn luyện thành các phân nhóm con dựa trên các thuộc tính, để tạo ra các nút và nhánh trong cây. Khi dữ liệu mới được đưa vào, cây sẽ dự đoán giá trị đầu ra bằng cách đi xuống từ nút gốc đến nút lá tương ứng.



Đoạn code trên sử dụng thuật toán Decision Tree để dự đoán giải nhất của xổ số miền Bắc dựa trên giải đặc biệt, với các bước:

Đọc dữ liệu từ file CSV và tách thành 2 series riêng biệt cho giải đặc biệt và giải nhất.

Tách dữ liệu thành tập huấn luyện và tập kiểm tra.

Xây dựng mô hình Decision Tree trên tập huấn luyện.

Đánh giá mô hình bằng cách tính toán hệ số R2 trên tập kiểm tra.

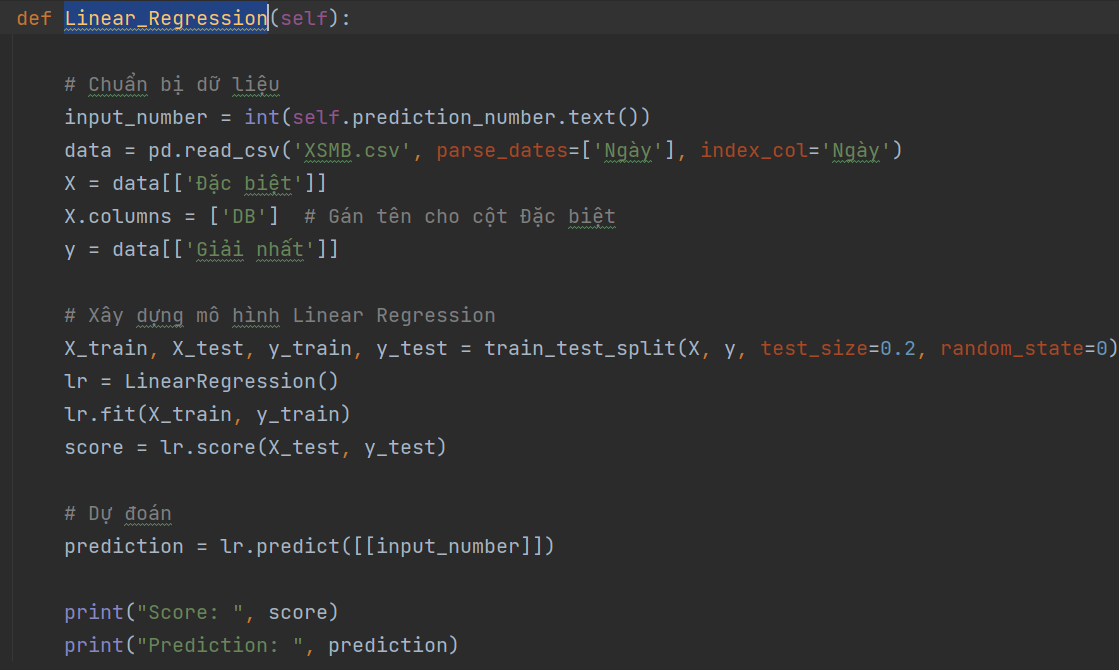
Dự đoán giải nhất cho một giải đặc biệt được nhập vào từ người dùng.

In ra kết quả dự đoán và hệ số R2.

Vì vậy, chu kỳ của đoạn mã trên là đọc dữ liệu từ file CSV, xử lý và chuyển đổi dữ liệu, xây dựng mô hình, đánh giá mô hình và dự đoán.

**Linear Regression**

Linear Regression là một trong những thuật toán học máy đơn giản nhất, được sử dụng để tìm mối quan hệ tuyến tính giữa các biến đầu vào và biến đầu ra. Nó dựa trên giả thiết rằng có một mối quan hệ tuyến tính giữa biến đầu vào và biến đầu ra. Thuật toán tìm cách xác định các trọng số của biến đầu vào để có thể dự đoán được giá trị của biến đầu ra. Cụ thể, trong Linear Regression, một mô hình được xây dựng bằng cách tính toán các trọng số sao cho tổng bình phương sai số giữa dự đoán và giá trị thực tế là nhỏ nhất. Các trọng số này sẽ được sử dụng để dự đoán giá trị của biến đầu ra cho các giá trị mới của các biến đầu vào.



Đoạn mã trên dùng thuật toán Linear Regression để xây dựng mô hình dự đoán kết quả xổ số. Chu kỳ của nó tương tự như các đoạn mã trước đó, phụ thuộc vào số lần thực hiện và thời gian thực hiện.