TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN HỌC: NHẬP MÔN HỌC MÁY

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**

Người thực hiện: **NGUYỄN THÁI HÒA - 52100413**

PHAN THỊ THÙY LINH – 52100697

TRẦN THỊ ANH THƯ - 52100489

Lớp **: 21050401**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN HỌC: NHẬP MÔN HỌC MÁY

Mã môn học: 503044

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**

Người thực hiện: **NGUYỄN THÁI HÒA - 52100413**

PHAN THỊ THÙY LINH – 52100697

TRẦN THỊ ANH THƯ - 52100489

Lớp **: 21050401**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Môn học Nhập môn Học máy thực sự rất bổ ích, đã bổ trợ thêm cho chúng em rất nhiều kiến thức liên quan đến chuyên ngành của mình. Em xin cảm ơn sự quan tâm của nhà trường và khoa CNTT khi đã đưa bộ môn vào giảng dạy. Đặc biệt em xin gửi lời cảm ơn chân thành đến giảng viên bộ môn – thầy PGS.TS. Lê Anh Cường đã luôn ân cần giảng dạy, giải đáp tận tình những thắc mắc của chúng em, giúp chúng em có thể hiểu và làm bài một cách đơn giản nhất.

Bài báo cáo này là minh chứng rõ nhất cho những cố gắng, nỗ lực học tập của nhóm em. Tuy vậy sự tiếp thu kiến thức của chúng em chỉ mới ở mức cơ bản và còn nhiều thiếu sót, kính mong thầy có thể cho chúng em thêm nhận xét và góp ý để bài báo cáo có thể hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

BÁO CÁO ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy PGS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 22 tháng 10 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Hòa

Nguyễn Thái Hòa

Linh

Phan Thị Thùy Linh

Thư

Trần Thị Anh Thư

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Theo yêu cầu bài 2, chúng em tiến hành chọn một bộ dữ liệu phù hợp với yêu cầu đề ra và tiến hành phân tích, tìm hiểu bộ dữ liệu. Sau đó, sử dụng các mô hình học máy cơ bản để giải quyết bài toán. Áp dụng kĩ thuật tránh Overfitting trên tất cả các mô hình huấn luyện. Sau khi huấn luyện xong, tiến hành phân tích các trường hợp sai để đề ra giải pháp và cải thiện độ chính xác của mô hình.

MỤC LỤC

DANH MỤC CÁC CHỮ VIẾT TẮT

1. kNN- K-nearest neighbor
2. MSE- Mean Squared Error
3. RSS - Residual Sum of Squares
4. RMSE- Root Mean Squared Error
5. RSS - Residual Sum of Squares
6. SGD- Stochastic Gradient Descent

DANH SÁCH CÁC BẢNG BIỂU, HÌNH VẼ

CHƯƠNG 1 – TỔNG QUAN

1.1 Giới thiệu và phân tích bài toán:

1.1.1 Giới thiệu bài toán:

Dựa trên những yêu cầu đề ra, chúng em tiến hành tìm hiểu, phân tích và chọn Heart Attack Analysis & Prediction Dataset là bài toán dự đoán có thể giải quyết bằng học máy. Bộ dữ liệu gồm 14 Features như sau:

Features	Details	Datatypes
age	Age of the person	Numerical
sex	Gender of the person	Categorical
cp	Chest Pain type	Categorical
trtbps	Resting blood pressure (in mm Hg)	Numerical
chol	Cholestoral in mg/dl fetched via BMI sensor	Numerical
fbs	(Fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)	Categorical
restecg	Resting electrocardiographic results	Categorical
thalachh	Maximum heart rate achieved	Numerical
exng	Exercise induced angina (1 = yes; 0 = no)	Categorical
oldpeak	Previous peak	Numerical
slp	Slope	Categorical
caa	Number of major vessels (0-3)	Categorical
thall	Thal rate	Categorical
output	Target variable	Categorical

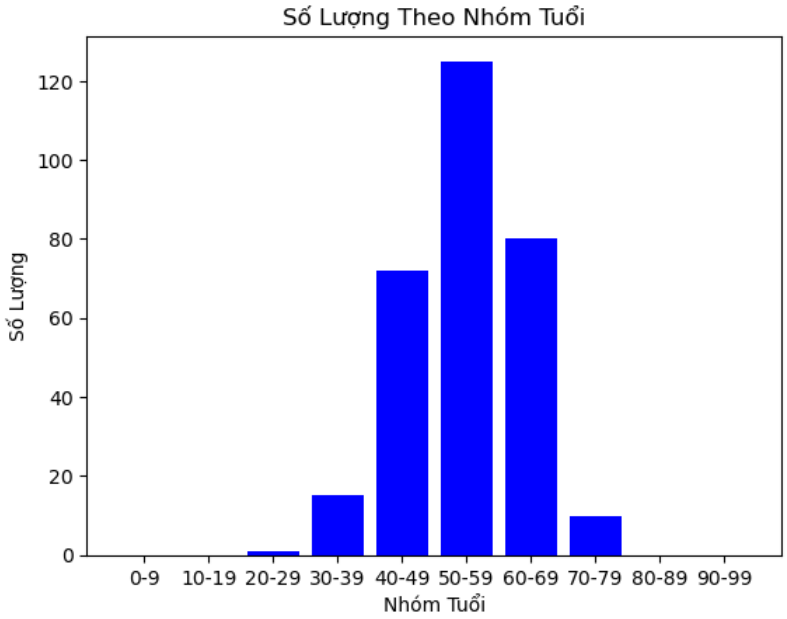
Các features trong Heart Attack Analysis & Prediction Dataset

Mục tiêu của bài toán là sử dụng mô hình học máy để dự đoán khả năng bị bệnh tim dựa trên các thông tin có sẵn của nam và nữ qua khảo sát. Bài toán đưa ra giúp chúng ta có thể hiểu sâu hơn về quá trình xử lý dữ liệu, xây dựng mô hình và đánh giá

hiệu xuất mô hình. Song song đó việc sử dụng các kỹ thuật Overfitting có thể cải thiện hiệu suất của mô hình.

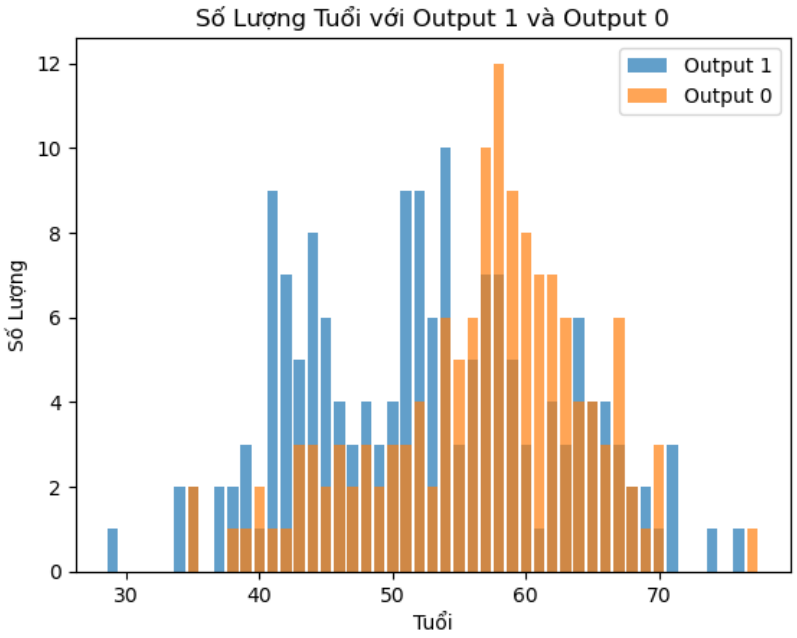
1.1.2 Phân tích dữ liệu bài toán:

- Age: tuổi



Biểu đồ thống kê số lượng tuổi theo từng nhóm tuổi

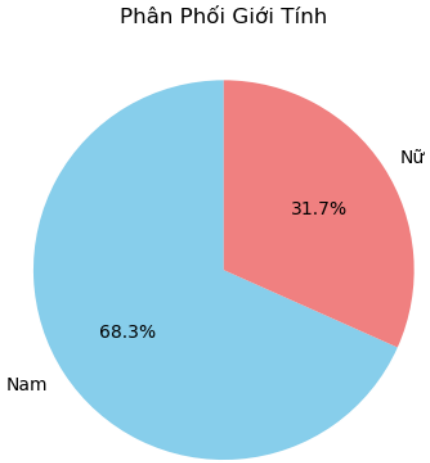
- Nhận xét: Số lượng dữ liệu về tuổi trong data chủ yếu tập trung ở khoảng 40 đến 70 tuổi đây là nhóm tuổi người trưởng thành.



Biểu đồ thống kê số lượng tuổi theo ouput

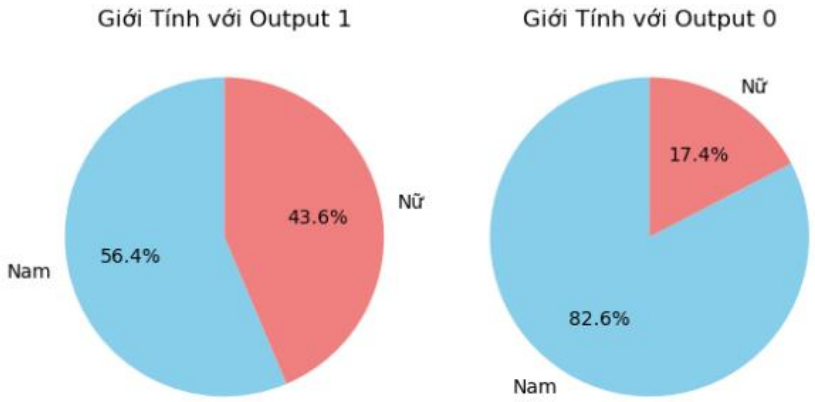
- Nhận xét: Biểu đồ thể hiện màu cam là tỉ lệ tuổi ít mắc bệnh về tim trong data nằm chủ yếu ở khoảng 60 tuổi và độ tuổi có nguy cơ bị bệnh tim nhiều nằm ở khoảng 40 đến 50 tuổi độ tuổi trẻ hơn những người không có nguy cơ mắc bệnh.

- Sex: Giới tính



Biểu đồ phân phối giới tính.

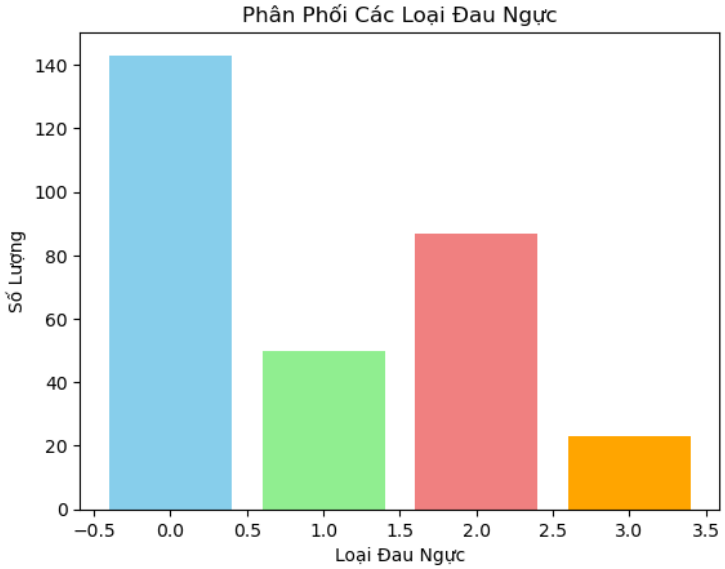
- Nhận xét: Trong tập dữ liệu có số lượng nam nhiều hơn nữ với 60% là nam.



Biểu đồ phân phối giới tính theo Output

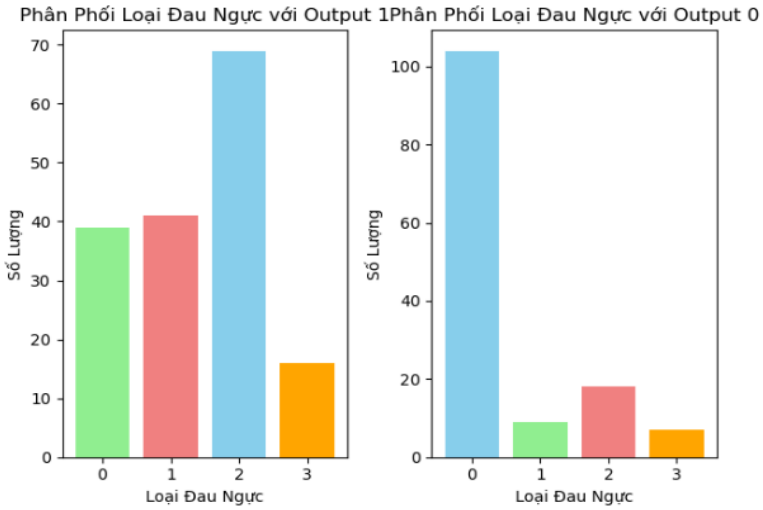
- Nhận xét: Với người có tỉ lệ mắc bệnh tim thì nam cao hơn nữ nhưng với số lượng nữ thấp hơn nam ở ban đầu thì đây là 1 tỉ lệ cao đối với nữ. Vì vậy khi ở người ít mắc bệnh tim thì nam vượt trội hơn so với nữ.

- Có các loại đau ngực:



Biểu đồ phân phối các loại đau ngực

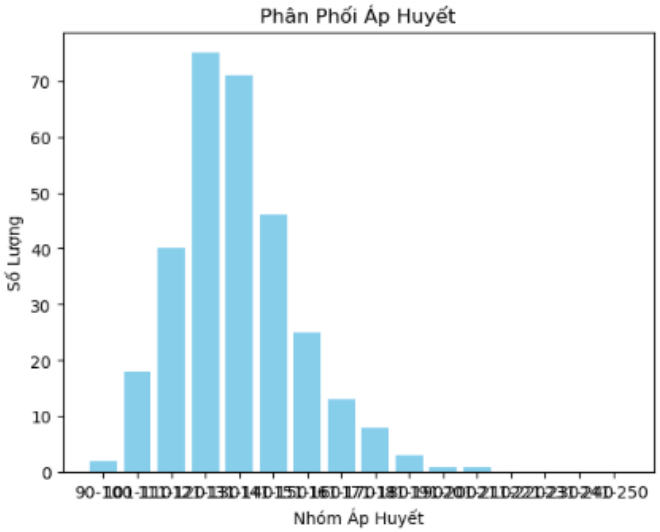
- Nhận xét: Có 4 mức độ đau ngực thì tổng thể số lượng người có triệu chứng đau ngực ít nhất chiếm số lượng lớn và những người đau dữ dội chiếm ít nhất.



Biểu đồ phân phối các loại đau ngực theo output

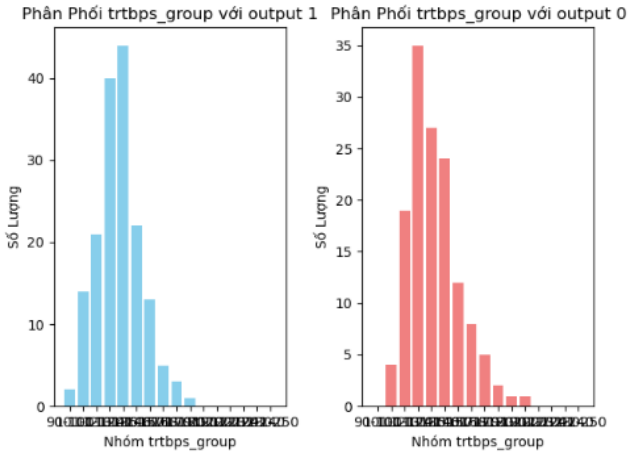
- Nhận xét: Ở phân phối những người có tỉ lệ mắc bệnh đau tim thì số lượng những người đau vừa phải và đau nhiều tăng. Còn ở phân phối những người ít mắc bệnh tim thì số lượng người triệu chứng đau ít vẫn nhiều nhất nhưng những triệu chứng còn lại không đáng kể.

- Huyết áp:



Biểu đồ phân phối nhóm áp huyết

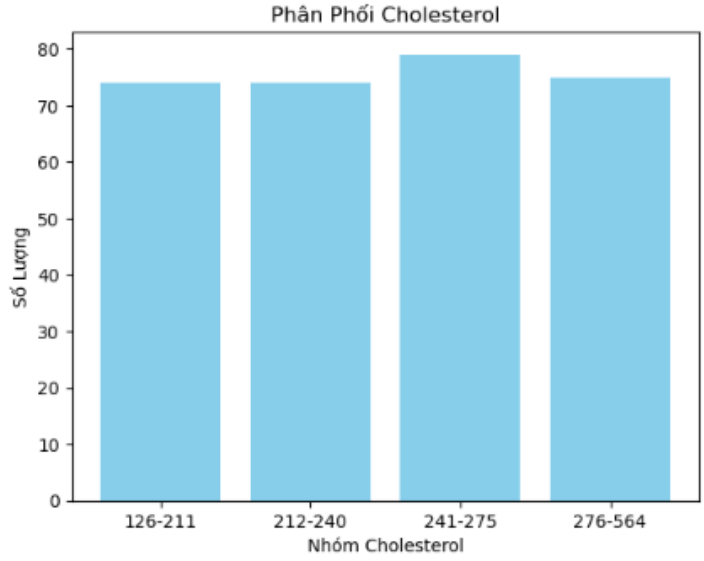
- Nhận xét: Tổng quan data số lượng người đạt huyết áp từ 120 đến 140 chiếm phần lớn và dàn đều ra 2 bên.



Biểu đồ phân phối nhóm áp huyết theo output

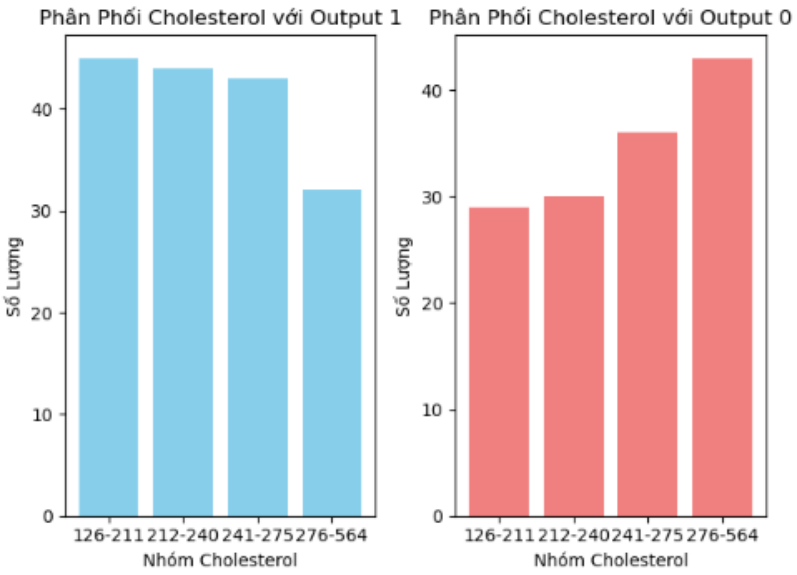
- Nhận xét: Về cơ bản khi nhìn qua biểu đồ so sánh thì huyết áp của những người có nguy cơ và không có nguy cơ bị tim tương đối giống nhau.

- Cholesterol



Biểu đồ phân phối Cholesterol

- Nhận xét: Số lượng người với các số lượng cholesterol có trong máu được dàn đều trong data.



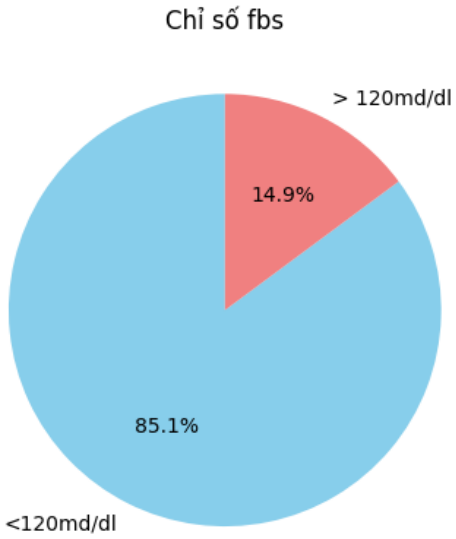
Biểu đồ phân phối Cholesterol theo output

- Nhận xét: Trong sự so sánh trên ta thấy số lượng cholesterol của người có nguy cơ và không có nguy cơ có số lượng tương đương nhau vậy nên ở thông tin này có vẻ không biểu thị rõ ràng về các nguy cơ bị tim.

- Chỉ số đường trong máu(fbs)

Dữ liệu Đường trong máu (FBS) trong bộ dữ liệu trên đo lượng đường trong máu sau khi bạn nhịn ăn ít nhất 8 giờ. Đây thường là xét nghiệm đầu tiên được thực hiện để kiểm tra tiền tiểu đường và tiểu đường.

(lượng đường trong máu > 120 mg/dl) (1 = true; 0 = false)



Biểu đồ thống kê chỉ số fbs

Biểu đồ cho thấy trong khoảng 300 trường hợp, có 14,9 % chỉ số đường huyết lúc đói >120ml/dl, còn lại khoảng 85,1% trường hợp chỉ số đường huyết < 120ml/dl, trong đó chỉ số đường huyết ổn định khi chưa ăn là dưới 120ml/dl, điều này có nghĩa là trong dataset trên, tỉ lệ người có khả năng mắc bệnh đường huyết thấp, tập trung chủ yếu ở độ tuổi. Với số lượng là 258 người chỉ số đường huyết < 120ml/dl; 45 người chỉ số đường huyết lúc đói >120ml/dl.

Nhóm tuổi có lượng đường huyết cao phân bố rộng khắp trên các độ tuổi, không đặc biệt phân bố ở bất kì nhóm tuổi nào.

- Điện tâm đồ (restecg-resting electrocardiographic results)

  Trên dataset cho biết Kết quả điện tâm đồ lúc nghỉ

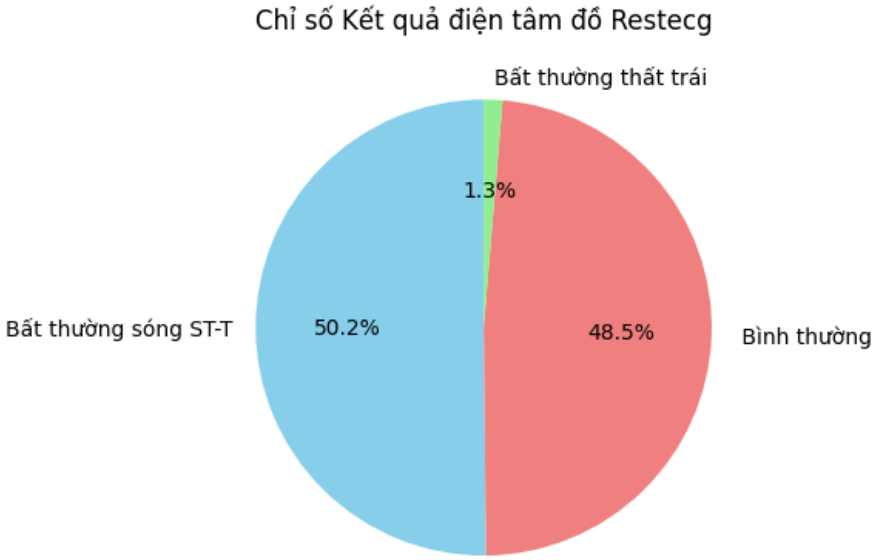
  Giá trị 0: bình thường

  Giá trị 1: có bất thường sóng ST-T (sóng T đảo ngược và/hoặc ST chênh lên hoặc chênh xuống > 0,05 mV)

  Giá trị 2: cho thấy thất trái có thể xảy ra hoặc xác định phì đại theo tiêu chuẩn Estes

Điện tâm đồ, hay còn gọi là đo điện tim (Electrocardiogram, viết tắt ECG, EKG) là một xét nghiệm ghi lại hoạt động điện học của tim dưới dạng đồ thị.

ECG trong bộ Dữ liệu Heart.csv trên được đo khi nghỉ ngơi, còn gọi là điện tim thường để phân biệt với các phương pháp đo điện tim khác (đo khi gắng sức hoặc đo Hotler 24 giờ).



Biểu đồ chỉ số kết quả điện tâm đồ Restecg

Nhóm tuổi có hoặc không bất thường về điện tâm đồ lúc nghỉ phân bố rộng khắp trên các độ tuổi, không đặc biệt phân bố ở bất kì nhóm tuổi nào.

Với số liệu là 152 người có bất thường sóng ST-T, 147 người bình thường, 4 người cho thấy thất trái có thể xảy ra hoặc xác định phì đại theo tiêu chuẩn Estes.

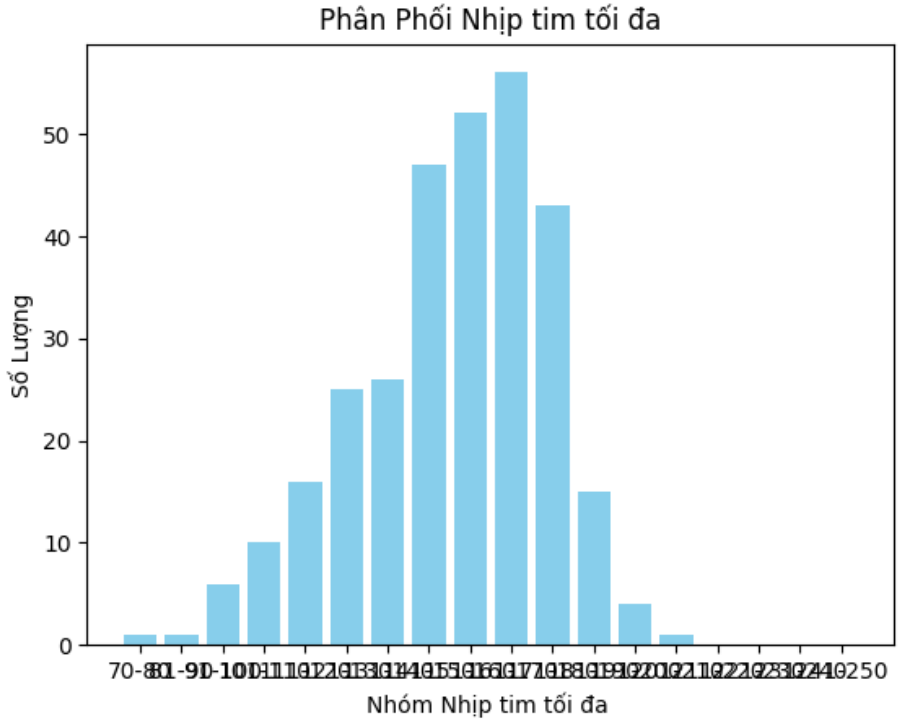
- Nhịp tim tối đa đạt được (thalach-maximum heart rate achieved) (từ 71 đến 202)

Nhịp tim tối đa là nhịp đập khi tim làm việc hết sức để đáp ứng nhu cầu oxy của cơ thể. Áp dụng công thức tính nhịp tim tối đa sẽ giúp bạn biết được phạm vi lý tưởng để giữ cho tim của bạn hoạt động tốt trong khi đang vận động thể chất cường độ cao.

Tập thể dục cường độ cao là cách tốt nhất để giảm nhịp tim nghỉ ngơi và tăng nhịp tim tối đa. Tuy nhiên, bạn nên cẩn thận để không khiến nhịp tim tối đa quá cao, sẽ gây căng thẳng cho cơ thể và có thể dẫn đến những rủi ro sức khỏe.

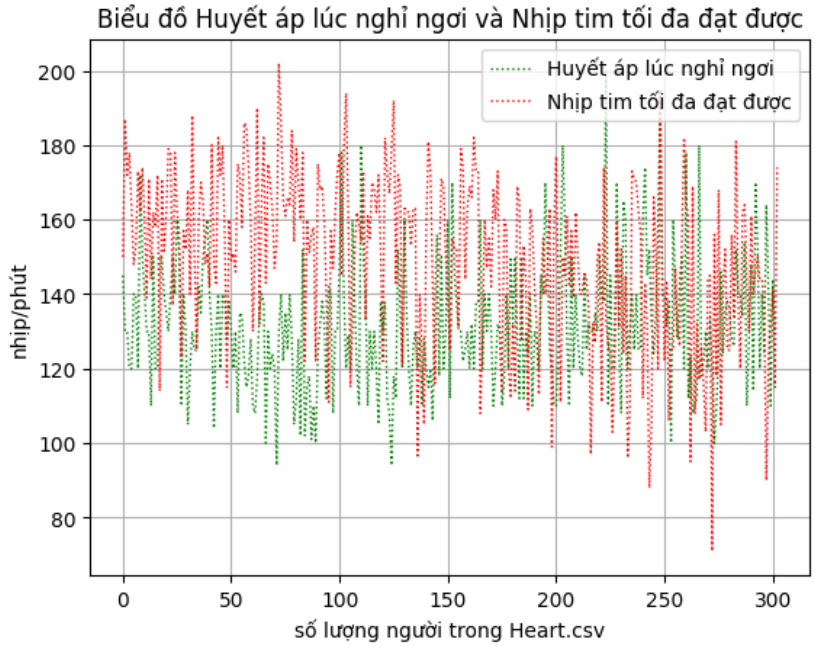
Khi tính trung bình Nhịp tim tối đa, ta nhận được:

- 171 người có Nhịp tim tối đa cao hơn trung bình
- 132 người thấp hơn trung bình



Biểu đồ phân phối nhịp tim tối đa

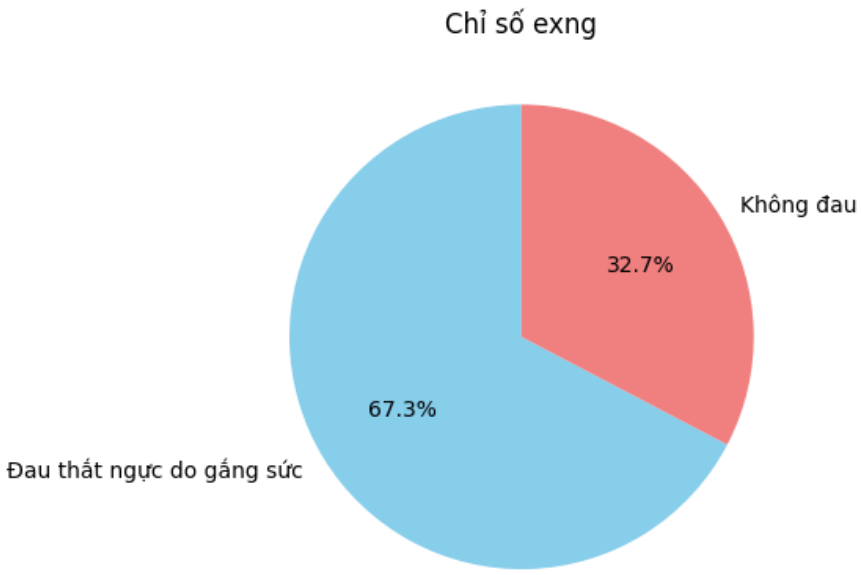
- Nhận xét: Phần lớn người trong dataset có Nhịp tim tối đa từ 101 cho đến 190



Biểu đồ huyết áp lúc nghỉ ngơi và nhịp tim tối đa đạt được

- Đau thắt ngực do gắng sức (exang-exercise induced angina (1 = yes; 0 = no))

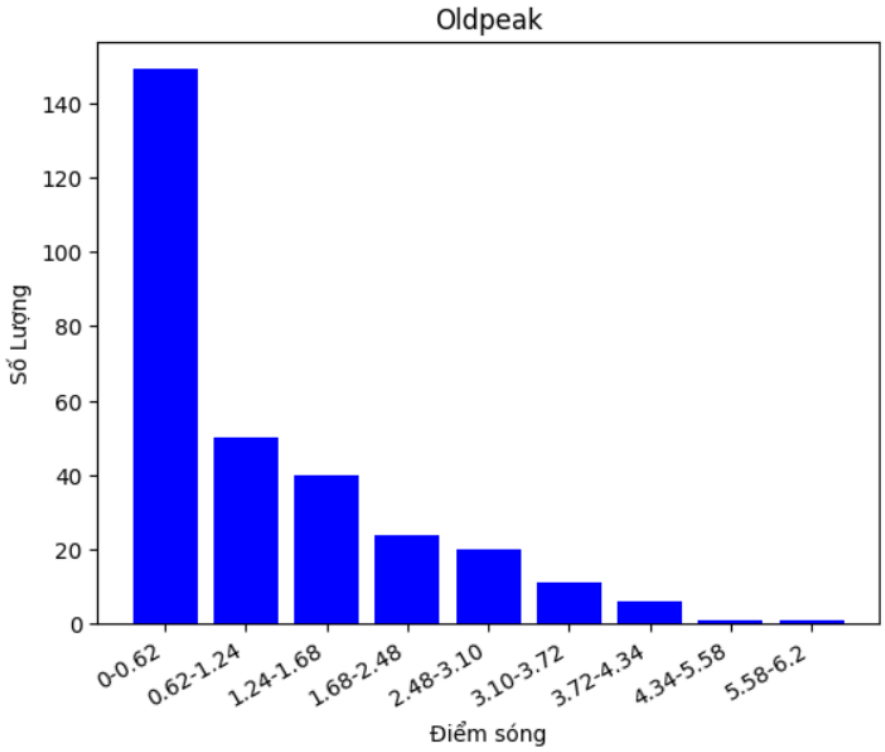
Hở van tim (van 2 lá, van 3 lá) ở mức độ nhẹ (1/4) đơn thuần thì không gây ra triệu chứng trên lâm sàng, nên khả năng triệu chứng đau ngực của bạn không liên quan đến tình trạng hở van tim nhẹ. Hở van tim 1/4 đơn thuần cũng chưa cần điều trị đặc hiệu. Cơn đau ngực kéo dài với thời gian ngắn, tự hết thường không điển hình cho bệnh lý tim mạch.



Biểu đồ chỉ số exng

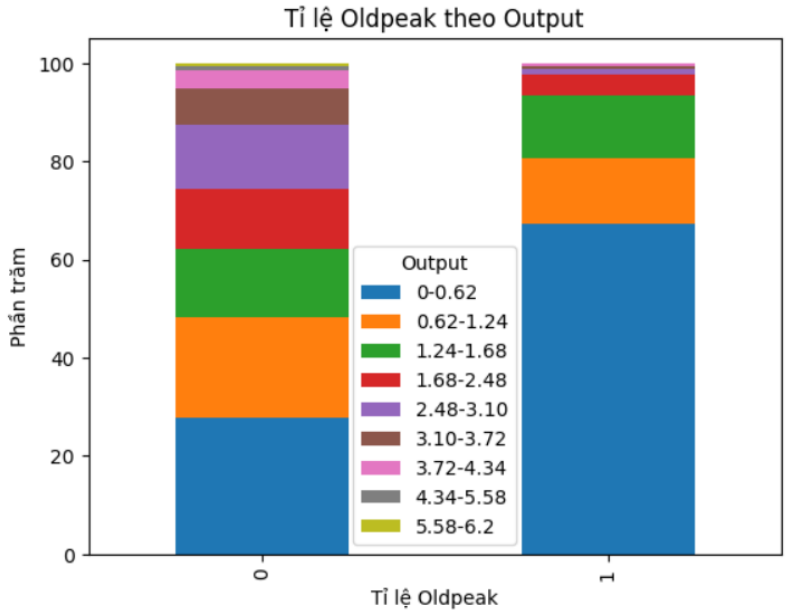
- Nhận xét: Biểu đồ cho thấy trong khoảng 300 trường hợp, có 67,3 % không đau thắt ngực do vận động, gắng sức, còn lại khoảng 32,7% trường hợp bị đau đo gắng sức, điều này có nghĩa là trong dataset trên, tỉ lệ người có khả năng mắc bệnh hở van tim thấp, đau nếu hết trong thời gian ngắn không điển hình cho bệnh lý tim mạch. Với số lượng là 209 người không đau thắt do gắng sức, 45 người đau. Nhóm tuổi đau và không đau phân bố rộng khắp trên các độ tuổi, không đặc biệt phân bố ở bất kì nhóm tuổi nào.

- Oldpeak: Độ giảm đỉnh ST do tập thể dục



Biểu đồ thống kê độ giảm điểm sóng

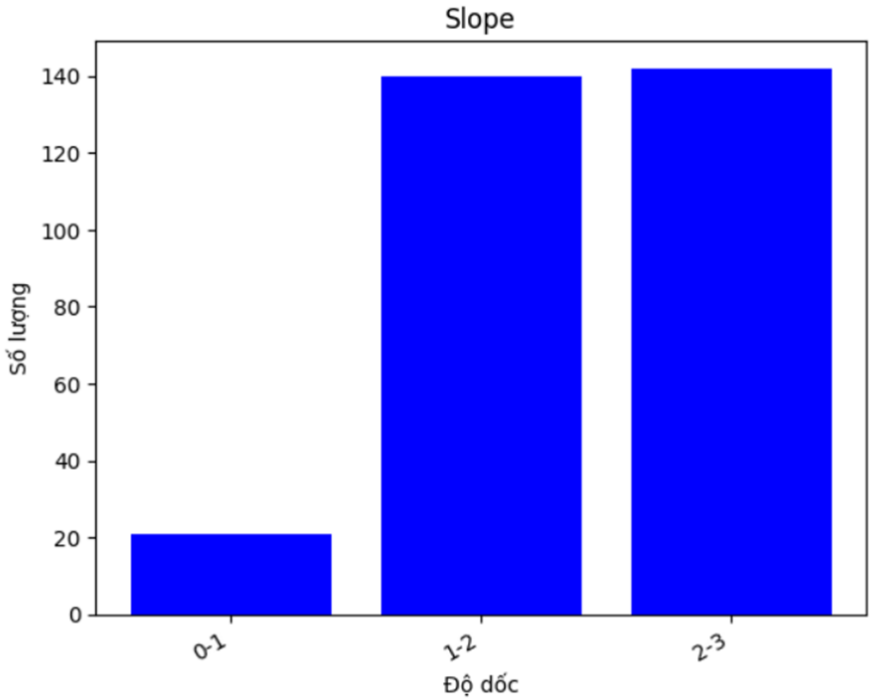
- Nhận xét: Theo dữ liệu phân tích ở trên, tình trạng thay đổi sóng của đỉnh ST do tập thể dục, có thể liên quan đến các sự kiện như thiếu máu tim hoặc tổn thương tim tập trung số lượng lớn ở những người có điểm sóng thấp từ 0-0.62. Và có dấu hiệu giảm dần đến những nơi có điểm sóng cao hơn.



Biểu đồ thống kê tỉ lệ Oldpeak theo output

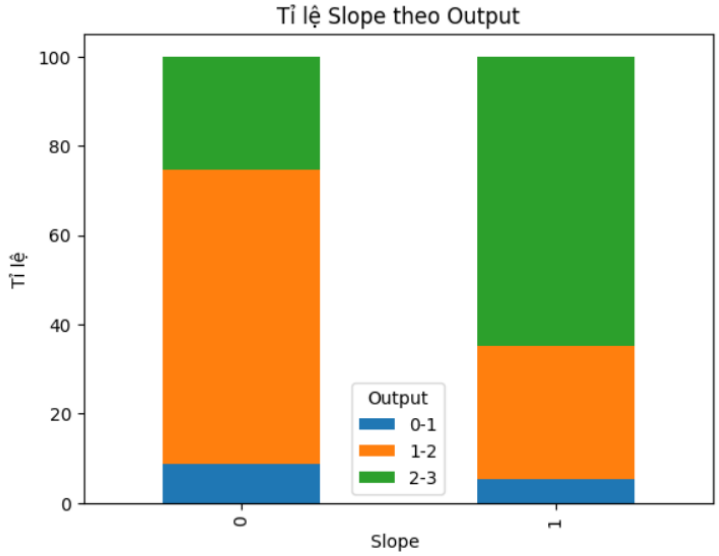
- Nhận xét: Biểu đồ tiến hành đánh giá tỉ lệ Oldpeak theo Output. Với 0 là người có tỉ lệ mắc bệnh tim thấp, biểu đồ cho thấy người có điểm sóng trong khoảng từ 0-0.62 và từ 0.62-1.24 là những người có tỉ lên an toàn cao hơn so với các điểm sóng còn lại. Tuy nhiên, điểm sóng từ 0-0.62 cũng là nhóm người có tỉ lệ mắc bệnh tim cao nhất so với các điểm sóng khác.

- Slope: Độ dốc thường dùng để đánh giá tình trạng tim mạch



Biểu đồ đánh giá tình trạng tim mạch

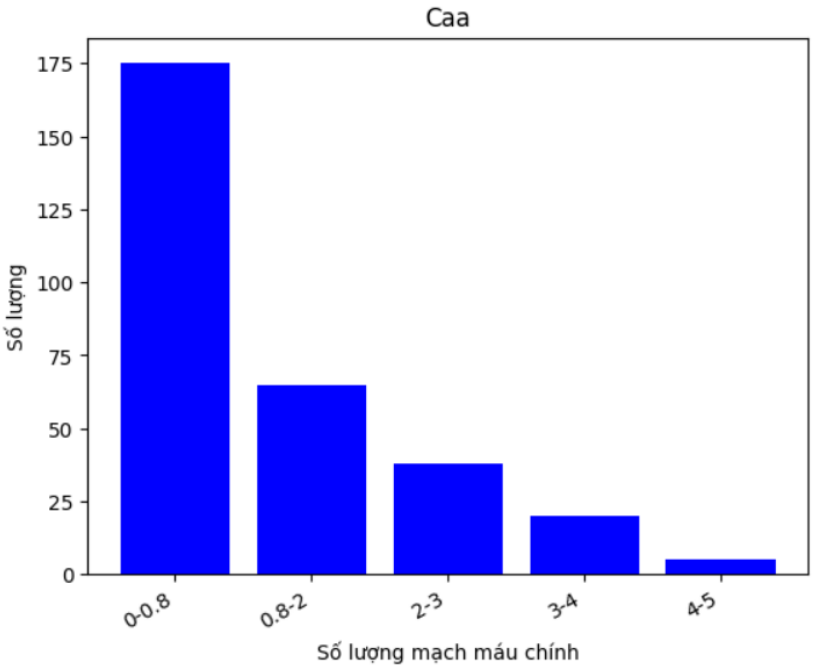
- Nhận xét: Theo dữ liệu thống kê trong biểu đồ trên, độ dốc trong khoảng từ 1-2 và từ 2-3 có số lượng cao và xấp xỉ nhau. Độ dốc trong khoảng từ 0-1 có số lượng thấp hơn hẳn so với 2 khoảng dốc trên.



Biểu đồ thống kê tỉ lệ Slope theo Output

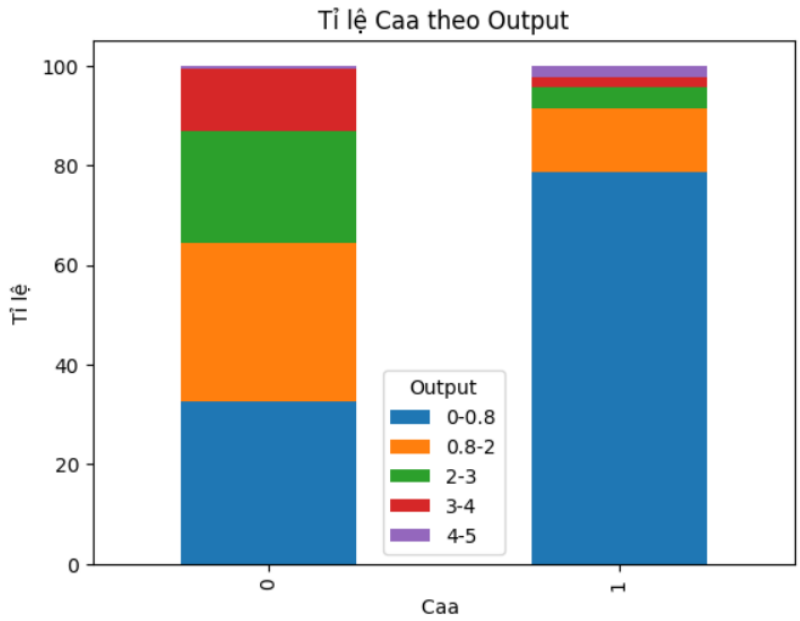
- Nhận xét: Tiến hành đánh giá tỉ lệ Slope theo Output, đối với những người có nguy cơ mắc bệnh tim thấp, có độ dốc nằm trong khoảng từ 1-2 chiếm tỉ lệ cao nhất, tiếp theo là trong khoảng 2-3 và 0-1. Đối với những người có nguy cơ mắc bệnh tim cao, độ dốc tập trung nhiều ở khoảng từ 2-3, giảm dần đến khoảng 1-2 và thấp nhất là 0-1.

- Caa: Tình trạng của mạch máu cung cấp cho tim.



Biểu đồ thống kê số lượng mạch máu chính cung cấp cho tim

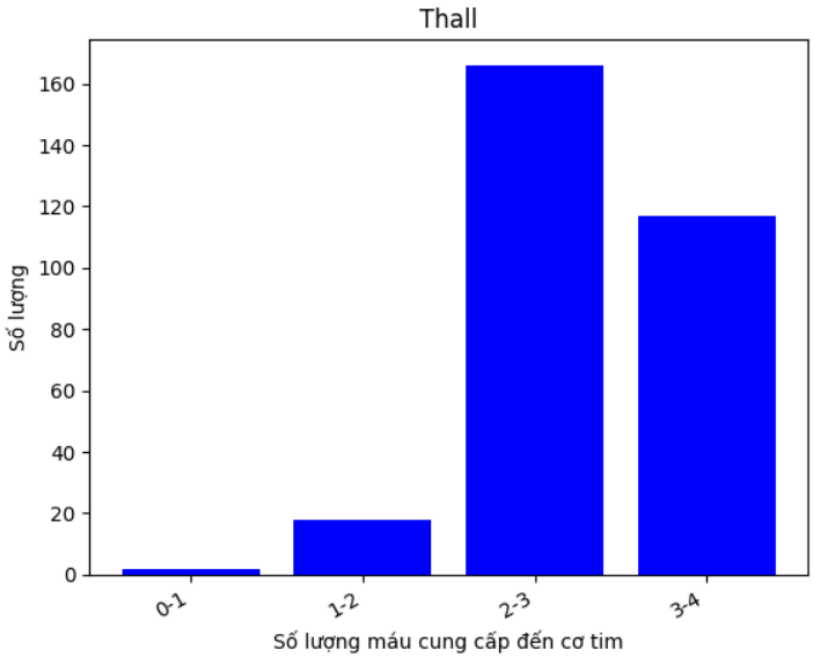
- Nhận xét: Số lượng mạch máu chính (major vessels) được đếm để đánh giá khả năng cung cấp máu đến cơ tim. Số lượng mạch máu chính tập trung cao trong khoảng từ 0-0.8. Và có xu hướng giảm dần từ 0.8-2, 2-3, 3-4 và 4-5.



Biểu đồ thống kê Caa theo Output

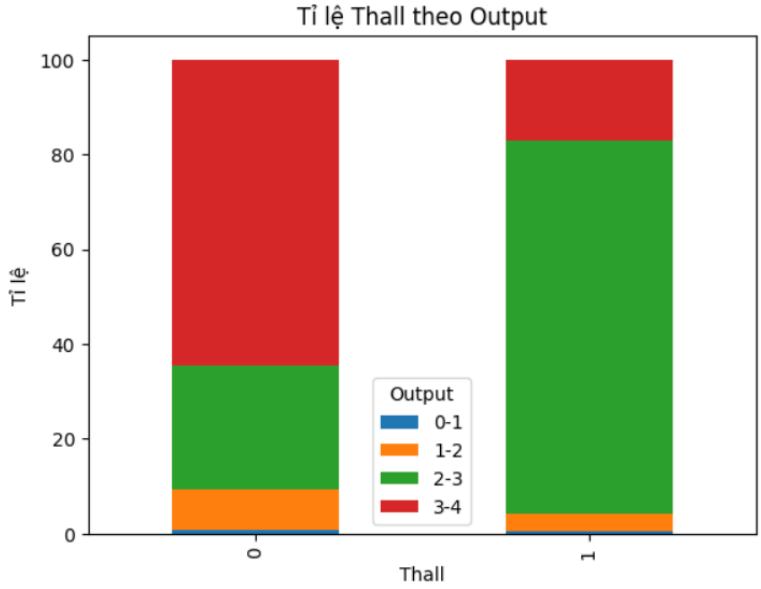
- Nhận xét: Đánh giá tỉ lệ Caa so với Output, nhận thấy rằng, số lượng mạch máu chính trong khoảng từ 0-0.8 và 0.8-2 là hai khoảng có tỉ lệ an toàn cao và có nguy cơ mắc bệnh tim thấp. Tỉ lệ an toàn chiếm tỉ lệ nhỏ hơn cho hoảng từ 2-3 và giảm dần đến các khoảng còn lại. Đối với những người có nguy cơ mắc bệnh tim cao, số mạch máu chính chiếm tỉ lệ cao nhất trong khoảng từ 0-0.8 và chiếm tỉ lệ thấp hơn đới với các khoảng còn lại.

- Thall: Lượng cung cấp máu đến cơ tim



Biểu đồ thống kê lượng máu cung cấp đến cơ tim

- Nhận xét: Theo số liệu trên, số lượng máu cung cấp đến cơ tim có số lượng cao nhất trong khoảng từ 2-3. Tiếp theo là khoảng 3-4 và số lượng máu trong khoảng từ 1-2 và 0-1 chiếm số lượng thấp.



Biểu đồ thống kê Thall theo Output

- Nhận xét: Tiến hành đánh giá số liệu trên theo Output, dễ dàng nhận thấy rằng, số lượng máu cung cấp đến cơ tim trong khoảng từ 3-4 chiếm tỉ lệ cao, nguy cơ mắc bệnh tim thấp. 2-3 là khoảng an toàn tiếp theo và giảm dần đến các khoảng còn lại. Đối với những người có nguy cơ mắc bệnh tim cao tập trung nhiều trong khoảng từ 2-3. Các khoảng còn lại chiếm tỉ lệ nhỏ hơn.

1.2 Các mô hình học máy áp dụng trong bài toán:

1.2.1 Decision tree classifier:

Mô hình Decision Tree Classifier là một mô hình học máy được sử dụng cho bài toán phân loại. Nó tạo ra một cây quyết định dựa trên dữ liệu đào tạo và sau đó sử dụng cây này để dự đoán lớp (hoặc nhãn) của các điểm dữ liệu mới.

Một số điểm quan trọng về Decision Tree Classifier:

- Xây dựng Cây Quyết Định:
  - Chọn thuộc tính chia: Ở mỗi nút quyết định, thuật toán chọn thuộc tính tốt nhất để chia dữ liệu. Một số phương pháp đánh giá sự tốt nhất của thuộc tính bao gồm Độ thuần nhất (Gini impurity), Entropy, hoặc độ giảm độ thuần nhất (Information Gain).
  - Chia dữ liệu: Dữ liệu được chia thành các nhóm con dựa trên giá trị của thuộc tính chia.
  - Lặp lại cho mỗi nhóm con: Quá trình trên được lặp lại đối với mỗi nhóm con đến khi một điều kiện dừng được đáp ứng.
- Dự đoán:
  - Khi cây quyết định đã được xây dựng, nó có thể được sử dụng để dự đoán lớp của các điểm dữ liệu mới bằng cách di chuyển từ nút gốc đến nút lá theo quy tắc quyết định đã xác định trong cây.
- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Dễ hiểu và giải thích.

- Có khả năng xử lý cả dữ liệu số và phân loại.
- Không yêu cầu chuẩn hóa dữ liệu.
- Thường cho kết quả tốt trên dữ liệu có cấu trúc tốt.

- Nhược điểm:
  - Có thể quá mức phù hợp với dữ liệu huấn luyện, dẫn đến hiện tượng quá mức (overfitting).
  - Nhạy cảm với nhiễu và thay đổi nhỏ trong dữ liệu.
  - Không thể nắm bắt được mối quan hệ phức tạp trong dữ liệu như một số mô hình phức tạp hơn.

1.2.2 Random forest classifier:

Random Forest Classifier là một mô hình học máy mạnh mẽ và phổ biến, được xây dựng dựa trên ý tưởng của nhiều cây quyết định (Decision Trees) hoạt động cùng một lúc. Nó kết hợp sức mạnh của nhiều cây quyết định để tạo ra một mô hình mạnh mẽ hơn và có khả năng chống lại hiện tượng quá mức (overfitting).

Một số đặc điểm quan trọng của Random Forest Classifier:

- Xây Dựng Random Forest:
  - Chọn ngẫu nhiên dữ liệu: Một phần của dữ liệu đào tạo được chọn ngẫu nhiên để xây dựng mỗi cây quyết định. Điều này giúp giảm khả năng overfitting và làm cho các cây quyết định độc lập.
  - Xây dựng nhiều cây quyết định: Số lượng cây quyết định (n_estimators) là một siêu tham số có thể được chọn. Mỗi cây được xây dựng độc lập nhau từ dữ liệu con được chọn ngẫu nhiên.
  - Voting hoặc averaging: Trong pha dự đoán, kết quả từ tất cả các cây quyết định được kết hợp thông qua voting (đối với bài toán phân loại) hoặc averaging (đối với bài toán dự đoán) để đưa ra dự đoán cuối cùng.
- Ưu điểm và Nhược điểm:
- Ưu điểm:

- Có khả năng xử lý các tập dữ liệu lớn với nhiều chiều dữ liệu.
- Giảm nguy cơ overfitting thông qua việc sử dụng nhiều cây quyết định độc lập.
- Độ chính xác cao và ổn định.
- Có khả năng ứng phó tốt với dữ liệu có nhiễu.

- Nhược điểm:
  - Cần nhiều tài nguyên tính toán để xây dựng và dự đoán.
  - Không như cây quyết định đơn lẻ, Random Forest không thể được biểu diễn một cách rõ ràng và giải thích được.

1.2.3 Feed Forward Neural Network:

Feedforward Neural Network (FNN) là một loại mạng nơ-ron nhân tạo cơ bản, được cấu trúc theo kiểu tiếp tuyến, tức là dữ liệu chỉ di chuyển một chiều từ lớp đầu vào đến lớp đầu ra mà không có chu kỳ phản hồi. Đây là kiểu mô hình học máy phổ biến và là cơ sở cho nhiều kiến trúc mạng nơ-ron phức tạp hơn.

- Cấu trúc của FNN:
  - Lớp Đầu vào (Input Layer): Nhận dữ liệu đầu vào và truyền chúng tới lớp ẩn đầu tiên. Số lượng nơ-ron trong lớp này phụ thuộc vào số chiều của dữ liệu đầu vào.
  - Lớp Ẩn (Hidden Layer): Có thể có một hoặc nhiều lớp ẩn, mỗi lớp chứa một số nơ-ron và có các trọng số và hàm kích hoạt riêng biệt. Lớp ẩn giúp mô hình học được các biểu diễn phức tạp của dữ liệu.
  - Lớp Đầu ra (Output Layer): Cho kết quả dự đoán
- Hàm Kích hoạt (Activation Function):
  - Mỗi nơ-ron trong lớp ẩn và lớp đầu ra thường áp dụng một hàm kích hoạt để đưa ra đầu ra phi tuyến tính. Các hàm kích hoạt phổ biến bao gồm:
  - Sigmoid: Được sử dụng trước đây, nhưng ít được ưa chuộng do vấn đề biến mất đạo hàm (vanishing gradient problem).

- Hyperbolic Tangent (tanh): Tương tự như sigmoid, nhưng có giá trị đầu ra trong khoảng [-1, 1].
- Rectified Linear Unit (ReLU): Phổ biến hiện nay, đặc biệt trong các lớp ẩn, vì nó giúp giảm vấn đề biến mất đạo hàm và thường cho hiệu suất tốt hơn.

- Huấn luyện FNN:

Quá trình huấn luyện FNN thường bao gồm hai giai đoạn chính: feedforward và backpropagation.

- Feedforward (lan truyền tiến): Dữ liệu được truyền qua mạng từ lớp đầu vào đến lớp đầu ra theo hướng một chiều. Các trọng số được kích thích bởi dữ liệu đào tạo.
- Backpropagation (lan truyền ngược): Sai số giữa đầu ra dự đoán và giá trị thực tế được tính toán, và sau đó được lan truyền ngược lại từ lớp đầu ra đến lớp đầu vào để điều chỉnh trọng số. Các thuật toán tối ưu hóa như Gradient Descent được sử dụng để cập nhật trọng số.

- Ưu điểm và Nhược điểm:
  - Ưu điểm:
    - Có khả năng học các biểu diễn phức tạp của dữ liệu.
    - Phù hợp cho nhiều loại bài toán, bao gồm phân loại và dự đoán.
    - Hiệu suất cao khi có đủ dữ liệu đào tạo.
  - Nhược điểm:
    - Yêu cầu lượng dữ liệu đào tạo lớn để tránh overfitting.
    - Cần nhiều tài nguyên tính toán, đặc biệt là với các mô hình lớn.
    - Có thể bị overfitting nếu không được kiểm soát (sử dụng kỹ thuật regularization, dropout, etc.).

1.2.4 Reccurent Neural Network:

Recurrent Neural Network (Mạng Nơ-ron Tái phát) là một dạng kiến trúc mạng nơ-ron nhân tạo thiết kế để xử lý dữ liệu dạng chuỗi hoặc dữ liệu có mối quan hệ thời

gian. Một đặc điểm quan trọng của RNN là khả năng lưu trữ thông tin từ quá khứ và sử dụng nó để đưa ra dự đoán trong tương lai. Mỗi nút trong mạng có khả năng gửi đầu ra của mình như là đầu vào cho chính nó ở bước tiếp theo, tạo ra một vòng lặp trong mô hình.

- Cấu trúc của một RNN:
    - Lớp Đầu vào (Input Layer): Nhận dữ liệu đầu vào từ mỗi bước thời gian.
    - Trạng thái ẩn (Hidden State): Duy trì trạng thái ẩn, chứa thông tin từ bước thời gian trước đó.
    - Lớp Đầu ra (Output Layer): Cho ra dự đoán cho bước thời gian hiện tại.
- Ưu điểm và Nhược điểm của RNN:
    - Ưu điểm:
        - Xử lý dữ liệu chuỗi và dữ liệu có mối quan hệ thời gian.
        - Linh hoạt trong việc xử lý đầu vào có độ dài thay đổi.
        - Phù hợp cho các bài toán như dự đoán chuỗi thời gian, ngôn ngữ tự nhiên, và nhiều ứng dụng khác.
    - Nhược điểm:
        - Có thể mất thông tin quá lâu trong quá trình lan truyền ngược (vanishing gradient problem).
        - Không hiệu quả với các chuỗi rất dài do khả năng mất thông tin.

1.2.5 Multilayer Perceptron:

Multilayer Perceptron (MLP) là một kiểu mô hình học máy thuộc loại mạng nơ-ron nhân tạo (Artificial Neural Network - ANN). Nó là một mô hình đa tầng, có nghĩa là nó bao gồm ít nhất hai tầng nơ-ron: một tầng đầu vào và một hoặc nhiều tầng ẩn, cùng với một tầng đầu ra.

- Kiến trúc của MLP:
    - Tầng đầu vào (Input Layer): Nhận dữ liệu đầu vào và truyền nó đến tầng ẩn tiếp theo.

- Tầng ẩn (Hidden Layer): Các tầng này chứa các nơ-ron ẩn, mỗi nơ-ron liên kết với tất cả các nơ-ron trong tầng trước và sau nó. Các tầng ẩn giúp mô hình học được các biểu diễn phức tạp của dữ liệu.
- Tầng đầu ra (Output Layer): Cho kết quả dự đoán của mô hình. Số lượng nơ-ron trong tầng này phụ thuộc vào bài toán cụ thể (ví dụ: một nơ-ron cho mỗi lớp trong bài toán phân loại).

- Huấn luyện MLP:
  - Feedforward: Dữ liệu được truyền qua mạng từ tầng đầu vào đến tầng đầu ra thông qua các tầng ẩn. Các trọng số của các liên kết được kích thích bởi dữ liệu đào tạo.
  - Backpropagation: Sai số dự đoán được tính toán, và sau đó được lan truyền ngược lại từ tầng đầu ra đến tầng đầu vào để điều chỉnh trọng số. Quá trình này được thực hiện để giảm thiểu sai số giữa dự đoán và giá trị thực tế.
  - Học và Điều chuẩn: Các kỹ thuật như dropout và regularization có thể được áp dụng để ngăn chặn overfitting trong quá trình huấn luyện.
- Ưu điểm và Nhược điểm:
- Ưu điểm:
  - Khả năng học được các biểu diễn phức tạp của dữ liệu.
  - Linh hoạt và có thể áp dụng cho nhiều loại dữ liệu khác nhau.
  - Có khả năng xử lý các vấn đề phức tạp, bao gồm cả bài toán phân loại và dự đoán.
- Nhược điểm:
  - Yêu cầu lượng dữ liệu đào tạo lớn để tránh overfitting.
  - Cần nhiều tài nguyên tính toán, đặc biệt là với các mô hình lớn.
  - Có thể khó giải thích được quá trình ra quyết định của mô hình.

1.2.6 kNN:

k-Nearest Neighbors (kNN) là một mô hình học máy đơn giản nhưng hiệu quả được sử dụng cho cả bài toán phân loại và dự đoán. Mô hình này quyết định lớp hoặc giá trị của một điểm dữ liệu mới bằng cách xem xét k điểm dữ liệu gần nhất từ tập dữ liệu đào tạo.

- Nguyên lý hoạt động:

Chọn giá trị k: K là số nguyên dương được chọn trước. Nó đại diện cho số lượng hàng xóm gần nhất sẽ được xem xét khi đưa ra dự đoán cho một điểm dữ liệu mới.

Tính khoảng cách: Khoảng cách giữa điểm dữ liệu mới và tất cả các điểm trong tập dữ liệu đào tạo được tính toán. Các phương pháp thường được sử dụng bao gồm khoảng cách Euclidean, khoảng cách Manhattan, hoặc các phương pháp khoảng cách khác.

Lựa chọn lớp (phân loại) hoặc tính giá trị trung bình (dự đoán): Với phân loại, lớp xuất hiện nhiều nhất trong các điểm láng giềng được chọn là kết quả dự đoán. Đối với dự đoán, giá trị trung bình của các giá trị trong các điểm láng giềng được sử dụng làm dự đoán.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Dễ hiểu và triển khai.
  - Không yêu cầu huấn luyện trước.
  - Hoạt động tốt đối với dữ liệu có cấu trúc đơn giản.
- Nhược điểm:
  - Nhạy cảm với nhiễu và các biến số không quan trọng.
  - Yêu cầu lượng dữ liệu đào tạo lớn.
  - Hiệu suất giảm khi số chiều dữ liệu tăng (hiệu ứng chiều cao).

1.2.7 Extra Tree classifier:

Extra Trees (Extremely Randomized Trees) Classifier là một biến thể của mô hình cây quyết định (Decision Tree) và thuộc vào danh mục các mô hình dựa trên rừng (ensemble models). Nó giống như Random Forest, nhưng có một số đặc điểm khác biệt trong quá trình xây dựng cây quyết định.

- Đặc điểm của Extra Trees Classifier:

Ngẫu nhiên hóa thêm: So với Random Forest, Extra Trees sử dụng một cách tiếp cận ngẫu nhiên hóa thêm khi chọn thuộc tính để chia nút trong cây quyết định. Thay vì tìm thuộc tính tốt nhất, Extra Trees chọn ngẫu nhiên một giá trị ngưỡng cho mỗi thuộc tính và chia dữ liệu dựa trên giá trị ngẫu nhiên đó.

Dự đoán bằng biểu quyết (Voting): Khi dự đoán, Extra Trees thường sử dụng biểu quyết của nhiều cây quyết định để đưa ra kết quả cuối cùng.

Chống overfitting: Do cách tiếp cận ngẫu nhiên hóa mạnh mẽ, Extra Trees có xu hướng ít bị overfitting hơn trong số các mô hình dựa trên cây quyết định.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Chống overfitting tốt hơn so với một số mô hình khác.
  - Hiệu suất tốt trên các tập dữ liệu lớn và có nhiều chiều.
  - Không yêu cầu nhiều siêu tham số để điều chỉnh.
- Nhược điểm:
  - Khó giải thích được so với các mô hình đơn giản hơn.
  - Cần nhiều tài nguyên tính toán để xây dựng và dự đoán.

1.2.8 XGBoost:

XGBoost (eXtreme Gradient Boosting) là một mô hình học máy thuộc lớp Gradient Boosting. Nó là một trong những mô hình ensemble mạnh mẽ và hiệu quả nhất được sử dụng rộng rãi trong các cuộc thi học máy và nhiều ứng dụng thực tế. XGBoost được thiết kế để giải quyết vấn đề overfitting và cung cấp hiệu suất tốt trên nhiều loại dữ liệu.

- Đặc điểm của XGBoost:

Gradient Boosting: XGBoost là một mô hình thuộc lớp Gradient Boosting, điều này có nghĩa là nó xây dựng một chuỗi các cây quyết định theo cách tuần tự, mỗi cây cố gắng sửa lỗi của cây trước đó.

Regularization: XGBoost áp dụng các kỹ thuật regularization để ngăn chặn overfitting. Cụ thể, nó sử dụng hàm mất mát (loss function) có phần tử penalty cho trọng số của các nút lá, giúp kiểm soát sự phức tạp của mô hình.

Tree Pruning: Cây quyết định trong XGBoost thường được tạo ra với số lượng lá lớn hơn, sau đó được cắt tỉa (pruned) dựa trên giá trị của hàm mất mát để tối ưu hóa hiệu suất.

Học nhánh và Tăng cường (Boosting): XGBoost có thể học từ dữ liệu phân loại và dự đoán, cũng như có thể giải quyết các vấn đề hồi quy.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Hiệu suất cao trên nhiều loại dữ liệu và bài toán.
  - Xử lý tốt các biến số không quan trọng và tương tác giữa chúng.
  - Regularization giúp kiểm soát overfitting.
  - Hỗ trợ parallel processing, làm tăng tốc quá trình huấn luyện.
- Nhược điểm:
  - Cần nhiều tài nguyên tính toán.
  - Dễ dàng bị overfitting nếu không điều chỉnh thích hợp các tham số.

1.2.9 Support Vector classifier:

Support Vector Classifier (SVC) hay còn được gọi là Support Vector Machine (SVM) là một mô hình học máy được sử dụng cho bài toán phân loại. SVM là một trong những mô hình phổ biến và mạnh mẽ, đặc biệt hiệu quả trong việc xử lý các bài toán phân loại tuyến tính và phi tuyến tính.

- Nguyên lý hoạt động của SVM:

Tìm đường ranh giới tối ưu (Optimal Hyperplane): SVM tìm một đường ranh giới (hyperplane) tối ưu nhất để phân chia giữa các lớp. Nếu dữ liệu không thể phân chia tuyến tính, SVM có thể sử dụng các hàm kernel để ánh xạ dữ liệu lên một không gian cao chiều hơn để tạo ra một đường ranh giới phi tuyến tính.

Support Vectors: Support Vectors là các điểm dữ liệu nằm gần ranh giới phân chia và quyết định vị trí của đường ranh giới.

Cực đại hóa ranh giới: SVM cố gắng cực đại hóa khoảng cách giữa các Support Vectors và đường ranh giới, điều này giúp tăng cường sự tổng quát hóa của mô hình.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Hiệu suất tốt trong các bài toán phân loại tuyến tính và phi tuyến tính.
  - Hiệu quả khi số chiều dữ liệu lớn.
  - Chống lại overfitting tốt.
- Nhược điểm:
  - Đòi hỏi một lượng dữ liệu đào tạo lớn.
  - Nhạy cảm với việc lựa chọn tham số, đặc biệt là tham số C (khiến cho mô hình đàn áp dữ liệu đào tạo).

1.2.10 Adaboost classifier:

Adaboost (Adaptive Boosting) Classifier là một mô hình ensemble thuộc lớp Boosting, được thiết kế để tăng cường khả năng của một mô hình yếu (weak learner) bằng cách tập trung vào các điểm dữ liệu bị phân loại sai. Adaboost giúp tạo ra một mô hình mạnh mẽ từ việc kết hợp nhiều mô hình yếu.

- Nguyên lý hoạt động của Adaboost:

Chọn mô hình yếu: Một mô hình yếu (ví dụ: cây quyết định) được chọn ban đầu để tạo thành mô hình cơ bản.

Tạo trọng số: Mỗi điểm dữ liệu được gán một trọng số, và các điểm dữ liệu bị phân loại sai nhận được trọng số lớn hơn.

Xây dựng mô hình mới: Một mô hình yếu khác được xây dựng trên dữ liệu với các trọng số mới được cập nhật.

Cập nhật trọng số: Trọng số của các điểm dữ liệu bị phân loại sai được cập nhật để tập trung vào các điểm khó phân loại hơn.

Kết hợp các mô hình: Mô hình mới được kết hợp với mô hình trước đó, và quá trình lặp lại từ bước 2 cho đến khi đạt đến số lượng mô hình yếu được đặt trước (n_estimators) hoặc đến khi độ chính xác đủ cao.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Hiệu suất tốt trên nhiều loại dữ liệu và bài toán.
  - Tích hợp tốt với các mô hình yếu khác nhau.
  - Khả năng làm việc với dữ liệu không cân bằng.
- Nhược điểm:
  - Nhạy cảm với nhiễu và outliers trong dữ liệu đào tạo.
  - Cần phải chọn số lượng mô hình yếu cẩn thận để tránh overfitting.

1.2.11 Gradient boosting machine:

Gradient Boosting Machine (GBM) là một mô hình ensemble thuộc lớp Boosting, giống như Adaboost, nhưng nó sử dụng một chiến lược gradient để xây dựng các mô hình yếu. GBM tập trung vào việc giảm độ lỗi của mô hình dự đoán trước đó bằng cách tạo ra các mô hình mới và cập nhật trọng số của chúng dựa trên độ lỗi.

- Nguyên lý hoạt động của GBM:

Chọn mô hình yếu ban đầu: Một mô hình yếu (thường là cây quyết định) được chọn làm mô hình cơ bản.

Tính đạo hàm (Gradient) của hàm mất mát: Đạo hàm của hàm mất mát (loss function) được tính dựa trên giá trị dự đoán và giá trị thực tế.

Xây dựng mô hình mới để giảm độ lỗi: Một mô hình yếu mới được xây dựng để giảm độ lỗi dự đoán của mô hình hiện tại. Mô hình mới được xây dựng bằng cách tối ưu hóa đạo hàm của hàm mất mát.

Cập nhật trọng số của mô hình mới: Trọng số của mô hình mới được cập nhật để có sự đóng góp đúng đắn vào mô hình tổng cộng.

Lặp lại quá trình: Các bước 2-4 được lặp lại cho đến khi đạt được số lượng mô hình yếu mong muốn hoặc đến khi độ lỗi không giảm đáng kể.

- Ưu điểm và Nhược điểm:

- Ưu điểm:
  - Hiệu suất tốt trên nhiều loại dữ liệu và bài toán.
  - Có khả năng xử lý dữ liệu có cấu trúc phức tạp và tương tác giữa các biến số.
  - Regularization tự nhiên thông qua việc sử dụng learning rate nhỏ.
- Nhược điểm:
  - Cần nhiều tài nguyên tính toán và thời gian đào tạo so với một số mô hình khác.
  - Nhạy cảm với outliers trong dữ liệu.

1.3 Các phương pháp tránh Overfitting:

1.3.1 Drop out:

- Ý tưởng chính:

Dropout là một kỹ thuật chính thức để ngăn chặn việc quá mức tinh chỉnh mô hình vào dữ liệu huấn luyện và giúp nó tổng quát hóa tốt hơn với dữ liệu mới.

Trong quá trình huấn luyện, một số lượng ngẫu nhiên các nơ-ron trong mạng nơ-ron sẽ bị "tắt" (ngưng hoạt động) ở mỗi lượt đi qua dữ liệu.

- Cách thức hoạt động:

Khi một nơ-ron bị tắt (dropout), nó không tham gia vào quá trình tính toán trong lượt đi qua dữ liệu đó. Cụ thể, giá trị đầu ra của nơ-ron này sẽ là 0.

Việc tắt ngẫu nhiên các nơ-ron mô phỏng việc mô hình không phụ thuộc quá mức vào một số đặc trưng cụ thể trong quá trình huấn luyện.

- Tỉ lệ dropout:

Tỉ lệ dropout là tỷ lệ các nơ-ron bị tắt trong mỗi lượt đi qua dữ liệu. Thông thường, giá trị này được chọn là một giá trị nhỏ như 0.2 đến 0.5.

Tỉ lệ này là một siêu tham số có thể được điều chỉnh để tối ưu hóa hiệu suất của mô hình.

- Ứng dụng trong các lớp khác nhau:

Dropout thường được áp dụng trong các lớp fully connected (hoặc dense) và một số trường hợp cũng được áp dụng trong các lớp convolutional.

- Lợi ích:

Dropout giúp mô hình tránh overfitting bằng cách làm cho các nơ-ron không thể dựa vào sự hiện diện của các nơ-ron khác trong quá trình dự đoán.

Nó giúp tạo ra một mô hình tổng quát hóa tốt hơn cho dữ liệu mới mà nó chưa từng thấy.

1.3.2 Ealry Stopping:

Kỹ thuật dừng sớm rất quan trọng, khi chúng ta dừng ở giai đoạn đào tạo trước cả khi mô hình học những dữ liệu gây nhiễu sẽ cho chúng ta được kết quả chính xác nhất.

- Các bước thực hiện:

1. Chia dữ liệu:

Chia dữ liệu thành tập huấn luyện và tập validation. Tập huấn luyện được sử dụng để cập nhật trọng số của mô hình, trong khi tập validation được sử dụng để đánh giá hiệu suất của mô hình trên dữ liệu mà nó chưa từng thấy.

2. Theo dõi hiệu suất:

Đo lường hiệu suất của mô hình trên tập validation sau mỗi epoch hoặc một số lượt huấn luyện cố định.

3. So sánh hiệu suất:

So sánh hiệu suất trên tập validation giữa các lượt huấn luyện. Nếu hiệu suất không còn cải thiện hoặc giảm đi, đặc biệt là sau một số epoch đã được chọn trước (thường gọi là patience), quá trình huấn luyện sẽ dừng lại.

4. Lưu mô hình tốt nhất:

Trong suốt quá trình huấn luyện, lưu lại trạng thái của mô hình có hiệu suất tốt nhất trên tập validation. Sau khi quá trình huấn luyện dừng lại, sử dụng mô hình này để dự đoán trên dữ liệu mới.

5. Cài đặt Patience:

Patience là số lượng epoch mà hiệu suất trên tập validation không cải thiện trước khi dừng lại. Patience giúp mô hình có thể "chấp nhận" một số sự biến động nhỏ trong hiệu suất mà không dừng quá sớm.

1.3.3 L1 regularization:

L1 regularization, còn được gọi là "Lasso regularization," là một phương pháp chính để kiểm soát quá mức phức tạp của mô hình trong quá trình huấn luyện. Nó được thêm vào hàm mất mát để đảm bảo rằng trọng số của các đặc trưng không trở nên quá lớn. Điều này giúp tránh tình trạng overfitting và tạo ra một mô hình có khả năng tổng quát hóa tốt hơn.

Cơ chế hoạt động của L1 regularization dựa trên việc thêm một thành phần vào hàm mất mát của mô hình, được tính bằng tổng giá trị tuyệt đối của trọng số của các đặc trưng. Đối với một mô hình hồi quy tuyến tính đơn giản, hàm mất mát với L1 regularization có thể được biểu diễn như sau:

$$J(w) = L(\hat{y}, y) + \lambda \sum_{i=1}^{n} |w_i|$$

Khi hàm mất mát được tối ưu hóa, mô hình sẽ cố gắng giảm giá trị tuyệt đối của các trọng số ($|w_i|$) để giảm ảnh hưởng của L1 regularization. Điều này dẫn đến việc chỉ một số lượng nhỏ các trọng số là khác 0, và do đó, mô hình trở nên thưa (sparse).

Ưu điểm của L1 regularization bao gồm khả năng chọn lọc đặc trưng (feature selection), giúp giảm số lượng đặc trưng không quan trọng. Tuy nhiên, đối với một số bài toán, L1 regularization có thể làm cho quá trình tối ưu hóa hàm mất mát trở nên khó khăn và yêu cầu sự cân nhắc khi sử dụng.

1.4 Giải pháp cải thiện độ chính xác:

1.4.1 Tăng độ phức tạp của mô hình:

Tăng độ phức tạp của mô hình là một phương pháp để cải thiện khả năng học và dự đoán của mô hình máy học máy. Một số giải pháp để tăng độ phức tạp của mô hình:

- Thêm lớp và nơ-ron trong mạng nơ-ron:
  - Thêm các lớp và nơ-ron vào mô hình Neural Network để làm tăng độ phức tạp.
  - Điều này có thể giúp mô hình học được các mối quan hệ phức tạp hơn giữa các đặc trưng.
- Sử dụng kiến trúc mô hình nâng cao:
  - Sử dụng kiến trúc mô hình nâng cao như Convolutional Neural Networks (CNN) cho dữ liệu hình ảnh, Recurrent Neural Networks (RNN) cho dữ liệu chuỗi thời gian, hoặc Transformer cho dữ liệu chuỗi không gian và thời gian.
- Tăng kích thước của mô hình:
  - Tăng số lượng lớp và nơ-ron trong mô hình. Tuy nhiên, việc tăng kích thước của mô hình cũng tăng khả năng overfitting, vì nó có thể học quá mức từ dữ liệu huấn luyện.
- Tăng Cường Regularization:
  - Sử dụng các kỹ thuật như L1 hoặc L2 regularization để kiểm soát quá mức học của mô hình. Regularization giúp giảm overfitting khi mô hình quá phức tạp.

1.4.2 Sử dụng kỹ thuật Ensemble:

Kỹ thuật Ensemble là một phương pháp trong machine learning, mà nó kết hợp các dự đoán từ nhiều mô hình khác nhau để tạo ra một dự đoán cuối cùng. Mục tiêu của Ensemble là cải thiện hiệu suất và ổn định của mô hình so với việc sử dụng một mô hình đơn lẻ.

- Voting:

Kỹ thuật voting trong machine learning là một phương pháp kết hợp dự đoán từ nhiều mô hình học máy khác nhau để tạo ra một dự đoán cuối cùng có độ chính xác cao hơn so với việc sử dụng một mô hình đơn lẻ. Có hai loại chính của kỹ thuật voting: Hard Voting và Soft Voting.

- Hard Voting:

Trong Hard Voting, nhiều mô hình đưa ra dự đoán và kết quả cuối cùng được quyết định bằng cách sử dụng đa số phiếu.

- Soft Voting:

Trong Soft Voting, mỗi mô hình đưa ra một dự đoán và trọng số được áp dụng cho từng mô hình. Kết quả cuối cùng được tính bằng cách lấy trung bình có trọng số của các dự đoán. Thường thì, các mô hình có độ tin cậy cao hơn được gán trọng số lớn hơn trong quyết định cuối cùng.

- Ưu điểm của kỹ thuật voting:

Tăng độ chính xác: Khi kết hợp nhiều mô hình, kỹ thuật voting thường cung cấp độ chính xác cao hơn so với việc sử dụng một mô hình đơn lẻ.

Ổn định và giảm overfitting: Kỹ thuật này có thể giúp giảm nguy cơ overfitting do sự đa dạng trong các mô hình.

- Nhược điểm:

Tăng độ phức tạp: Việc triển khai và quản lý nhiều mô hình có thể làm tăng độ phức tạp của hệ thống.

Yêu cầu nhiều tài nguyên: Sử dụng nhiều mô hình đồng thời có thể đòi hỏi nhiều tài nguyên tính toán.

- Bagging:

Bagging là một phương pháp kết hợp (ensemble) trong machine learning, sử dụng để cải thiện độ chính xác của mô hình dự đoán. "Bagging" là viết tắt của "Bootstrap Aggregating". Phương pháp này thường được áp dụng cho các thuật toán học máy có tính chất nhiễu và dễ bị overfitting.

- Qui trình thực hiện:

1. Bootstrap Sampling (Lấy mẫu theo phương pháp bootstrap):

Dữ liệu huấn luyện được lấy mẫu ngẫu nhiên nhiều lần với lặp lại, tạo ra các tập con-dữ liệu (bootstrap samples). Mỗi tập con này có kích thước bằng với tập dữ liệu huấn luyện ban đầu, nhưng có thể chứa các mẫu trùng lặp và một số mẫu không được chọn.

2. Huấn luyện mô hình:

Trên mỗi tập con-dữ liệu bootstrap, một mô hình base (cơ sở) được huấn luyện. Các mô hình này có thể là những phiên bản nhỏ khác nhau do sự đa dạng của dữ liệu bootstrap.

3. Kết hợp dự đoán:

Dự đoán từ mỗi mô hình base được kết hợp để tạo ra dự đoán cuối cùng. Trong trường hợp phân loại, thì kết quả có thể được quyết định bằng cách sử dụng đa số phiếu (hard voting) hoặc trung bình có trọng số của các xác suất dự đoán (soft voting).

- Ưu điểm:

Giảm Variance (Độ biến thiên): Bằng cách sử dụng nhiều mô hình được huấn luyện trên các tập con-dữ liệu khác nhau, Bagging giúp giảm độ biến thiên của mô hình tổng hợp.

Giảm Overfitting: Các mô hình base thường có xu hướng overfitting trên dữ liệu huấn luyện. Tuy nhiên, khi kết hợp chúng thông qua Bagging, khả năng overfitting này giảm đi.

- Nhược điểm:

Tăng Độ Phức Tạp: Bagging yêu cầu huấn luyện và dự đoán từ nhiều mô hình base, điều này có thể làm tăng độ phức tạp của quá trình.

Không Giảm Bias: Mặc dù Bagging thường giúp giảm độ biến thiên và overfitting, nhưng nó không giảm bias của mô hình base. Nếu mô hình base có độ bias cao, thì cả mô hình Bagging cũng có thể bị ảnh hưởng.

Khó Diễn Giải: Việc kết hợp nhiều mô hình có thể làm cho mô hình tổng hợp trở nên khó diễn giải. Điều này có thể gây khó khăn trong việc hiểu rõ cách mô hình đưa ra quyết định.

- Boosting:

Boosting là một kỹ thuật kết hợp (ensemble) khác trong machine learning, cũng nhằm mục đích cải thiện độ chính xác của mô hình dự đoán. Trái ngược với Bagging, Boosting tập trung vào việc tăng cường trọng số cho các mẫu dữ liệu được dự đoán sai, giúp mô hình tập trung hơn vào các trường hợp khó học.

- Các bước cơ bản trong quá trình Boosting:

Huấn luyện Mô hình Base (Base Learner):

Một mô hình base (thường là cây quyết định nhỏ hoặc weak learner) được huấn luyện trên tập dữ liệu huấn luyện.

1. Đánh giá Hiệu suất:

Dự đoán được thực hiện trên toàn bộ tập dữ liệu, và các trọng số được áp dụng cho mỗi mẫu dữ liệu. Các mẫu dữ liệu được dự đoán sai sẽ có trọng số cao hơn.

2. Cập Nhật Trọng Số:

Trọng số của các mẫu dữ liệu được cập nhật, tăng cường trọng số cho các mẫu được dự đoán sai.

3. Huấn luyện Mô hình Tiếp Theo:

Một mô hình base mới được huấn luyện, với việc xem xét các trọng số mới của mẫu dữ liệu.

4. Lặp Lại:

Các bước 2-4 được lặp lại cho đến khi đạt được số lượng mô hình cần thiết hoặc khi độ chính xác không còn tăng.

5. Kết hợp Dự đoán:

Dự đoán cuối cùng được tạo ra bằng cách kết hợp dự đoán của tất cả các mô hình base, thường dùng trọng số để xác định ảnh hưởng của mỗi mô hình.

- Ưu điểm của Boosting:

Tăng chính xác: Boosting thường dẫn đến mô hình có độ chính xác cao hơn so với mô hình base.

Xử lý hiệu quả trường hợp khó học: Boosting tập trung vào việc cải thiện dự đoán cho các trường hợp khó học, giúp làm giảm bias và tăng độ chính xác.

- Nhược điểm:

Nhạy Cảm với Nhiễu và Outliers: Boosting có thể làm tăng ảnh hưởng của nhiễu và outliers trong dữ liệu huấn luyện. Nếu có nhiễu mạnh hoặc outliers, các mô hình base có thể tập trung vào chúng, dẫn đến overfitting.

Khả Năng Overfitting: Boosting có thể dễ bị overfitting đặc biệt là khi số lượng mô hình base là quá lớn. Việc tối ưu hóa dự đoán trên tập huấn luyện có thể dẫn đến việc tăng độ phức tạp của mô hình và không tạo ra dự đoán tổng quát tốt trên dữ liệu mới.

Yêu Cầu Nhiều Tài Nguyên: Huấn luyện một chuỗi các mô hình base và tối ưu hóa trọng số có thể đòi hỏi nhiều tài nguyên tính toán và thời gian hơn so với một số phương pháp khác.

Khả Năng Khó Điều Chỉnh Tham Số: Việc điều chỉnh tham số cho các mô hình base và thậm chí cho quá trình tối ưu hóa có thể là một thách thức. Việc chọn lựa tham số không phù hợp có thể dẫn đến overfitting hoặc underfitting.

- Stacking:

Stacking, hay còn được gọi là Stacked Generalization, là một phương pháp kết hợp (ensemble) khác trong machine learning. Khác với Bagging và Boosting, Stacking

không chỉ kết hợp dự đoán của các mô hình base, mà còn sử dụng một mô hình meta (meta-model) để học cách kết hợp các dự đoán của các mô hình base.

- Các bước chính của quá trình Stacking:

1. Chia tập dữ liệu huấn luyện:

Tập dữ liệu huấn luyện được chia thành hai phần: một phần được sử dụng để huấn luyện các mô hình base, và phần còn lại được sử dụng để huấn luyện mô hình meta.

2. Huấn luyện các Mô hình Base:

Một số mô hình base được huấn luyện trên phần của tập dữ liệu huấn luyện được dành cho chúng.

3. Tạo Dự đoán của các Mô hình Base:

Các mô hình base được sử dụng để dự đoán trên phần còn lại của tập dữ liệu huấn luyện, tạo ra các dự đoán cho mỗi mô hình.

4. Huấn luyện Mô hình Meta:

Dự đoán của các mô hình base được sử dụng làm đặc trưng đầu vào cho mô hình meta. Mô hình meta được huấn luyện trên các dự đoán này cùng với nhãn thực tế của tập dữ liệu.

5. Kết hợp Dự đoán của các Mô hình Base:

Khi có một dự đoán mới, mỗi mô hình base dự đoán và được kết hợp bởi mô hình meta để tạo ra dự đoán cuối cùng.

- Ưu điểm:

Tận dụng sự đa dạng của các mô hình: Stacking có thể tận dụng sự đa dạng của các mô hình base để tạo ra một mô hình tổng hợp mạnh mẽ.

Hiệu suất cao: Khi được cấu hình đúng, Stacking có thể mang lại độ chính xác cao hơn so với các phương pháp kết hợp khác.

- Nhược điểm:

Tăng độ phức tạp: Quá trình huấn luyện và triển khai mô hình Stacking có thể tăng độ phức tạp của hệ thống.

Yêu cầu nhiều tài nguyên: Cần nhiều tài nguyên tính toán hơn để huấn luyện và duy trì một mô hình Stacking so với các phương pháp kết hợp khác.

1.4.3 Sử dụng phương pháp Oversampling và Undersampling:

- Oversampling:

Phương pháp Oversampling là một kỹ thuật được sử dụng trong việc xử lý mất cân bằng dữ liệu, đặc biệt là trong bài toán phân loại khi có sự chênh lệch lớn giữa số lượng các mẫu thuộc các lớp khác nhau. Mục tiêu của Oversampling là tăng cường số lượng mẫu của lớp thiểu số để giảm hiện tượng chệch và cải thiện khả năng dự đoán của mô hình. Một số phương pháp Oversampling phổ biến:

  - Random Oversampling: Tạo ra các bản sao của mẫu ngẫu nhiên thuộc lớp thiểu số để làm tăng số lượng mẫu của lớp đó.
  - SMOTE (Synthetic Minority Over-sampling Technique): Tạo ra các mẫu giả mạo bằng cách kết hợp các mẫu của lớp thiểu số với các điểm dữ liệu láng giềng.
  - ADASYN (Adaptive Synthetic Sampling): Tương tự như SMOTE, nhưng ADASYN tăng cường mẫu cho các điểm dữ liệu gần biên quyết định hơn.

- Ưu điểm: Tăng cường số lượng mẫu cho lớp thiểu số có thể giúp mô hình học được mô hình quyết định phức tạp hơn và đạt độ chính xác cao hơn.
- Nhược điểm: Có thể dẫn đến overfitting, đặc biệt nếu sử dụng một cách ngẫu nhiên quá mức và không kiểm soát được quá trình tạo mẫu.

- Undersampling:

Undersampling là một phương pháp khác trong việc xử lý mất cân bằng dữ liệu, đặc biệt là trong các bài toán phân loại khi có sự chênh lệch lớn giữa số lượng các mẫu thuộc các lớp khác nhau. Ngược lại với Oversampling, Undersampling giảm số lượng

mẫu của lớp đa số để làm cho dữ liệu cân bằng hơn. Một số phương pháp Undersampling bao gồm:

- Random Undersampling: Ngẫu nhiên loại bỏ một số mẫu từ lớp đa số để làm giảm số lượng mẫu của lớp đó.
- Cluster Centroids: Thay thế các cluster của lớp đa số bằng các centroid của chúng để giảm số lượng mẫu.
- Tomek Links: Loại bỏ các cặp gần nhau thuộc hai lớp khác nhau (Tomek Links) để giảm số lượng mẫu.

- Ưu điểm: Giảm số lượng mẫu của lớp đa số giúp giảm overfitting và giữ cho mô hình đơn giản hơn.
- Nhược điểm: Mất mát thông tin, đặc biệt là nếu loại bỏ các mẫu quan trọng.

CHƯƠNG 2 – GIẢI QUYẾT BÀI TOÁN

2.1 Ứng dụng các mô hình học máy cơ bản để giải quyết bài toán:

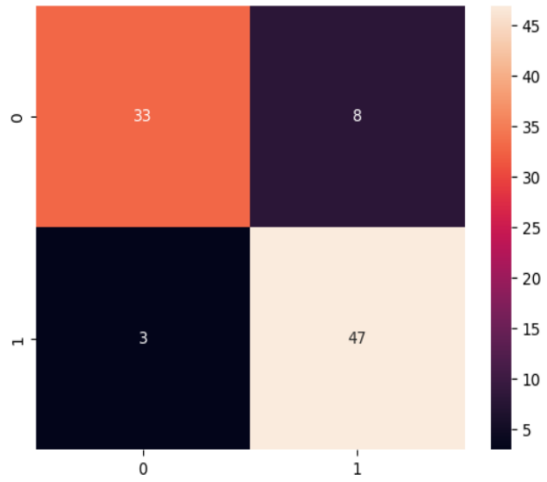
2.1.1 Các mô hình áp dụng:

- Random Forest Classifier:

```
##Random Forest Classifier
CM=confusion_matrix(y_test,y_pred_rfe)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_rfe)
acc= accuracy_score(y_test, y_pred_rfe)
roc=roc_auc_score(y_test, y_pred_rfe)
prec = precision_score(y_test, y_pred_rfe)
rec = recall_score(y_test, y_pred_rfe)
f1 = f1_score(y_test, y_pred_rfe)

mathew = matthews_corrcoef(y_test, y_pred_rfe)
model_results =pd.DataFrame([['Random Forest Classifier',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình Random Forest Classifier

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	Random Forest Classifier	0.879121	0.854545	0.94	0.804878	0.895238	0.872439	4.356925	0.757931



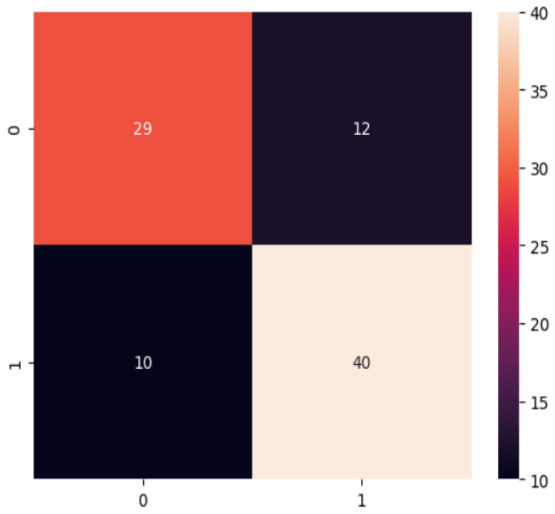
Kết quả đánh giá mô hình Random Forest Classifier

- Decision Tree Classifier:

```
##decision Tree Classifier
CM=confusion_matrix(y_test,y_pred_decc)
sns.heatmap(CM, annot=True)
#TN (True Negative)
TN = CM[0][0]
#FN (False Negative)
FN = CM[1][0]
#TP (True Positive
TP = CM[1][1]
#FP (False Positive)
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_decc)
acc= accuracy_score(y_test, y_pred_decc)
roc=roc_auc_score(y_test, y_pred_decc)
prec = precision_score(y_test, y_pred_decc)
rec = recall_score(y_test, y_pred_decc)
f1 = f1_score(y_test, y_pred_decc)

mathew = matthews_corrcoef(y_test, y_pred_decc)
model_results =pd.DataFrame([['decision Tree Classifier',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình Decision Tree Classifier

	model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	decision Tree Classifier	0.758242	0.769231	0.8	0.707317	0.784314	0.753659	8.71385	0.510061



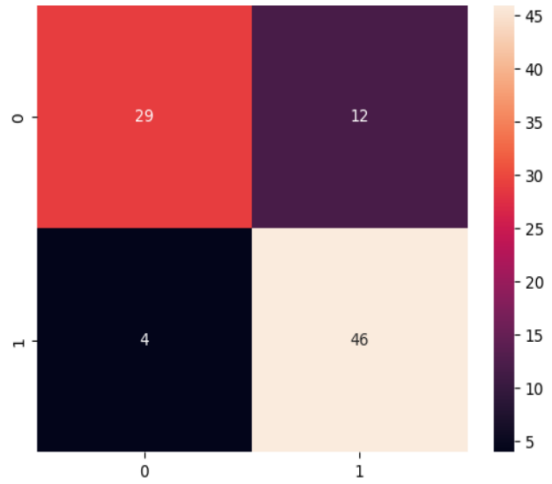
Kết quả đánh giá mô hình Decision Tree Classifier

- kNN:

```
##K nearest neighbour (n=9)
CM=confusion_matrix(y_test,y_pred_knn)
sns.heatmap(CM, annot=True)
#TN (True Negative)
TN = CM[0][0]
#FN (False Negative)
FN = CM[1][0]
#TP (True Positive
TP = CM[1][1]
#FP (False Positive)
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_knn)
acc= accuracy_score(y_test, y_pred_knn)
roc=roc_auc_score(y_test, y_pred_knn)
prec = precision_score(y_test, y_pred_knn)
rec = recall_score(y_test, y_pred_knn)
f1 = f1_score(y_test, y_pred_knn)

mathew = matthews_corrcoef(y_test, y_pred_knn)
model_results =pd.DataFrame([['K nearest neighbour (n=9)',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình kNN

	model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	K nearest neighbour (n=9)	0.824176	0.793103	0.92	0.707317	0.851852	0.813659	6.337346	0.649222



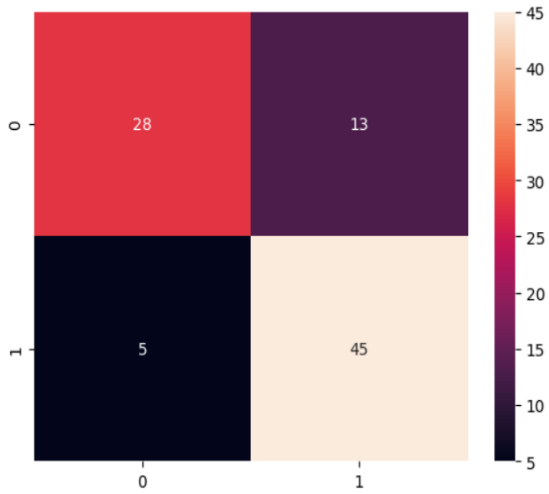
Kết quả đánh giá mô hình kNN

- Extra Tree Classifier:

```
##Extra Tree Classifier (n_estimators=500)
CM=confusion_matrix(y_test,y_pred_et500)
sns.heatmap(CM, annot=True)
#TN (True Negative)
TN = CM[0][0]
#FN (False Negative)
FN = CM[1][0]
#TP (True Positive
TP = CM[1][1]
#FP (False Positive)
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_et500)
acc= accuracy_score(y_test, y_pred_et500)
roc=roc_auc_score(y_test, y_pred_et500)
prec = precision_score(y_test, y_pred_et500)
rec = recall_score(y_test, y_pred_et500)
f1 = f1_score(y_test, y_pred_et500)

mathew = matthews_corrcoef(y_test, y_pred_et500)
model_results =pd.DataFrame([['Extra Tree Classifier (n_estimators=500)',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình Extra Tree Classifier

	model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	Extra Tree Classifier (n_estimators=500)	0.802198	0.775862	0.9	0.682927	0.833333	0.791463	7.129514	0.603282



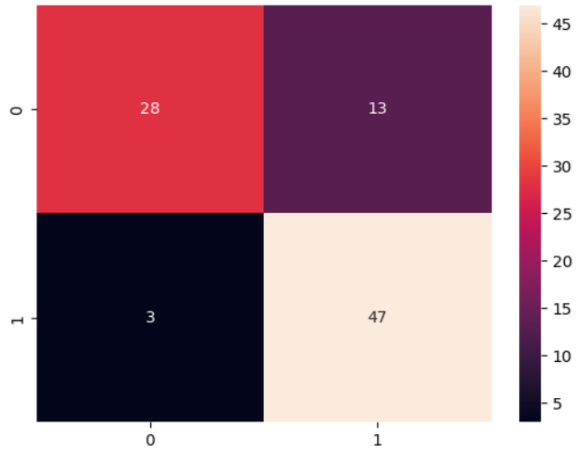
Kết quả đánh giá hiệu suất mô hình Extra Tree Classifier

- XGBoost:

```
##XGBoost (n_estimators=100)
CM=confusion_matrix(y_test,y_pred_xgb)
sns.heatmap(CM, annot=True)
#TN (True Negative)
TN = CM[0][0]
#FN (False Negative)
FN = CM[1][0]
#TP (True Positive
TP = CM[1][1]
#FP (False Positive)
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_xgb)
acc= accuracy_score(y_test, y_pred_xgb)
roc=roc_auc_score(y_test, y_pred_xgb)
prec = precision_score(y_test, y_pred_xgb)
rec = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)

mathew = matthews_corrcoef(y_test, y_pred_xgb)
model_results =pd.DataFrame([['XGBoost (n_estimators=100)',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình XGBoost

	model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	XGBoost (n_estimators=100)	0.824176	0.783333	0.94	0.682927	0.854545	0.811463	6.337346	0.65397



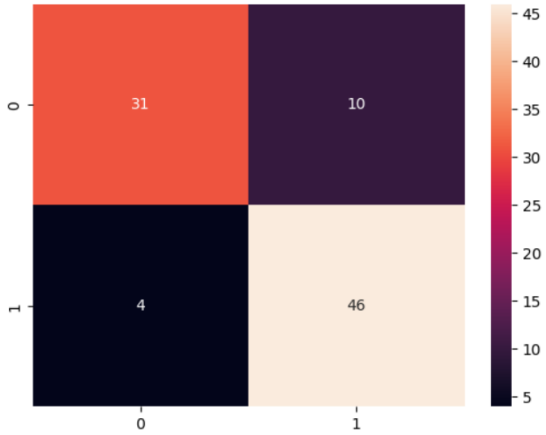
Kết quả đánh giá hiệu suất mô hình XGBoost

- Support Vector Classifier:

```
##Support Vector Classifier (kernel='linear')
CM=confusion_matrix(y_test,y_pred_svc)
sns.heatmap(CM, annot=True)
#TN (True Negative)
TN = CM[0][0]
#FN (False Negative)
FN = CM[1][0]
#TP (True Positive
TP = CM[1][1]
#FP (False Positive)
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_svc)
acc= accuracy_score(y_test, y_pred_svc)
roc=roc_auc_score(y_test, y_pred_svc)
prec = precision_score(y_test, y_pred_svc)
rec = recall_score(y_test, y_pred_svc)
f1 = f1_score(y_test, y_pred_svc)

mathew = matthews_corrcoef(y_test, y_pred_svc)
model_results =pd.DataFrame([['Support Vector Classifier',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình Support Vector Classifier

	model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	Support Vector Classifier	0.846154	0.821429	0.92	0.756098	0.867925	0.838049	5.545177	0.691446



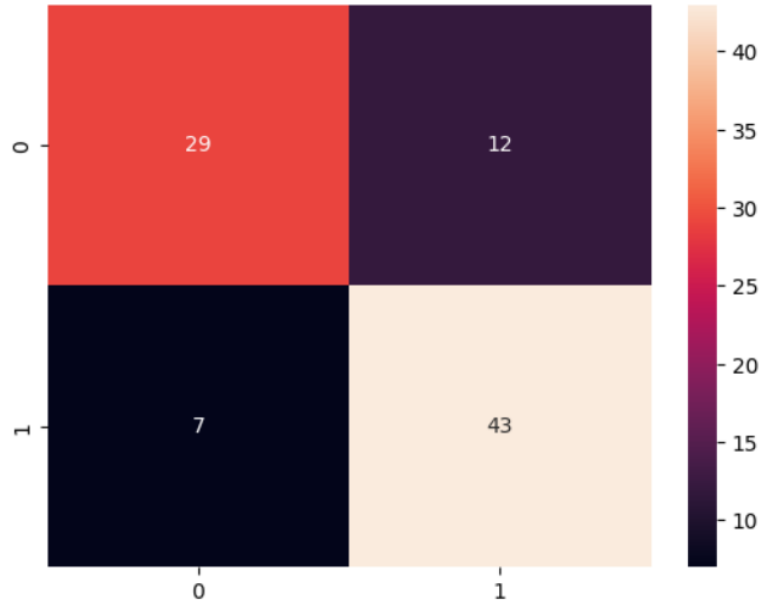
Kết quả đánh giá hiệu suất mô hình Support Vector Classifier

- Adaboost Classifier:

```
#Adaboost Classifier
CM = confusion_matrix(y_test, y_pred_ada)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN / (TN + FP)
loss_log = log_loss(y_test, y_pred_ada)
acc = accuracy_score(y_test, y_pred_ada)
roc = roc_auc_score(y_test, y_pred_ada)
prec = precision_score(y_test, y_pred_ada)
rec = recall_score(y_test, y_pred_ada)
f1 = f1_score(y_test, y_pred_ada)

mathew = matthews_corrcoef(y_test, y_pred_ada)
model_results = pd.DataFrame([['AdaBoost Classifier', acc, prec, rec, specificity, f1, roc, loss_log, mathew]],
                             columns=['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1 Score', 'ROC', 'Log_Los

print(model_results)
```

Đánh giá hiệu suất mô hình Adaboost Classifier

```
                Model  Accuracy  Precision  Sensitivity  Specificity
0  AdaBoost Classifier  0.791209   0.781818         0.86     0.707317  \

   F1 Score       ROC  Log_Loss  Matthews_corrcoef
0  0.819048  0.783659  7.525598           0.577258
```



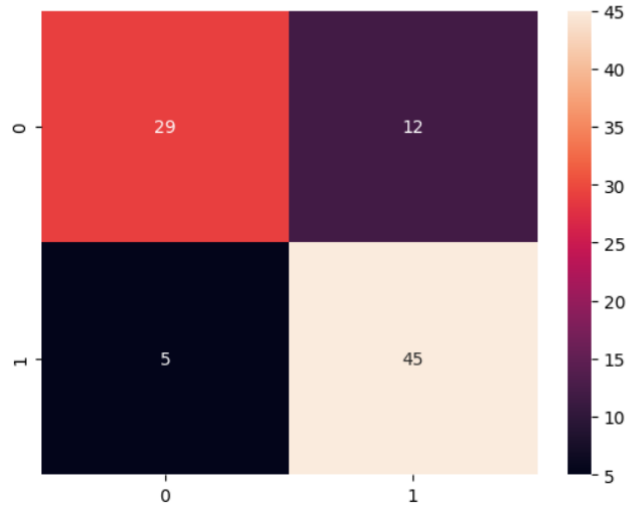
Kết quả đánh giá mô hình Adaboost Classifier

- Gradient boosting machine:

```
##Gradient boosting machine

CM = confusion_matrix(y_test, y_pred_gbm)
sns.heatmap(CM, annot=True)

# Tính toán các độ đo hiệu suất
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN / (TN + FP)
loss_log = log_loss(y_test, y_pred_gbm)
acc = accuracy_score(y_test, y_pred_gbm)
roc = roc_auc_score(y_test, y_pred_gbm)
prec = precision_score(y_test, y_pred_gbm)
rec = recall_score(y_test, y_pred_gbm)
f1 = f1_score(y_test, y_pred_gbm)
mathew = matthews_corrcoef(y_test, y_pred_gbm)

model_results = pd.DataFrame([['Gradient boosting machine', acc, prec, rec, specificity, f1, roc, loss_log, mathew]],
                             columns=['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1 Score', 'ROC', 'Log_Los

model_results
```

Đánh giá hiệu suất mô hình Gradient boosting machine

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	Matthews Corrcoef
0	Gradient boosting machine	0.813187	0.789474	0.9	0.707317	0.841121	0.803659	6.73343	0.624619



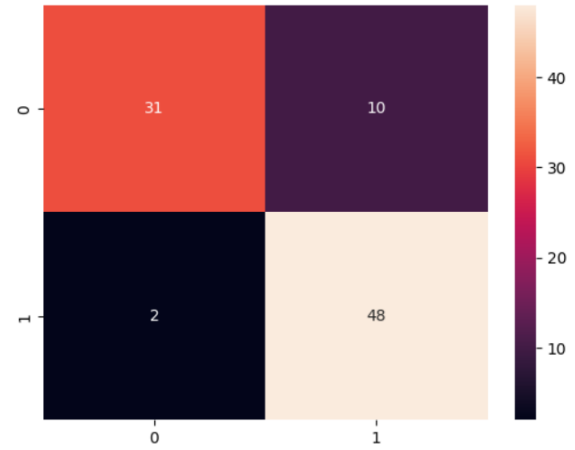
Kết quả đánh giá mô hình Gradient boosting

- Stack Ensemble Classifier:

```
CM=confusion_matrix(y_test,y_pred_all)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_all)
acc= accuracy_score(y_test, y_pred_all)
roc=roc_auc_score(y_test, y_pred_all)
prec = precision_score(y_test, y_pred_all)
rec = recall_score(y_test, y_pred_all)
f1 = f1_score(y_test, y_pred_all)

mathew = matthews_corrcoef(y_test, y_pred_all)
model_results =pd.DataFrame([['STacked Classifier',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
               columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrc

model_results
```

Đánh giá hiệu suất mô hình Stack Ensemble Classifier

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	STacked Classifier	0.868132	0.827586	0.96	0.756098	0.888889	0.858049	4.753009	0.741102



Kết quả đánh giá mô hình Ensemble Classifier

2.1.2 Train với KNN và 5 mô hình đại diện Random forest, SVM, KNN, Adaboost, Stacking thể hiện sự overfitting của tập dữ liệu:

a. Train với KNN:

Thực hiện quá trình phân chia dữ liệu và huấn luyện mô hình sử dụng thuật toán K-nearest neighbors (KNN) để dự đoán một biến mục tiêu.

```python
import plotly.graph_objs as go
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.cluster import AgglomerativeClustering
import warnings
import os
warnings.filterwarnings("ignore")
#First Model
kf = KFold(n_splits=4)
mae_train = []
mae_test = []
for train_index, test_index in kf.split(X):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model = KNeighborsClassifier(n_neighbors=2)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    mae_train.append(mean_absolute_error(y_train, y_train_pred))
    mae_test.append(mean_absolute_error(y_test, y_test_pred))

    print("First Model - Train mae: {} Test mae: {}".format(mae_train, mae_test))

```

```
First Model - Train mae: [0.19823788546255505] Test mae: [0.7368421052631579]
First Model - Train mae: [0.19823788546255505, 0.18061674008810572] Test mae: [0.7368421052631579
First Model - Train mae: [0.19823788546255505, 0.18061674008810572, 0.1762114537444934] Test mae:
First Model - Train mae: [0.19823788546255505, 0.18061674008810572, 0.1762114537444934, 0.1622807
```

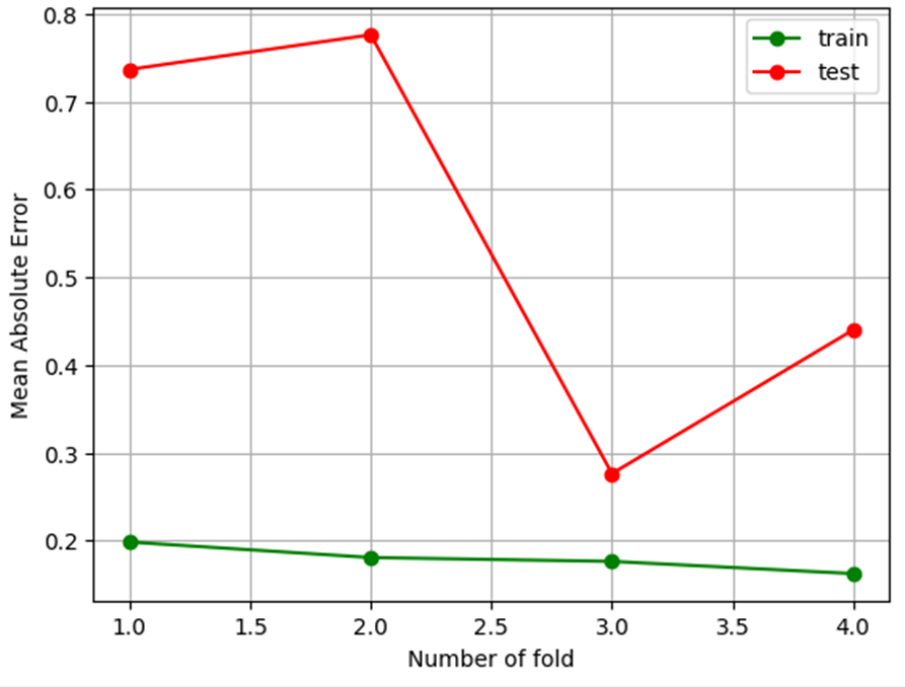
Train với kNN

Dự đoán kết quả trên tập kiểm tra và sau đó tính toán báo cáo phân loại sử dụng module classification_report từ sklearn.metrics.

```python
model.fit(X_train, y_train )
answer = model.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, answer))
```

```
              precision    recall  f1-score   support

           0       1.00      0.56      0.72        75
           1       0.00      0.00      0.00         0

    accuracy                           0.56        75
   macro avg       0.50      0.28      0.36        75
weighted avg       1.00      0.56      0.72        75
```

Báo cáo chi tiết về hiệu suất mô hình.



Biểu đồ kết quả

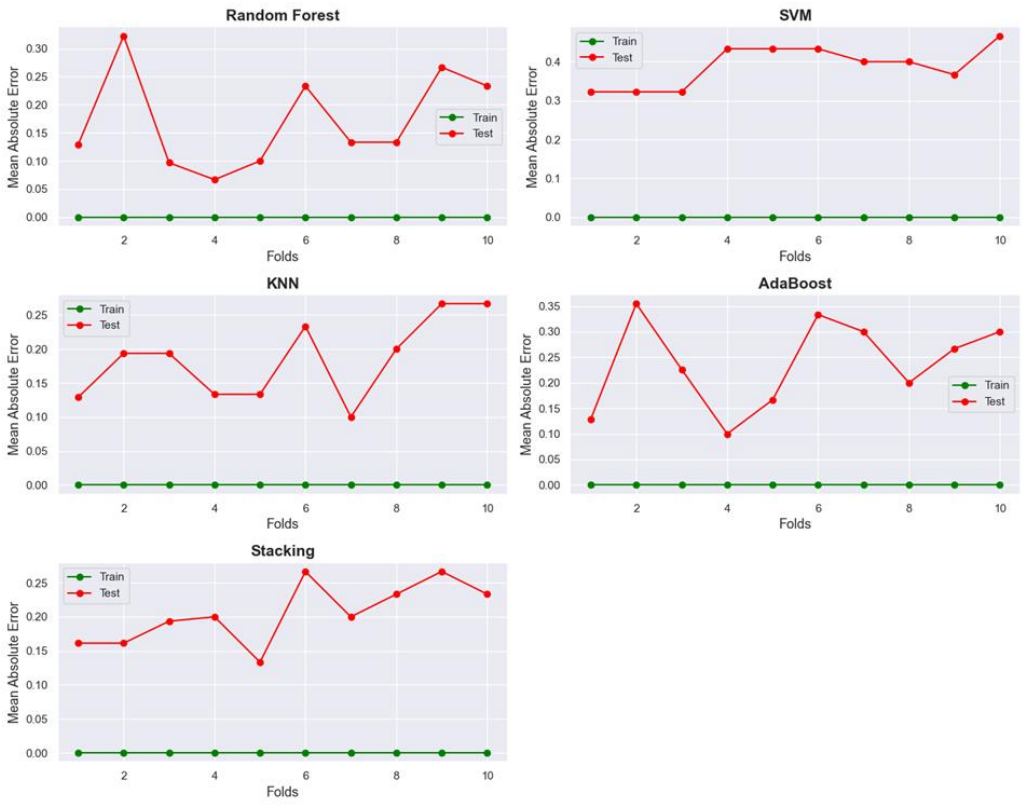
So sánh giữa Erorr của training set và test set, có thể thấy Erorrr của training set có giá trị rất nhỏ (từ biểu đồ có MAE hoặc Mean absolute error) thấp hơn 0,2, trong khi error Test set có giá trị lớn hơn rất nhiều nằm trong khoảng 0,3 - 0,8. Có thể nói rằng trong khi huấn luyện mô hình này, có thể cho kết quả dự đoán chính xác. Tuy nhiên, khi áp dụng cho tập dữ liệu mới (Test set), sai số sẽ lớn hơn đáng kể. Điều này cho thấy tình trạng overfitting của tập dữ liệu này.

b. 5 mô hình đại diện Random forest, SVM, KNN, Adaboost, Stacking:

Thực hiện một quá trình huấn luyện và đánh giá nhiều mô hình máy học khác nhau sử dụng phương pháp phân chia dữ liệu K-fold cross-validation.

```python
for train_index,test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #random forest
    final_models[0].fit(X_train,y_train)
    test_eval(final_models[0],X_test,y_test,model_names[0])

    #svm
    final_models[1].fit(X_train,y_train)
    test_eval(final_models[1],X_test,y_test,model_names[1])

    #knn
    final_models[2].fit(X_train,y_train)
    test_eval(final_models[2],X_test,y_test,model_names[2])

    #adaboost
    final_models[3].fit(X_train,y_train)
    test_eval(final_models[3],X_test,y_test,model_names[3])

    #stacking
    final_models[4].fit(X_train,y_train)
    test_eval(final_models[4],X_test,y_test,model_names[4])
```

Huấn luyện trên nhiều mô hình



Biểu đồ kết quả

Biểu đồ MSE 5 FIRST: thể hiện chỉ số MAE đang rất cao lúc train bằng 5 mô hình Random forest, SVM, KNN, Adaboost, Stacking không sử dụng các biện pháp giảm overfitting

Mức độ overfitting của mô hình Heart.csv có xu hướng giảm đi đã thấy trong các biểu đồ, tuy nhiên mức MAE vẫn còn khá cao, sau đây là các biện pháp giảm overfitting đồng thời cải thiện độ chính xác của các mô hình đang được nghiên cứu.

2.1.3 Giảm overfitting với Random Over Sampling

Áp dụng Random Over Sampling (Lấy mẫu ngẫu nhiên) để tránh mất cân bằng dữ liệu.

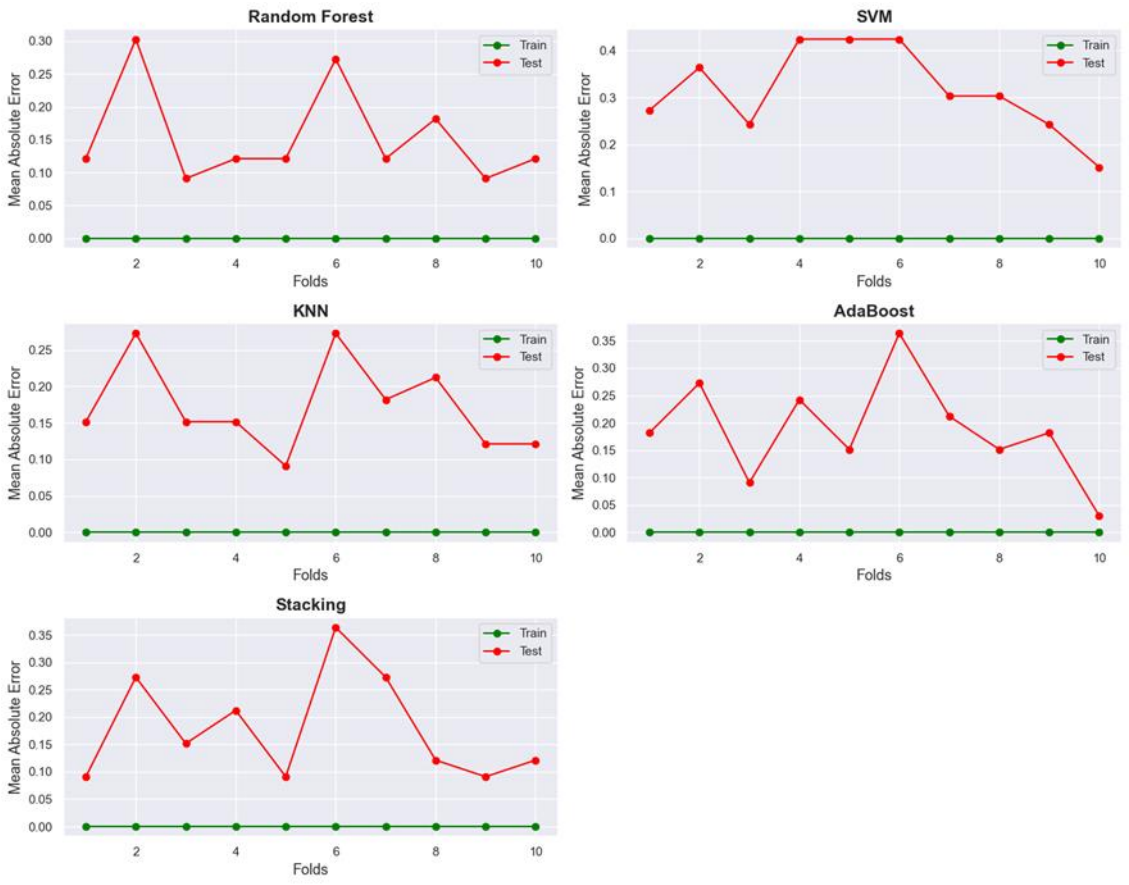
```
sampler = RandomOverSampler()

X_sampler,y_sampler = sampler.fit_resample(X,y)

data_resample = pd.concat([X_sampler,y_sampler],axis=1)

X = data_resample.drop('output',axis=1)
y = data_resample['output']
```

Random Over Sampling

```
for train_index,test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #random forest
    final_models[0].fit(X_train,y_train)
    test_eval(final_models[0],X_test,y_test,model_names[0])

    #svm
    final_models[1].fit(X_train,y_train)
    test_eval(final_models[1],X_test,y_test,model_names[1])

    #knn
    final_models[2].fit(X_train,y_train)
    test_eval(final_models[2],X_test,y_test,model_names[2])

    #adaboost
    final_models[3].fit(X_train,y_train)
    test_eval(final_models[3],X_test,y_test,model_names[3])

    #stacking
    final_models[4].fit(X_train,y_train)
    test_eval(final_models[4],X_test,y_test,model_names[4])
```

Huấn luyện các mô hình

Thực hiện một quá trình huấn luyện và đánh giá nhiều mô hình máy học khác nhau sử dụng phương pháp phân chia dữ liệu K-fold cross-validation.



Biểu đồ MSE với 5 mô hình sau khi sử dụng Random Over Sampling

Sau khi sử dụng phương pháp Random Over Sampling, mức độ overfiting của mô hình Heart.csv có xu hướng càng về sau càng giảm đi một chút như đã thấy trong các biểu đồ Random Forest, KNN và Stacking so với biểu đồ tương tự trên không sử dụng Random over sample.

Mô hình không đạt được mức độ phù hợp như mong muốn, tuy nhiên khi áp dụng Random Over Sampling, hiệu suất của mô hình được cải thiện một chút. Việc thêm nhiều dữ liệu hơn vào tập dữ liệu có thể là giải pháp để giải quyết triệt để tình trạng overfitting và điểm số liệu thấp.

2.1.4 Đơn giản hóa mô hình với Staking classifier và KNN để giảm overfitting và tăng độ chính xác của mô hình:

a. Đơn giản hóa dữ liệu với Stacking Classifier

Với hàm MinMaxScaler, nghĩa là chỉ chia tỷ lệ dữ liệu giữa các giá trị tối đa và tối thiểu. Điều này được thực hiện để dữ liệu không bị ảnh hưởng bởi dữ liệu ngoại lệ hoặc có các ngoại lệ.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
for column in X.columns:
    feature = np.array(X[column]).reshape(-1,1)
    scaler = MinMaxScaler()
    scaler.fit(feature)
    feature_scaled = scaler.transform(feature)
    X[column] = feature_scaled.reshape(1,-1)[0]
```

Train với Stacking

Train với stacking với 4 mô hình RandomForestClassifier, KNeighborsClassifier, SVC, AdaBoostClassifier

```
import xgboost as xgboost
# selecting list of top performing models to be used in stacked ensemble method
models = [
    RandomForestClassifier(criterion='entropy',n_estimators=100),
    KNeighborsClassifier(9),
    SVC(probability=True),
    AdaBoostClassifier(),

]
```

```
Stackf = KFold(n_splits=4)
mae_train = []
mae_test = []
for train_index, test_index in Stackf.split(X):

    S_train, S_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model_af = MLPClassifier()
    model_af.fit(S_train, y_train)
    y_train_pred = model_af.predict(S_train)
    y_test_pred = model_af.predict(S_test)
    mae_train.append(mean_absolute_error(y_train, y_train_pred))
    mae_test.append(mean_absolute_error(y_test, y_test_pred))

    print("1 Solution - Train mae: {} Test mae: {}".format(mae_train, mae_test))
```

Huấn luyện mô hình

Sử dụng mô hình model_af để huấn luyện trên tập huấn luyện S_train và nhãn tương ứng y_train. Sau đó, chúng ta sử dụng mô hình đã được huấn luyện để dự đoán kết quả trên tập kiểm tra S_test.

```
1  model_af.fit(S_train, y_train )
2  answer = model_af.predict(S_test)
3  from sklearn.metrics import classification_report
4  print(classification_report(y_test, answer))
```

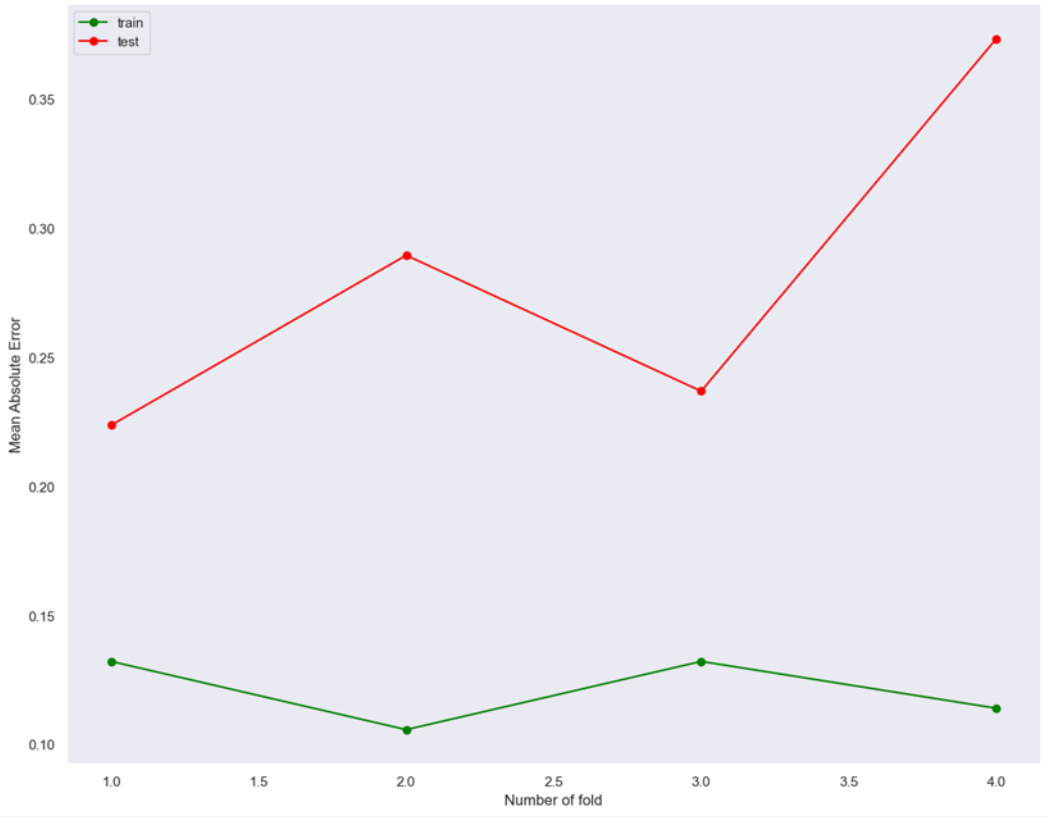
```
              precision    recall  f1-score   support

           0       1.00      0.64      0.78        75
           1       0.00      0.00      0.00         0

    accuracy                           0.64        75
   macro avg       0.50      0.32      0.39        75
weighted avg       1.00      0.64      0.78        75
```

Báo cáo chi tiết về hiệu suất của mô hình

```
1  y_pred_Stackf = model_af.predict(S_test)
2  accuracy = accuracy_score(y_test, y_pred_Stackf)
3  print("Accuracy:", accuracy)
```

```
Accuracy: 0.64
```

Kết quả độ chính xác



Biểu đồ kết quả

Kết quả là một biểu đồ trong đó các giá trị lỗi từ tập huấn luyện có xu hướng giảm. Eror từ Test set tuy có xu hướng tăng nhưng giá trị error vẫn rất nhỏ (<0.4) so với con số từ 0.3 - 0.8 trước đó. Stacking classifier vẫn giảm overfitting với phương pháp Đơn giản hóa.

b. Đơn giản hóa dữ liệu với KNN

Tương tự như Staking classifier, sử dụng KNN để đơn giản hóa dữ liệu với MinMaxScaler:

```
kf = KFold(n_splits=4)
mae_train = []
mae_test = []
for train_index, test_index in kf.split(X):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model_af = KNeighborsClassifier(n_neighbors=2)
    model_af.fit(X_train, y_train)
    y_train_pred = model_af.predict(X_train)
    y_test_pred = model_af.predict(X_test)
    mae_train.append(mean_absolute_error(y_train, y_train_pred))
    mae_test.append(mean_absolute_error(y_test, y_test_pred))

    print("1 Solution - Train mae: {} Test mae: {}".format(mae_train, mae_test))
```

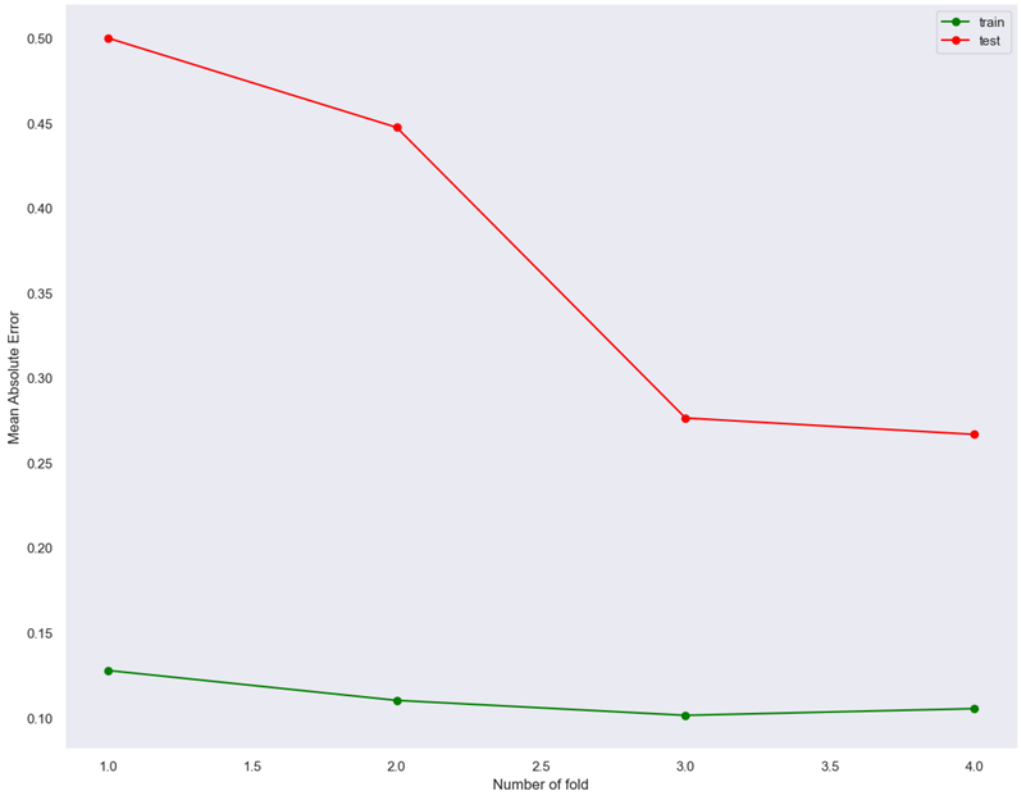
Sử dụng phương pháp K-fold cross-validation để huấn luyện và đánh giá mô hình model_af sử dụng thuật toán KNeighborsClassifier trên dữ liệu X và nhãn y.

```
1  model_af.fit(X_train, y_train )
2  answer = model_af.predict(X_test)
3  from sklearn.metrics import classification_report
4  print(classification_report(y_test, answer))
```

```
              precision    recall  f1-score   support

           0       1.00      0.73      0.85        75
           1       0.00      0.00      0.00         0

    accuracy                           0.73        75
   macro avg       0.50      0.37      0.42        75
weighted avg       1.00      0.73      0.85        75
```

```
1  y_pred = model_af.predict(X_test)
2  accuracy = accuracy_score(y_test, y_pred)
3  print("Accuracy:", accuracy)
```

```
Accuracy: 0.7333333333333333
```



Kết quả là một biểu đồ trong đó các giá trị lỗi từ test set có xu hướng giảm đáng kể so với gái trị đầu. Nó có thể là 0,25 - 0,50.

2.1.5 Select Feature để giảm OVERFITTING và Tăng độ chính xác cho mô hình

Sử dụng lựa chọn các tính năng quan trọng để mô hình không cần phải học hỏi quá nhiều tính năng.

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)
fit.scores_
dfscores = pd.DataFrame(fit.scores_)
dfcol = pd.DataFrame(X.columns)
```

Trong đoạn mã trên, chúng ta sử dụng module sklearn.feature_selection để thực hiện việc chọn đặc trưng (feature selection) bằng phương pháp chi-square (chi2) và lựa chọn K tốt nhất (SelectKBest) từ dữ liệu X và nhãn y.

```
featureScore = pd.concat([dfcol, dfscores], axis = 1)
featureScore.columns = ['Feature', 'Score']

print(featureScore.nlargest(4, 'Score'))
```

```
    Feature      Score
8      exng  38.914377
2        cp  20.866033
11      caa  16.610191
9   oldpeak  11.716815
```

Sử dụng phương pháp K-fold cross-validation để huấn luyện và đánh giá mô hình model_af2 sử dụng thuật toán KNeighborsClassifier trên dữ liệu X_new và nhãn y.

```
kf = KFold(n_splits=4)
mae_train_2 = []
mae_test_2 = []
for train_index, test_index in kf.split(X_new):

    X_train, X_test = X_new[train_index], X_new[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model_af2 = KNeighborsClassifier(n_neighbors=2)
    model_af2.fit(X_train, y_train)
    y_train_pred = model_af2.predict(X_train)
    y_test_pred = model_af2.predict(X_test)
    mae_train_2.append(mean_absolute_error(y_train, y_train_pred))
    mae_test_2.append(mean_absolute_error(y_test, y_test_pred))
print("2 Solution - Train mae: {} Test mae: {}".format(mae_train_2, mae_test_2))
```

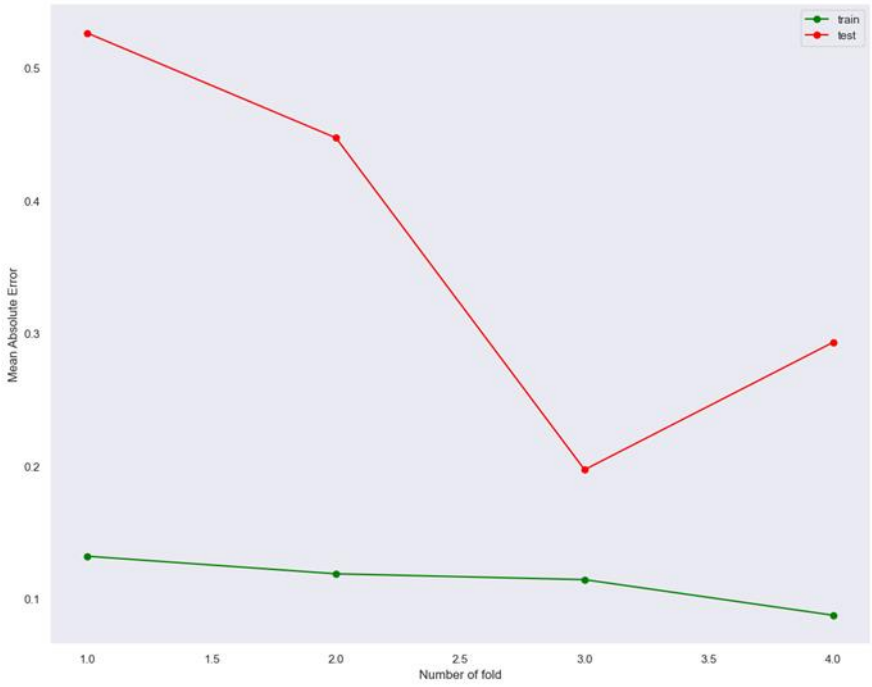
```
model_af2.fit(X_train, y_train )
answer = model_af2.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, answer))
```

```
              precision    recall  f1-score   support

           0       1.00      0.71      0.83        75
           1       0.00      0.00      0.00         0

    accuracy                           0.71        75
   macro avg       0.50      0.35      0.41        75
weighted avg       1.00      0.71      0.83        75
```

Sử dụng mô hình model_af2 đã được huấn luyện trên tập huấn luyện (X_train, y_train) để dự đoán nhãn trên tập kiểm tra (X_test).

```
y_pred = model_af2.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7066666666666667
```



Bằng cách sử dụng FS này, có thể thấy các giá trị MAE của mô hình giữa tập Huấn luyện và tập Kiểm tra đang bắt đầu xích lại gần nhau hơn từ 0.2 - 0.55

2.1.6 Train một mô hình đơn giản khác

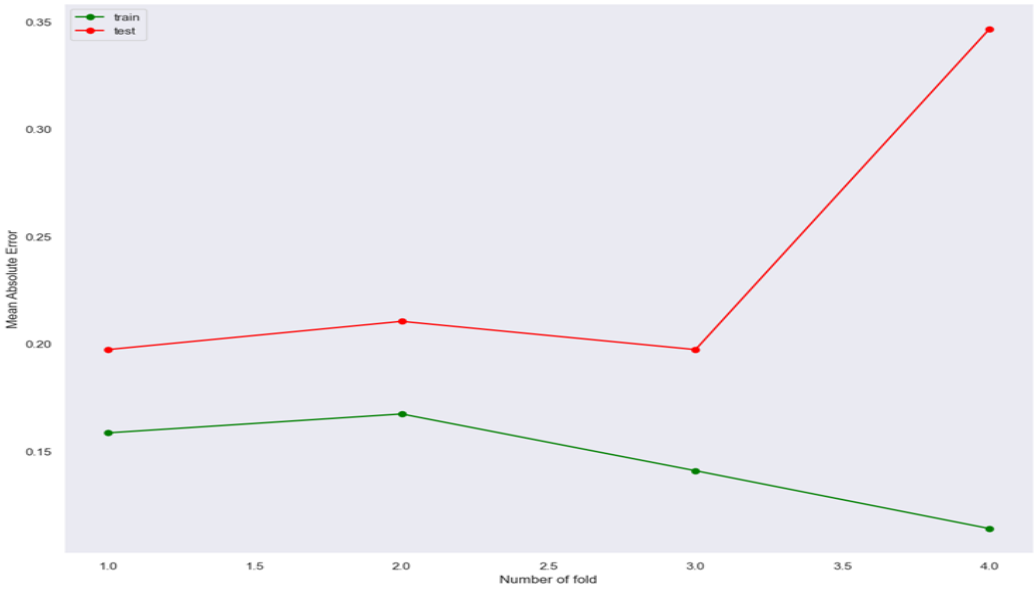
Sử dụng Naive Bayes vì đây là mô hình không quá phức tạp và khá đơn giản ở chỗ là hàm giả thuyết Simple (linear), giúp giảm thiểu vấn đề overfitting.

```
from sklearn.naive_bayes import GaussianNB
kf = KFold(n_splits=4)
mae_train_gnb = []
mae_test_gnb = []
gnb = GaussianNB()
for train_index, test_index in kf.split(X):

    X_train_gnb, X_test_gnb = X.iloc[train_index], X.iloc[test_index]
    y_train_gnb, y_test_gnb = y[train_index], y[test_index]
    gnb.fit(X_train_gnb, y_train_gnb)
    y_train_pred_gnb = gnb.predict(X_train_gnb)
    y_test_pred_gnb = gnb.predict(X_test_gnb)

    print("Gnb model - Train mae: {} Test mae: {}".format(mae_train_gnb, mae_test_gnb))

    mae_train_gnb.append(mean_absolute_error(y_train_gnb, y_train_pred_gnb))
    mae_test_gnb.append(mean_absolute_error(y_test_gnb, y_test_pred_gnb))
```

Train mô hình

```
1   y_pred = gnb.predict(X_test_gnb)
2   accuracy = accuracy_score(y_test_gnb, y_pred)
3   print("Accuracy:", accuracy)

Accuracy: 0.6533333333333333
```

Tính độ chính xác



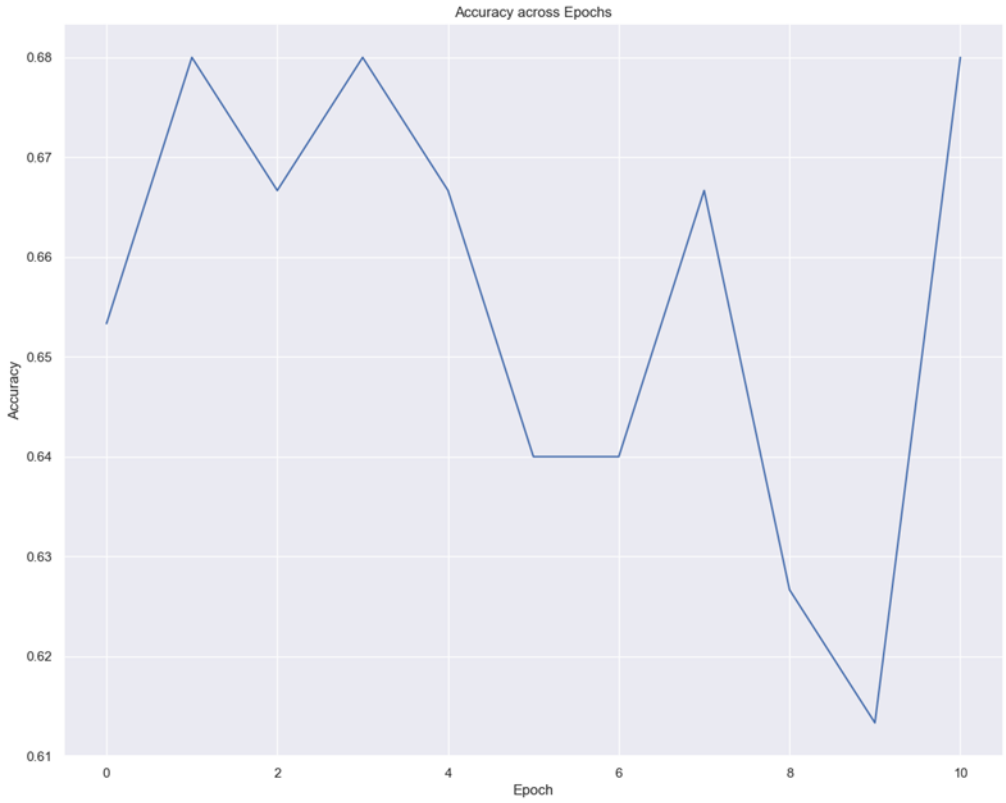
MAE value mà Giá trị thu được chỉ nằm trong khoảng 0,20 - 0,35, tốt hơn so với KNN dù đã sử dụng featear selection

2.1.7 EARLY STOP với ADA Class

Giá trị mang tính tham khảo dựa trên ý tưởng chính là theo dõi hiệu suất và dừng quá trình huấn luyện khi không có cải thiện đáng kể.

```
import matplotlib.pyplot as plt
epoch_list = []
accuracy_list = []

best_model = None
best_accuracy = 0
tolerance = 9  # Số lượng epochs không có cải thiện chấp nhận trước khi dừng lại
no_improvement_count = 0

for epoch in range(100):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Epoch:", epoch, "Accuracy:", accuracy)

    epoch_list.append(epoch) ##add thông tin
    accuracy_list.append(accuracy)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = model
        no_improvement_count = 0
    else:
        no_improvement_count += 1

    if no_improvement_count >= tolerance:
        print("Early stopping at epoch", epoch)
        break
y_pred = best_model.predict(X_test) #Giá trị mang tính tham khảo dựa trên ý tưởng chính là theo dõi hiệu suất
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy after early stopping:", accuracy)
```

```
Epoch: 0 Accuracy: 0.6533333333333333
Epoch: 1 Accuracy: 0.68
Epoch: 2 Accuracy: 0.6666666666666666
Epoch: 3 Accuracy: 0.68
Epoch: 4 Accuracy: 0.6666666666666666
Epoch: 5 Accuracy: 0.64
Epoch: 6 Accuracy: 0.64
Epoch: 7 Accuracy: 0.6666666666666666
Epoch: 8 Accuracy: 0.6266666666666667
Epoch: 9 Accuracy: 0.6133333333333333
Epoch: 10 Accuracy: 0.68
Early stopping at epoch 10
Accuracy after early stopping: 0.68
```



2.1.8 Cross validation

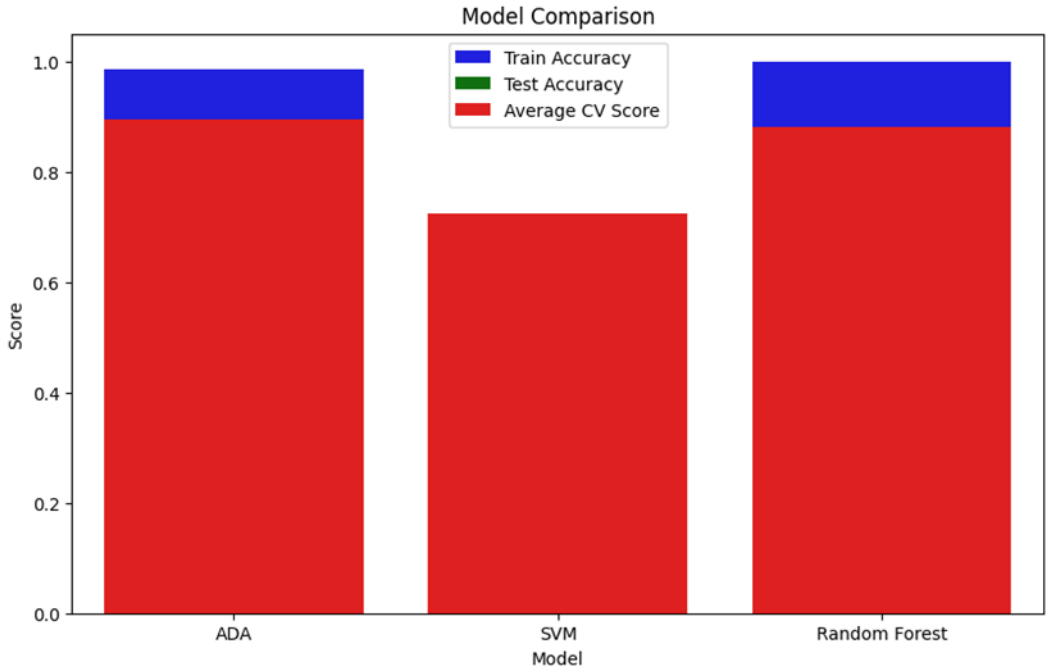
Được sử dụng nhiều trong bài từ những kỹ thuật tránh Overfitting và Tăng độ chính xác đã được sử dụng ở trên. Đánh giá mô hình trên nhiều tập dữ liệu khác nhau và đảm bảo rằng mô hình không chỉ hoạt động tốt trên tập huấn luyện mà còn tổng quát hóa tốt trên các tập dữ liệu mới.

Huấn luyện và đánh giá các mô hình AdaBoostClassifier, SVC và RandomForestClassifier

```
models = [
    ("ADA", AdaBoostClassifier()),
    ("SVM", SVC()),
    ("Random Forest", RandomForestClassifier())
]
results = []

# Huấn luyện và đánh giá từng mô hình
for name, model in models:
    # Huấn luyện mô hình
    model.fit(X_train, y_train)

    # Dự đoán trên tập huấn luyện và tập kiểm tra
    train_predictions = model.predict(X_train)
    test_predictions = model.predict(X_test)

    # Tính toán độ chính xác
    train_accuracy = accuracy_score(y_train, train_predictions)
    test_accuracy = accuracy_score(y_test, test_predictions)
    print(f"{name}:")
    print("Train Accuracy:", train_accuracy)
    print("Test Accuracy:", test_accuracy)
    cv_scores = cross_val_score(model, X_train, y_train, cv=5)
    avg_cv_score = np.mean(cv_scores)
```

```python
    # Áp dụng kỹ thuật cross-validation để đánh giá mô hình
    cv_scores = cross_val_score(model, X_train, y_train, cv=5)
    print("Cross-Validation Scores:", cv_scores)
    print("Average Cross-Validation Score:", np.mean(cv_scores))

    results.append({
        'Model': name,
        'Train Accuracy': train_accuracy,
        'Test Accuracy': test_accuracy,
        'Average CV Score': avg_cv_score
    })

results_df = pd.DataFrame(results)


plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Train Accuracy', data=results_df, label='Train Accuracy', color='b')
sns.barplot(x='Model', y='Test Accuracy', data=results_df, label='Test Accuracy', color='g')
sns.barplot(x='Model', y='Average CV Score', data=results_df, label='Average CV Score', color='r')

plt.title('Model Comparison')
plt.ylabel('Score')
plt.legend()
plt.show()
```

```
ADA:
Train Accuracy: 0.9868421052631579
Test Accuracy: 0.6
Cross-Validation Scores: [0.84782609 0.95652174 0.86956522 0.93333333 0.86666667]
Average Cross-Validation Score: 0.8947826086956521
SVM:
Train Accuracy: 0.7236842105263158
Test Accuracy: 0.0
Cross-Validation Scores: [0.7173913  0.7173913  0.7173913  0.73333333 0.73333333]
Average Cross-Validation Score: 0.723768115942029
Random Forest:
Train Accuracy: 1.0
Test Accuracy: 0.5333333333333333
Cross-Validation Scores: [0.89130435 0.89130435 0.7826087  0.88888889 0.91111111]
Average Cross-Validation Score: 0.8730434782608695
```



- Đối với mô hình AdaBoostClassifier (ADA), ta có:

Độ chính xác trên tập huấn luyện: 0.9868

Độ chính xác trên tập kiểm tra: 0.6

Điểm số cross-validation trung bình: 0.8948

Mô hình AdaBoostClassifier có độ chính xác khá cao trên tập huấn luyện (98.68%), tuy nhiên, độ chính xác trên tập kiểm tra (60%) thấp hơn nhiều. Điểm số cross-validation trung bình cũng chỉ ở mức 89.48%. Điều này cho thấy mô hình có khả năng bị overfitting.

- Với mô hình SVM, ta có:

Độ chính xác trên tập huấn luyện: 0.7237

Độ chính xác trên tập kiểm tra: 0.0

Điểm số cross-validation trung bình: 0.7238

Mô hình SVM có độ chính xác khá thấp cả trên tập huấn luyện (72.37%) và tập kiểm tra (0%). Điểm số cross-validation trung bình cũng không cao (72.38%). Điều này cho thấy mô hình không hiệu quả trong việc dự đoán.

- Và với mô hình Random Forest:

Độ chính xác trên tập huấn luyện: 1.0

Độ chính xác trên tập kiểm tra: 0.5333

Điểm số cross-validation trung bình: 0.8730

Mô hình Random Forest có độ chính xác 100% trên tập huấn luyện, nhưng độ chính xác trên tập kiểm tra (53.33%) thấp hơn nhiều. Điểm số cross-validation trung bình là 87.30%. Điều này cho thấy mô hình có khả năng bị overfitting và không thể tổng quát hóa tốt trên dữ liệu mới.

Cross validation cho ta thấy được không có mô hình nào cho kết quả tốt một cách toàn diện. Mô hình AdaBoostClassifier có độ chính xác cao trên tập huấn luyện, nhưng không tổng quát hóa tốt trên tập kiểm tra. Mô hình SVM và Random Forest đều cho kết quả không tốt cả trên tập huấn luyện và tập kiểm tra. Để tránh Overfitting và tăng độ chính xác, ta có thể dùng Xác thực chéo kiểm tra và sử dụng mô hình phù hợp nhất với nhu cầu.

2.2 Feed Forward Neural Network và Reccurent Neural Network:

2.2.1 Feed Forward Neural Network:

Sau khi đã phân tích, chuẩn hóa và huấn luyện dữ liệu sao cho phù hợp với yêu cầu của bài toán, chúng em tiến hành xây dựng mô hình Feedforward Neural Networl bằng cách sử dụng thư viện TensorFlow và Keras.

```
model_ffnn = Sequential()
model_ffnn.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model_ffnn.add(Dense(64, activation='relu'))
model_ffnn.add(Dense(1, activation='sigmoid'))


# Biên dịch mô hình
model_ffnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Huấn luyện mô hình trên tập huấn luyện
history_ffnn = model_ffnn.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test))
```

Xây dựng kiến trúc mô hình FFNN

Sử dụng lớp Dense để thêm các tầng fully connected layers cho mô hình. Số đơn vị của mỗi tầng được chỉ định bởi các tham số (128, 64) và lớp cuối cùng có một đơn vị với hàm kích hoạt Sigmoid.

Tiến hành huấn luyện mô hình với phương pháp Optimizer là Adam, trong 30 lần học với kích thước batch là 32.

```
y_pred = model_ffnn.predict(X_test)
y_pred = [1 if y>=0.5 else 0 for y in y_pred]
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy: %.2f%%" % (accuracy*100))
```

```
3/3 [==============================] - 0s 1ms/step
Model Accuracy: 82.42%
```

Tính toán độ chính xác của mô hình trên tập kiểm thử

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.68      0.78        41
           1       0.78      0.94      0.85        50

    accuracy                           0.82        91
   macro avg       0.84      0.81      0.82        91
weighted avg       0.84      0.82      0.82        91
```

Báo cáo chi tiết về hiệu suất của mô hình

2.2.2 Recurrent Neural Network:

Tiến hành xây dựng mô hình Simple Recurrent Neural Network (SimpleRNN) bằng cách sử dụng thư viện Keras theo kiểu layer by layer.

```
model_rnn = Sequential()
model_rnn.add(SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], 1)))
model_rnn.add(Dense(1, activation='sigmoid'))

model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

X_train_rnn = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_rnn = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))

history_rnn = model_rnn.fit(X_train_rnn, y_train, epochs=40, batch_size=32, validation_data=(X_test_rnn, y_test))
```

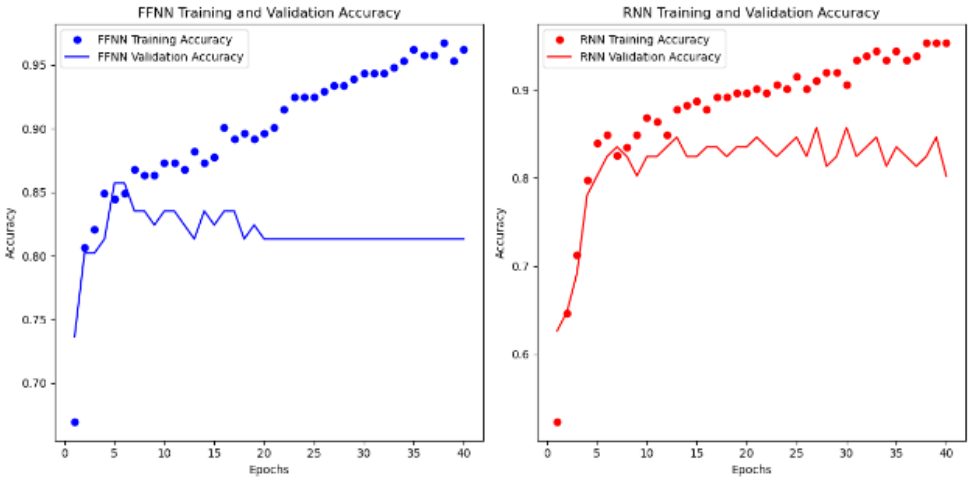
Xây dựng mô hình Simple Recurrent Neural Network.

```
y_pred_rnn = model_rnn.predict(X_test_rnn)
y_pred_rnn = [1 if y >= 0.5 else 0 for y in y_pred_rnn]
accuracy_rnn = accuracy_score(y_test, y_pred_rnn)

print("RNN Model Accuracy: %.2f%%" % (accuracy_rnn * 100))
print(classification_report(y_test, y_pred_rnn))
```

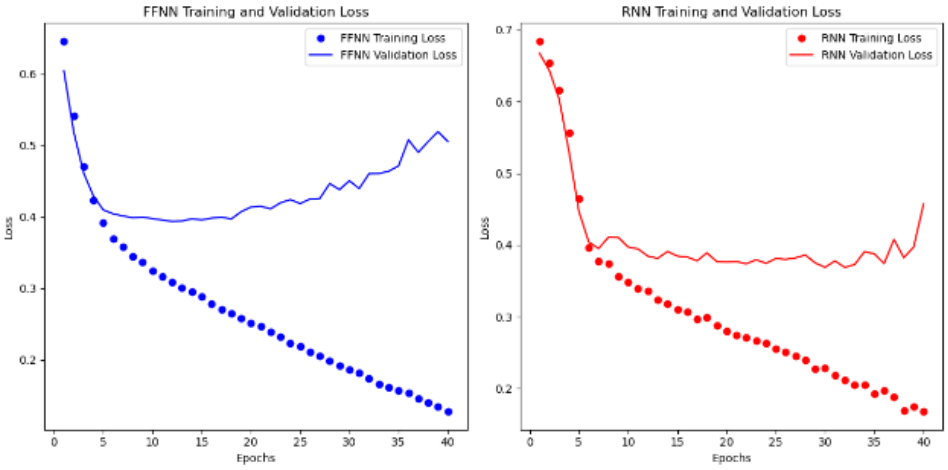
```
3/3 [==============================] - 0s 2ms/step
RNN Model Accuracy: 80.22%
              precision    recall  f1-score   support

           0       0.81      0.73      0.77        41
           1       0.80      0.86      0.83        50

    accuracy                           0.80        91
   macro avg       0.80      0.80      0.80        91
weighted avg       0.80      0.80      0.80        91
```

Tính độ chính xác và báo cáo chi tiết về hiệu suất mô hình.

2.2.3 So sánh mô hình:



So sánh độ chính xác giữa hai mô hình



So sánh tỉ lệ mất mát giữ hai mô hình

- Nhận xét:
    - Cả 2 mô hình đều có xu hướng tăng độ chính xác và giảm mất mát qua các epoch
    - Mô hình FFNN đạt được hiệu suất tốt hơn với độ chính xác và loss thấp hơn trên cả tập huấn luyện và tập kiểm tra.
    - RNN cần ít epoch hơn để đạt được hiệu suất tốt nhưng độ hiệu quả không bằng FFNN.
    - ⇨ Dựa trên các so sánh trên FFNN là mô hình hiệu quả hơn cho bài toán này.

2.2.4 Áp dụng kỹ thuật tránh Overfitting cho hai mô hình:

- Mô hình FFNN:

- Áp dụng các kỹ thuật tránh Overfitting: Early stop, Drop out và L1 Regularization cho mô hình FFNN:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping

# Create a sequential model
model_ffnn = Sequential()

# Add the first hidden layer with dropout
model_ffnn.add(Dense(64, input_dim=7, activation='relu', kernel_regularizer=regularizers.l1(1e-6)))
model_ffnn.add(Dropout(0.5))  # Add Dropout with a dropout rate of 0.5

# Add the second hidden layer with dropout
model_ffnn.add(Dense(32, activation='relu'))
model_ffnn.add(Dropout(0.5))  # Add Dropout with a dropout rate of 0.5

# Add the third hidden layer
model_ffnn.add(Dense(13, activation='relu'))

# Finally, add the output layer
model_ffnn.add(Dense(1, activation='sigmoid'))

# Compile the model
model_ffnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=10)
```

```
# Biên dịch mô hình
model_ffnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Huấn luyện mô hình trên tập huấn luyện
history_ffnn = model_ffnn.fit(X_train, y_train, epochs=90, batch_size=16,validation_data=(X_test, y_test),callbacks = [early_stop
```

Sử dụng các kỹ thuật tránh Overfitting

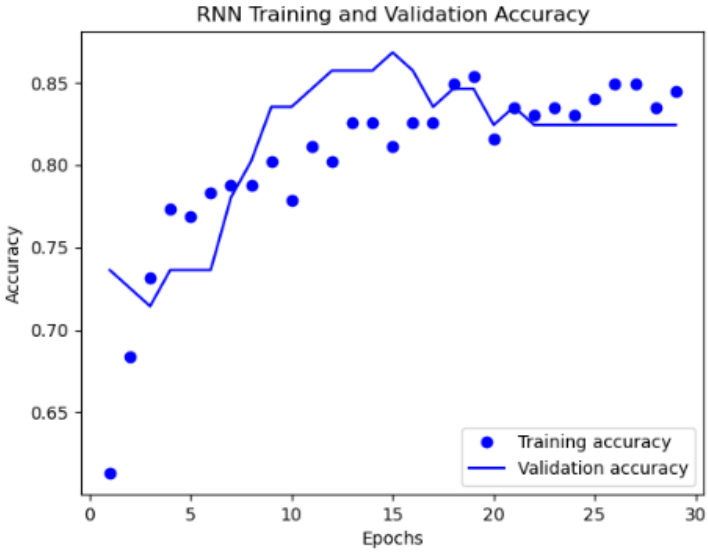
```
y_pred = model_ffnn.predict(X_test)
y_pred = [1 if y>=0.5 else 0 for y in y_pred]
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy: %.2f%%" % (accuracy*100))
```

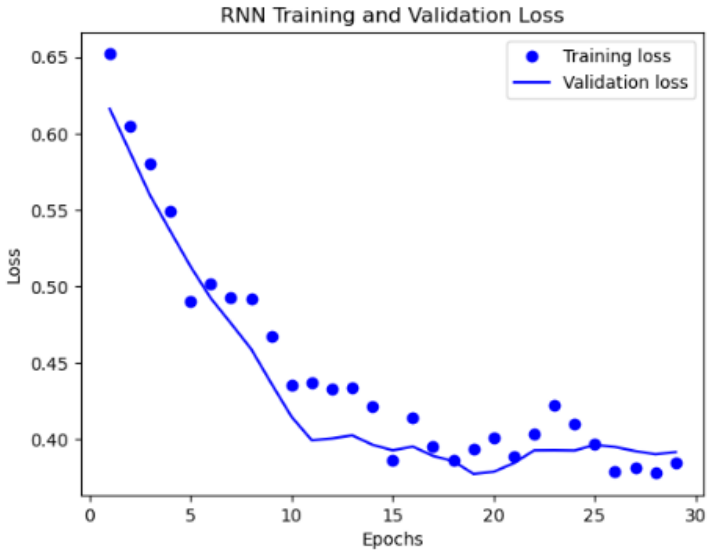
```
3/3 [==============================] - 0s 2ms/step
Model Accuracy: 86.81%
```

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.85	0.85	0.85	41
1	0.88	0.88	0.88	50
accuracy			0.87	91
macro avg	0.87	0.87	0.87	91
weighted avg	0.87	0.87	0.87	91

Tính độ chính xác và báo cáo chi tiết về hiệu suất mô hình.


RNN Training and Validation Accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy
0.85
0.80
0.75
0.70
0.65
0
5
10
15
20
25
30


RNN Training and Validation Loss
Loss
Epochs
Training loss
Validation loss
0.65
0.60
0.55
0.50
0.45
0.40
0
5
10
15
20
25
30

Mô hình RNN:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping

# Create a sequential model for RNN
model_rnn = Sequential()

# Add the SimpleRNN layer with dropout
model_rnn.add(SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], 1), kernel_regularizer=regularizers.l1(1e-6)))
model_rnn.add(Dropout(0.5))  # Add Dropout with a dropout rate of 0.5

# Add the output layer
model_rnn.add(Dense(1, activation='sigmoid'))

# Compile the model
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=10)
```

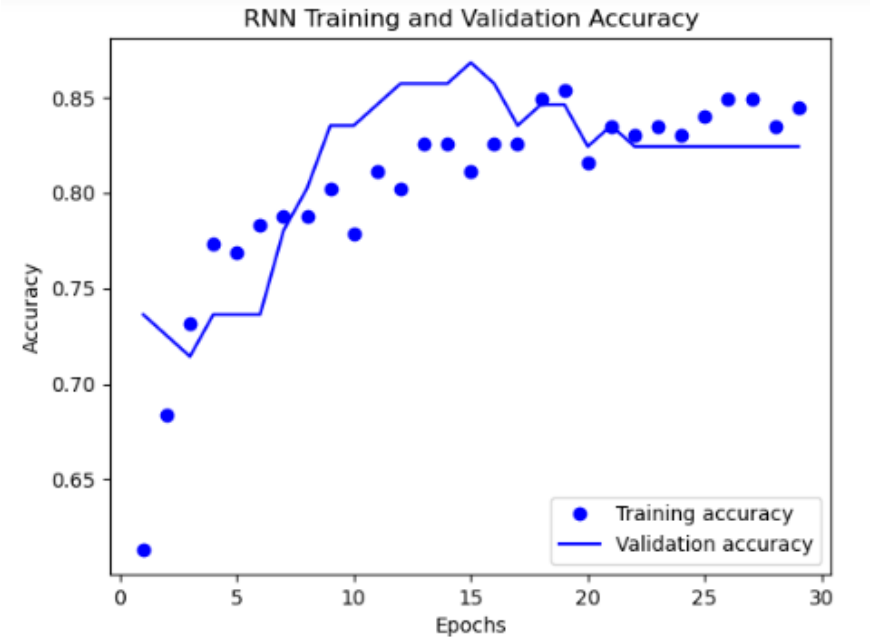
```
# Huấn luyện mô hình
history_rnn = model_rnn.fit(X_train_rnn, y_train, epochs=80, batch_size=32, validation_data=(X_test_rnn, y_test),callbacks = [ea
```

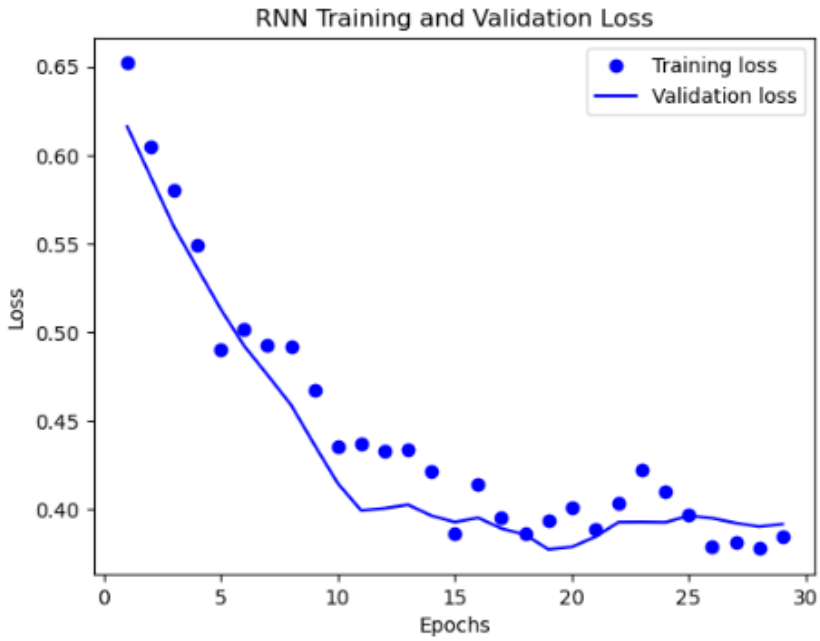
Sử dụng các kỹ thuật tránh Overfitting

```
# Đánh giá mô hình
y_pred_rnn = model_rnn.predict(X_test_rnn)
y_pred_rnn = [1 if y >= 0.5 else 0 for y in y_pred_rnn]
accuracy_rnn = accuracy_score(y_test, y_pred_rnn)

print("RNN Model Accuracy: %.2f%%" % (accuracy_rnn * 100))
print(classification_report(y_test, y_pred_rnn))
```

```
3/3 [==============================] - 0s 2ms/step
RNN Model Accuracy: 82.42%
              precision    recall  f1-score   support

           0       0.84      0.76      0.79        41
           1       0.81      0.88      0.85        50

    accuracy                           0.82        91
   macro avg       0.83      0.82      0.82        91
weighted avg       0.83      0.82      0.82        91
```

Tính độ chính xác và báo cáo chi tiết về hiệu suất mô hình.





- Nhận xét: Dựa vào biểu đồ và số liệu ta có thể thấy dùng các phương pháp tránh overfitting giúp mô hình của chúng ta sẽ ít tốn tài nguyên hơn, cải thiện độ chính xác và mô hình có thể tổng quát hóa hơn khi mà chúng ta có thể giảm tình trạng overfitting rồi

2.2.5 Cải thiện độ chính xác của hai mô hình:

Để cải thiện độ chính xác của mô hình, tiến hành làm tăng độ phức tạp của các mô hình bằng cách thêm các lớp và nơ-ron vào mô hình Neural Network để làm tăng độ phức tạp. Kết hợp kỹ thuật Voting Classifier trong Ensemble kết hợp trong cả hai mô hình Random Forest và FFNN với phương thức bỏ phiếu soft. Ngoài ra, phương pháp SMOTE trong Oversampling cũng được kết hợp để cải thiện khả năng của mô hình trong việc dự đoán các trường hợp thuộc lớp đó.

- Mô hình FFNN:

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train, y_train = sm.fit_resample(X_train, y_train)
```

Phương pháp SMOTE trong Oversampling

```
def create_ffnn_model():
    model = Sequential()
    # Tầng ẩn 1
    model.add(Dense(128, input_dim=7, activation='relu'))
    # Tầng ẩn 2
    model.add(Dense(64, activation='relu'))
    # Tầng ẩn 3
    model.add(Dense(32, activation='relu'))
    # Tầng ẩn 4
    model.add(Dense(16, activation='relu'))
    # Tầng đầu ra
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

ffnn_model = KerasClassifier(build_fn=create_ffnn_model, epochs=30, batch_size=16, verbose=0)

rf_model = RandomForestClassifier(n_estimators=100, random_state=142)

ensemble_model = VotingClassifier(estimators=[('random_forest', rf_model), ('ffnn', ffnn_model)], voting='soft')

history_ensemble = ensemble_model.fit(X_train, y_train)

y_pred = ensemble_model.predict(X_test)

print("Ensemble Model Accuracy: %.2f%%" % (accuracy_score(y_test, y_pred) * 100))
print(classification_report(y_test, y_pred))
```

Tăng độ phức tạp của mô hình FFNN

```
Ensemble Model Accuracy: 81.32%
              precision    recall  f1-score   support

           0       0.85      0.71      0.77        41
           1       0.79      0.90      0.84        50

    accuracy                           0.81        91
   macro avg       0.82      0.80      0.81        91
weighted avg       0.82      0.81      0.81        91
```

Tính độ chính xác và báo cáo chi tiết về hiệu suất mô hình.

- Mô hình RNN:

```
def create_rnn_model():
    model = Sequential()
    model.add(SimpleRNN(100, activation='relu', input_shape=(X_train.shape[1], 1), kernel_regularizer=regularizers.l1(1e-6)))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model
rnn_model = KerasClassifier(build_fn=create_rnn_model, epochs=90, batch_size=32, verbose=0)
# Create a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=142)

# Create a VotingRegressor with Random Forest and RNN models
ensemble_model = VotingClassifier(estimators=[('random_forest', rf_model), ('rnn', rnn_model)])

# Fit the ensemble model on the training data
history_ensemble = ensemble_model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = ensemble_model.predict(X_test)
y_pred = [1 if y >= 0.5 else 0 for y in y_pred]

# Print the performance metrics
print("Ensemble Model Accuracy: %.2f%%" % (accuracy_score(y_test, y_pred) * 100))
print(classification_report(y_test, y_pred))
```

Voting Classifier với Random Forest và RNN

```
Ensemble Model Accuracy: 76.92%
              precision    recall  f1-score   support

           0       0.72      0.80      0.76        41
           1       0.82      0.74      0.78        50

    accuracy                           0.77        91
   macro avg       0.77      0.77      0.77        91
weighted avg       0.77      0.77      0.77        91
```

Tính độ chính xác và báo cáo chi tiết về hiệu suất mô hình.

- Nhận xét: So sánh sau khi cải thiện với trước khi cải thiện.
    - Khi tăng độ phức tạp của mô hình lên có thể dẫn đến hiện tượng quá khớp bởi vì kích thước dữ liệu nhỏ, việc sử dụng nhiều lớp neuron sẽ khó thích nghi với bộ dữ liệu.
    - Nếu các mô hình đồng nhất hoặc quá giống nhau, Ensemble có thể không mang lại sự cải thiện đáng kể.
    - Dữ liệu đào tạo không đủ lớn khi áp dụng các phương pháp cải thiện hiệu suất có thể sẽ không có ảnh hưởng lớn.

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng việt: