TRƯỜNG ĐẠI HỌC BÁCH KHOA TPHCM KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

TRIỂN KHAI GIẢI THUẬT RANDOM FOREST VÀ ÁP DỤNG VÀO DỰ ĐOÁN BỆNH NHÂN MẮC BỆNH TIM MẠCH

Môn học: Học Máy và Ứng Dụng

GVHD: PGS.TS Dương Tuấn Anh

Sinh viên thực hiện:

Nguyễn Quang Gia Bảo 2370265

TPHCM, ngày 13 tháng 12 năm 2024

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

RF	Random Forest

1. GIỚI THIỆU RANDOM FOREST (RF)

1.1. Thuật toán Random Forest là gì

Random Forest là một thuật toán machine learning supervised learning mạnh mẽ và linh hoạt, được sử dụng cho cả classification và regression. Ví dụ, trong bài toán classification, Random Forest có thể được áp dụng để phân loại email thành spam hoặc không spam dựa trên các thuộc tính như từ khóa, số lượng liên kết, hay kích thước email. Đối với regression, Random Forest có thể được sử dụng để dự đoán giá nhà dựa trên các thuộc tính như diện tích, số phòng ngủ, và vị trí địa lý. Thuật toán này được xây dựng dựa trên tập hợp nhiều cây quyết định (được gọi là forest). Ý tưởng chính là kết hợp nhiều cây quyết định để tạo ra một mô hình dự báo tốt hơn, đồng thời giảm thiểu độ nhiễu và độ chênh lệch.

Random Forest thuộc loại ensemble learning, nơi kết hợp dự báo của nhiều mô hình để cải thiện hiệu suất. Trong bài toán classification, kết quả cuối cùng được xác định bằng voting (đa số phiếu) từ các cây. Cụ thể, mỗi cây quyết định trong rừng sẽ đưa ra một dự đoán cho lớp của dữ liệu đầu vào. Sau đó, lớp nào nhận được nhiều phiếu nhất từ các cây sẽ được chọn làm kết quả cuối cùng. Ví dụ, nếu có 100 cây trong rừng và 60 cây dự đoán rằng dữ liệu thuộc lớp A, trong khi 40 cây dự đoán thuộc lớp B, thì kết quả cuối cùng sẽ là lớp A. Cơ chế voting này giúp tăng độ chính xác và giảm thiểu rủi ro do lỗi từ các cây đơn lẻ. Trong regression, đó là trung bình của các kết quả.

1.2. Mô tả thuật toán Random Forest

Random Forest xây dựng nhiều cây quyết định độc lập từ tập dữ liệu huấn luyện bằng cách:

- Chọn ngẫu nhiên một tập con dữ liệu từ tập huấn luyện gốc (được gọi là bootstrap sampling).
- Xây dựng mỗi cây quyết định từ tập con này. Trong quá trình chia nút, chỉ xem xét một tập con ngẫu nhiên các thuộc tính (để giảm độ tương quan giữa các cây).
- Dự báo cuối cùng được xác định bằng voting (đối với classification)

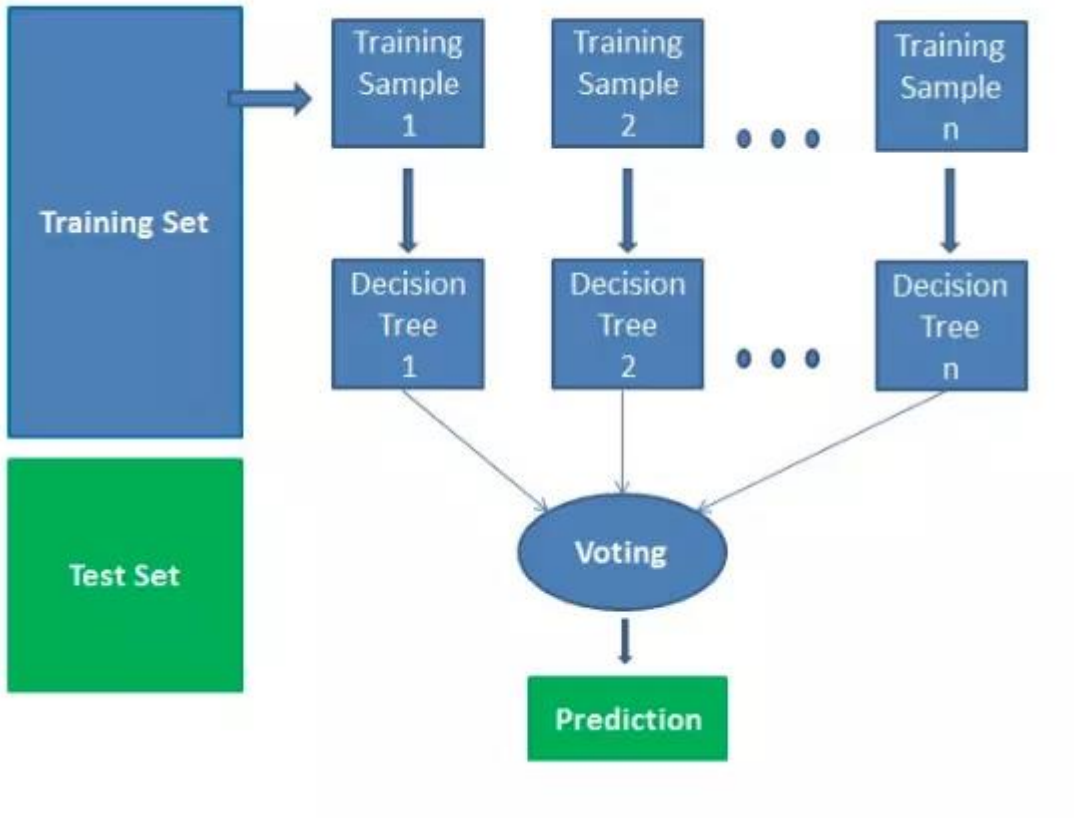
hoặc trung bình (với regression) của tất cả cây.

Các bước thực hiện Random Forest:

- Chọn số lượng cây quyết định (n_estimators).
- Thực hiện bootstrap sampling từ tập dữ liệu huấn luyện.
- Xây dựng mỗi cây quyết định với tập con dữ liệu và tập con thuộc tính.
- Kết hợp dự báo từ tất cả cây.

Để hiểu rõ random forest được dùng để phân lớp như thế nào ta xem minh hoạ dưới đây:

Trong hình sau, bắt đầu bằng việc chia nhỏ dữ liệu gốc thành nhiều tập con thông qua kỹ thuật bootstrapping. Đối với mỗi tập con, một cây quyết định được xây dựng bằng cách sử dụng một tập hợp ngẫu nhiên của các tính năng, giúp cây quyết định giảm thiểu độ lệch và tránh việc overfitting. Sau khi tất cả các cây được huấn luyện, Random Forest sẽ dự đoán bằng cách cho mỗi cây đưa ra dự đoán của mình, và kết quả cuối cùng được xác định bằng cách tổng hợp các dự đoán này, thường là thông qua việc bỏ phiếu đa số đối với các bài toán phân loại hoặc tính giá trị trung bình đối với các bài toán hồi quy. Sự đa dạng trong các cây quyết định giúp Random Forest trở nên mạnh mẽ hơn và giảm thiểu sai số tổng quát hóa so với các mô hình đơn lẻ.



Hình 1. Minh hoạ Random forest

2. ĐỘ ĐO HIỆU SUẤT

2.1. Out-of-Bag Error (OOB Error)

Vì Random Forest sử dụng bootstrap sampling, khoảng 1/3 dữ liệu huấn luyện sẽ không được sử dụng để xây dựng cây. Phần dữ liệu này (được gọi là out-of-bag data) được sử dụng để đánh giá hiệu suất mô hình mà không cần tập test riêng.

OOB Error được tính bằng cách sử dụng phần dữ liệu out-of-bag (OOB) từ bootstrap sampling. Mỗi cây quyết định trong Random Forest chỉ sử dụng một tập con dữ liệu để huấn luyện, vì vậy các mẫu không được sử dụng sẽ trở thành dữ liệu OOB. Để tính OOB Error, từng mẫu trong dữ liệu OOB được dự báo bởi các cây không sử dụng mẫu đó trong quá trình xây dựng. Tỷ lệ dự báo sai trên toàn bộ dữ liệu OOB chính là OOB Error, cung cấp một cách đánh giá hiệu suất mô hình mà không cần sử dụng tập dữ liệu kiểm tra riêng biệt.

2.2. Feature Importance

Random Forest cung cấp đánh giá tầm quan trọng của các thuộc tính dựa trên mức giảm thiểu impurity (độ bất định) tại các nút chia. Ví dụ, nếu một thuộc tính giúp giảm impurity nhiều nhất tại một số lượng lớn các nút trong toàn bộ cây của rừng, thì thuộc tính đó được coi là quan trọng. Giả sử trong một bài toán phân loại, thuộc tính "Độ tuổi" có thể thường xuyên được chọn để chia các nút và tạo ra sự phân tách rõ ràng, do đó, tầm quan trọng của "Độ tuổi" sẽ được tính toán dựa trên tổng mức giảm impurity mà nó tạo ra trong toàn bộ mô hình. Đều này giúp hiểu rõ hơn đóng góp của từng thuộc tính trong việc dự báo.

3. LỰA CHỈNH SỐ LƯỢNG CÂY VÀ THUỘC TÍNH

3.1. Số lượng cây (n_estimators)

Số lượng cây càng lớn thì kết quả dự báo càng độc lập và chính xác. Tuy nhiên, tăng số lượng cây cũng làm tăng chi phí tính toán và thời gian xử lý, đặc biệt khi làm việc với tập dữ liệu lớn. Do đó, cần cân nhắc điểm cân bằng giữa hiệu suất và chi phí. Một cách thông dụng là thử nghiệm với các giá trị khác nhau của số lượng cây và chọn giá trị tối ưu dựa trên hiệu suất mô hình và tài nguyên tính toán sẵn có. Tuy nhiên, việc tăng số cây sẽ làm tăng chi phí tính toán.

3.2. Số thuộc tính xét tại mỗi nút (max_features)

Chọn ngẫu nhiên một tập con thuộc tính để giảm tương quan giữa các cây và tăng độ độc lập của mô hình.

4. TÍNH TOÁN ĐỘ CHÍNH XÁC

Để đánh giá độ chính xác của Random Forest, ta xem số dữ liệu được dự báo đúng trong tập test và chia cho tổng số lượng dữ liệu trong tập test. Random Forest thường được đánh giá cao nhờ khả năng giảm overfitting nhờ vào việc kết hợp nhiều cây..

5. ÁP DỤNG RANDOM FOREST VÀO BÀI TOÁN DỰ ĐOÁN MẮC BỆNH TIM

5.1. Định nghĩa bài toán

Bệnh tim mạch là do các rối loạn của tim và mạch máu. Bệnh tim mạch bao gồm bệnh mạch vành (nhồi máu cơ tim), tai biến mạch máu não (đột quỵ), tăng huyết áp tăng (cao huyết áp), bệnh động mạch ngoại biên, bệnh thấp tim, bệnh tim bẩm sinh và suy tim.

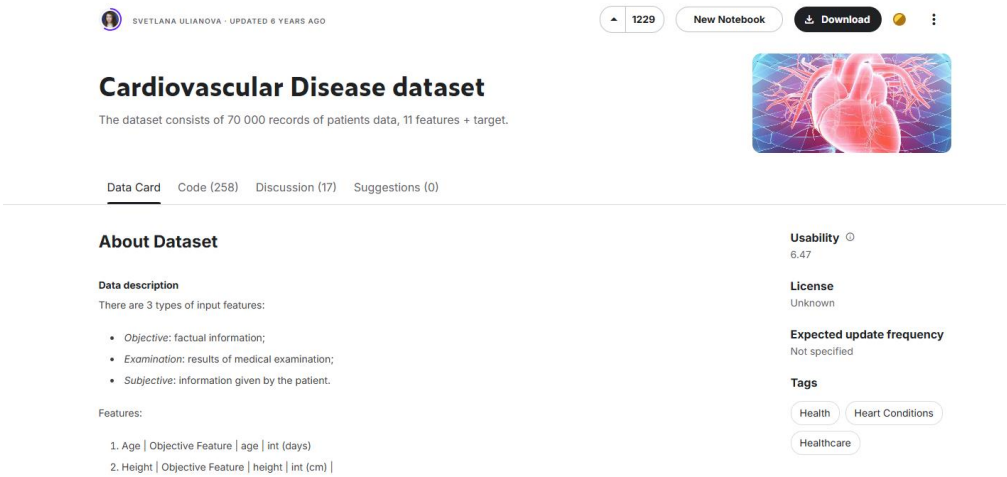
Theo WHO, bệnh tim mạch hiện là nguyên nhân hàng đầu gây tử vong trên toàn cầu. Tỷ lệ bệnh tim mạch đang ngày càng tăng cao ở các nước đang phát triển, trong đó có Việt Nam. Bệnh tim mạch là nguyên nhân cho 31% tổng số ca tử vong tại nước ta trong năm 2016, tương đương 170.000 ca. Bên cạnh việc làm giảm năng suất lao động, chi phí cho khám và chữa bệnh tim mạch cũng là gánh nặng kinh tế rất lớn.

Những dấu hiệu bệnh tim mạch thường xuất hiện thoáng qua, không rõ ràng, khiến người bệnh thường không để ý đến, cho tới khi có các dấu hiệu nặng thì đã muộn. Do đó, phát hiện và ngăn ngừa các yếu tố có ảnh hưởng lớn nhất đến bệnh tim là rất quan trọng trong việc chăm sóc sức khỏe. Sự phát triển của công nghệ ngày nay đã cho phép ta có thể ứng dụng học máy để dự đoán tình trạng của bệnh nhân nhanh hơn.

5.2. Thu thập và tiền xử lý dữ liệu

- Tên dataset: Cardiovascular Disease dataset.

- Dữ liệu do tài Svetlana Ulianova thu thập và cung cấp trên nền tảng Kaggle.



Hình 2. Nguồn dữ liệu

- Bao gồm 70,000 dòng dữ liệu và 12 cột thuộc tính.

- Giá trị bị thiếu (missing value): 0.

- Mô tả: Dữ liệu gốc thuộc về CDC – một phần chính của Hệ thống giám sát các yếu tố rủi ro hành vi (BRFSS). Đây là kết quả cuộc khảo sát thường niên qua điện thoại vào năm 2020 để thu thập dữ liệu về tình trạng sức khỏe của người dân Hoa Kỳ. Phần lớn các cột là câu hỏi được đặt ra cho người tham gia khảo sát về tình trạng sức khỏe của họ. Từ gần 300 thuộc tính, tác giả chọn ra những nhân tố chính có thể ảnh hưởng trực tiếp hoặc gián tiếp đến bệnh tim, biến đổi các biến phân loại từ dạng số sang dạng chuỗi để thuận tiện cho việc phân tích, loại bỏ các hàng bị thiếu trong bảng ghi. Dữ liệu bao gồm các thuộc tính:

  - Age | Objective Feature | age | int (days)
  - Height | Objective Feature | height | int (cm) |
  - Weight | Objective Feature | weight | float (kg) |
  - Gender | Objective Feature | gender | categorical code |

- Systolic blood pressure | Examination Feature | ap_hi | int |
- Diastolic blood pressure | Examination Feature | ap_lo | int |
- Cholesterol | Examination Feature | cholesterol | 1: normal, 2: above normal, 3: well above normal |
- Glucose | Examination Feature | gluc | 1: normal, 2: above normal, 3: well above normal |
- Smoking | Subjective Feature | smoke | binary |
- Alcohol intake | Subjective Feature | alco | binary |
- Physical activity | Subjective Feature | active | binary |
- Một thuộc tính còn lại là kết quả dự đoán bệnh tim
    - Presence
    - Absence

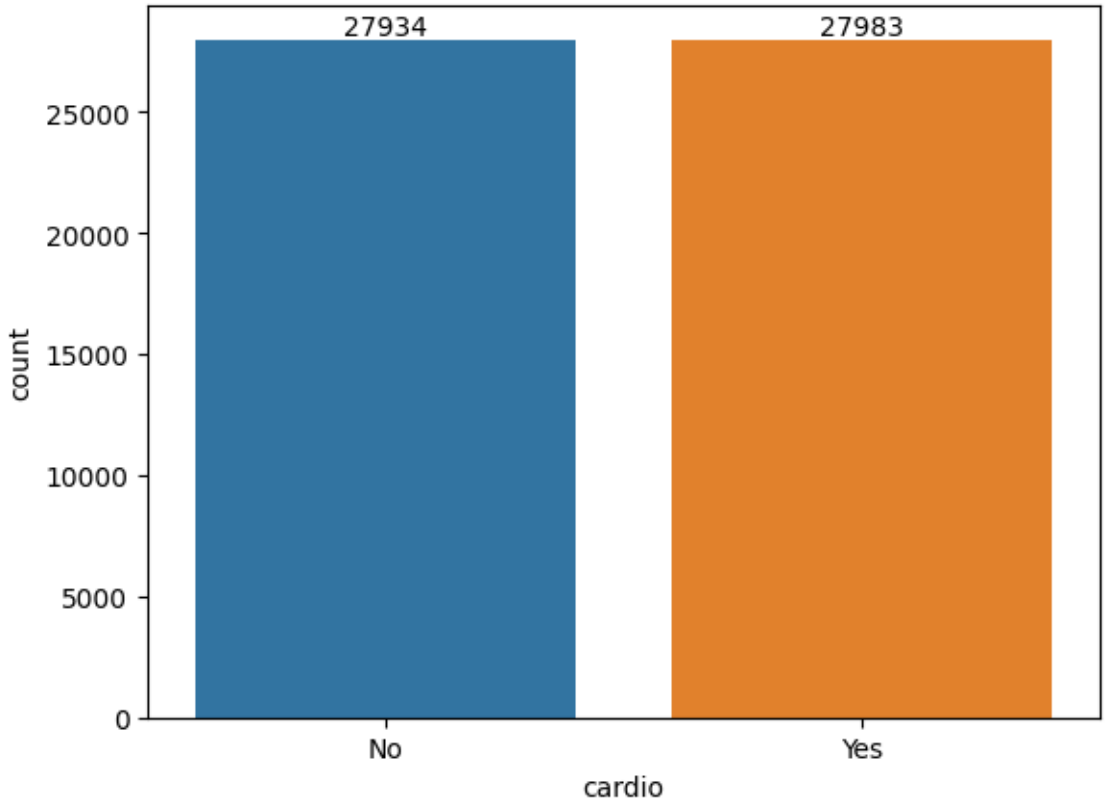
Trong tập dữ liệu heart disease, các giá trị nhỏ nhất, trung bình và lớn nhất ứng với các thuộc tính lần lượt như sau:

	Giá trị nhỏ nhất	**Giá trị lớn nhất**	**Giá trị trung bình**
Age	10798.0	23713.0	19468.9
Gender	1.0	2.0	1.3
Height	55.0	250.0	164.4
Weight	10.0	200.0	74.2
Ap hi	- 150.0	16020.0	128.8
Ap lo	-70.0	11000.0	96.6
Cholesterol	1.0	3.0	1.4
Glu	1.0	3.0	1.2

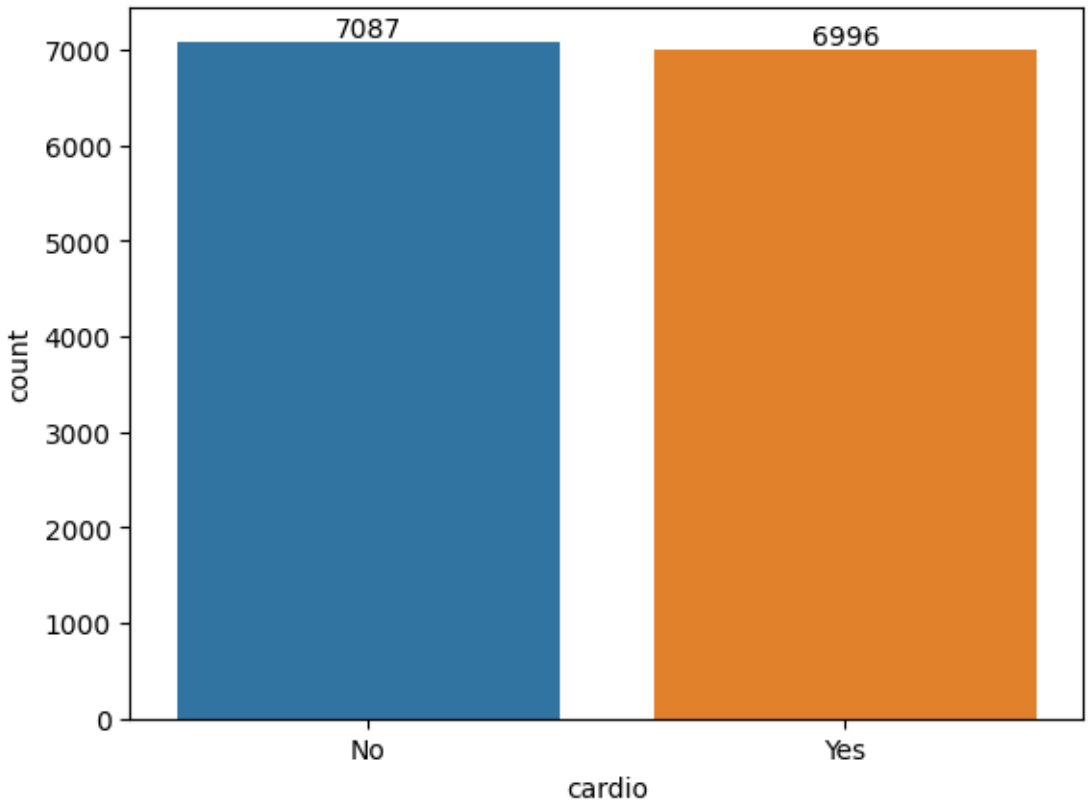
Smoke	0.0	1.0	0.1
Alco	0.0	1.0	0.8
Active	0.0	1.0	0.5

Bảng 1. Giá trị trong tập dữ liệu Heart Disease

Tỷ lệ phân chia cho khả năng mắc bệnh tim là 50%



Hình 3 . Tỉ lệ phân chia khả năng mắc bệnh tim trong tập train



Hình 4 . Tỉ lệ phân chia khả năng mắc bệnh tim trong tập test

5.2.1. Làm sạch dữ liệu (data cleaning)

Thiếu giá trị: khi xảy ra sự thiếu thông tin ở một thuộc tính nào đó trong một bộ dòng của bộ dữ liệu thu thập, khi mà tính đảm bảo số bộ dòng chia đều cho cả 2 trường hợp mắc bệnh và không mắc bệnh (~50%) và số lượng bộ dòng thiếu là ít ta có thể áp dụng phương pháp loại bỏ. Trường hợp cần điền giá trị thiếu, ta có thể áp dụng các giá trị trung bình bên trên.

5.2.2. Chọn lọc dữ liệu (data selection)

Tích hợp và dư thừa dữ liệu: do dữ liệu của heart disease được thu thập từ một nguồn duy nhất nên việc tích hợp là không cần thiết trong trường hợp này, các thuộc tính của dữ liệu độc lập nhau, không có mối quan hệ tương quan nào, các thuộc tính đã được rút gọn chọn lọc nên không cần việc phân tích dư thừa dữ liệu đối với tập dữ liệu này.

5.2.3. Chuyển đổi dữ liệu (data transformation)

Với tập dữ liệu, chuẩn hoá dữ liệu về khoảng giá trị [0, 1] được sử dụng để đảm bảo việc cân bằng dữ liệu, không có thuộc tính mang giá trị lớn sẽ ảnh

hưởng lớn đến đến các thuộc tính còn lại.

5.3 Triển khai mô hình dự đoán bệnh tim

heart disease dataset là một bộ dữ liệu trung bình. Bộ dữ liệu này bao gồm thông tin của 2 trường hợp mắc bệnh và không mắc bệnh.

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Hình 5. Giá trị mẫu

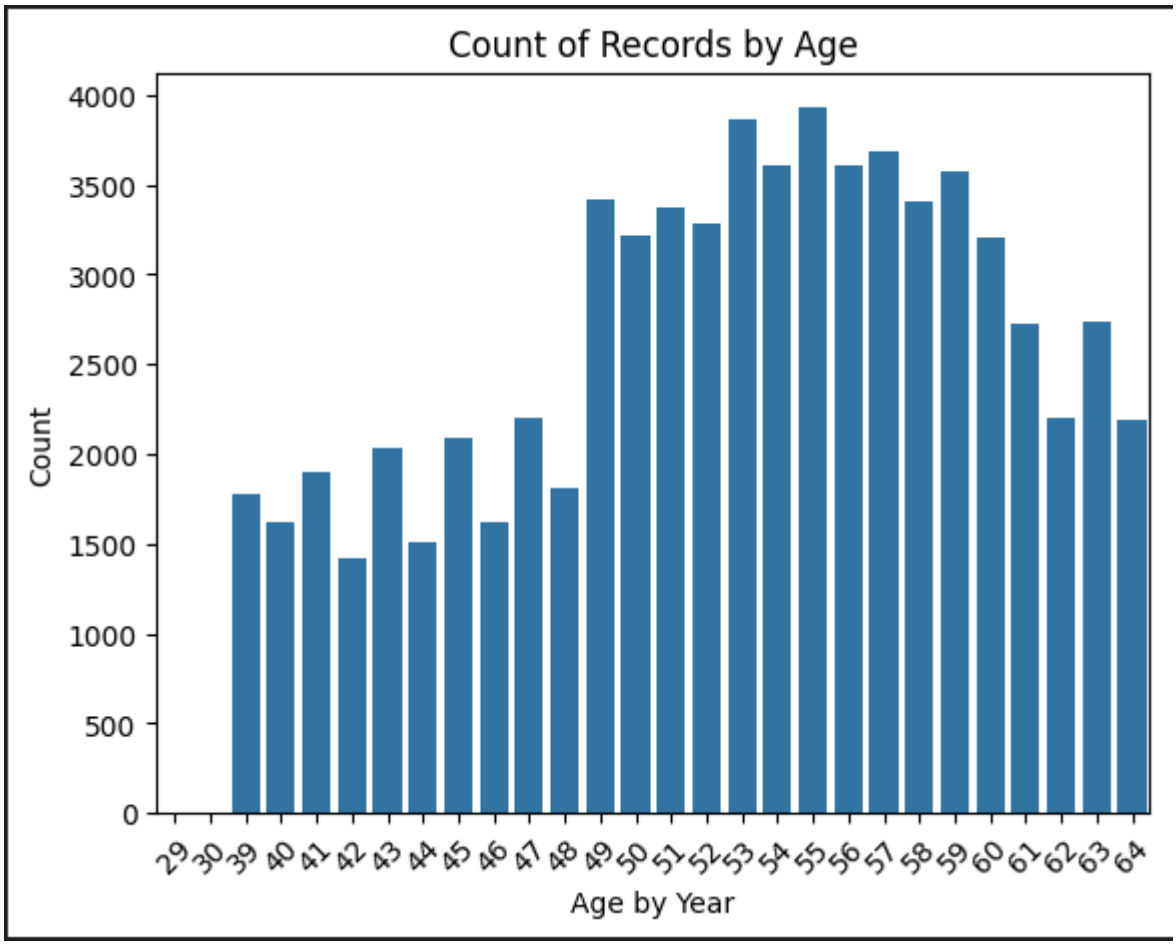
Dataset không có các dữ liệu missing hay null. Có 70,000 mẫu dữ liệu trong bộ dataset. Phân bố dữ liệu trong dataset là đều không có hiện tượng mất cân bằng dữ liệu.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           70000 non-null  int64
 1   age          70000 non-null  int64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
 13  age_by_year  70000 non-null  int64
dtypes: float64(1), int64(13)
memory usage: 7.5 MB
```

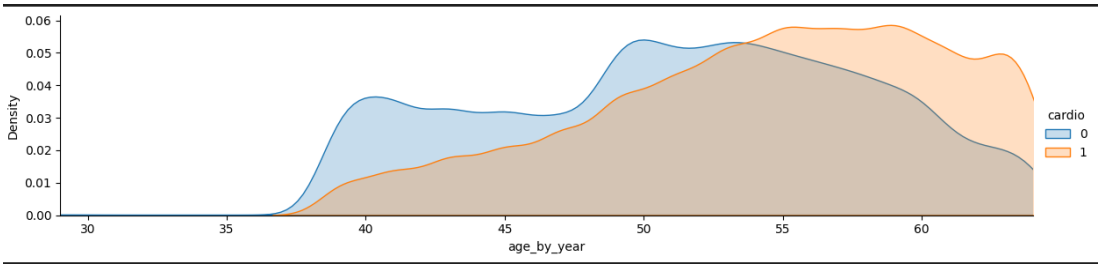
Hình 6. Thông tin dữ liệu

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	age_by_year
count	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0	70000.0
mean	49972.4	19468.9	1.3	164.4	74.2	128.8	96.6	1.4	1.2	0.1	0.1	0.8	0.5	52.8
std	28851.3	2467.3	0.5	8.2	14.4	154.0	188.5	0.7	0.6	0.3	0.2	0.4	0.5	6.8
min	0.0	10798.0	1.0	55.0	10.0	-150.0	-70.0	1.0	1.0	0.0	0.0	0.0	0.0	29.0
25%	25006.8	17664.0	1.0	159.0	65.0	120.0	80.0	1.0	1.0	0.0	0.0	1.0	0.0	48.0
50%	50001.5	19703.0	1.0	165.0	72.0	120.0	80.0	1.0	1.0	0.0	0.0	1.0	0.0	53.0
75%	74889.2	21327.0	2.0	170.0	82.0	140.0	90.0	2.0	1.0	0.0	0.0	1.0	1.0	58.0
max	99999.0	23713.0	2.0	250.0	200.0	16020.0	11000.0	3.0	3.0	1.0	1.0	1.0	1.0	64.0

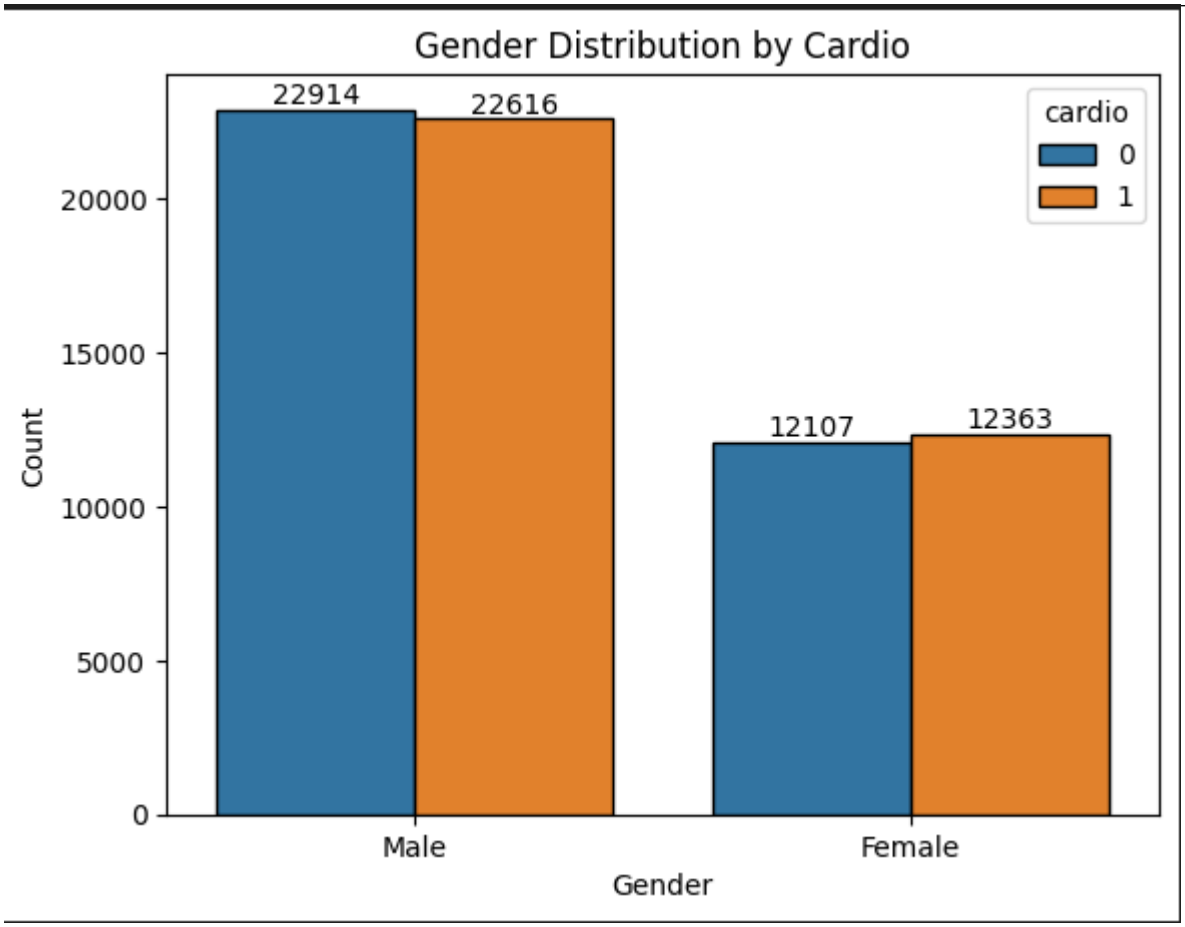
Bảng 2. Giá trị đặc trưng chênh lệch



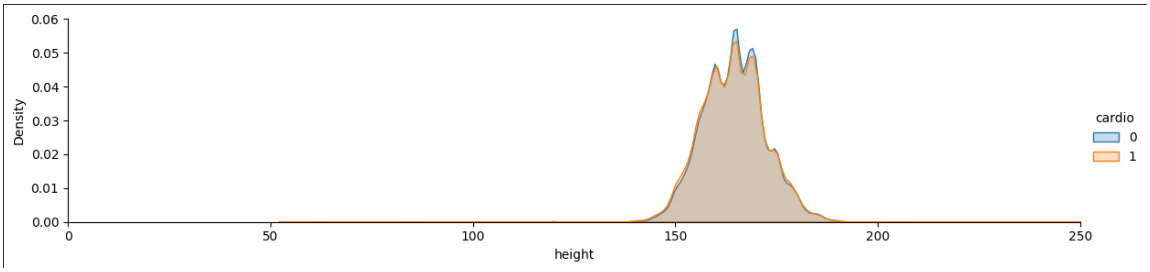
Hình 7. Sự tương quan phân bố dữ liệu theo thuộc tính age



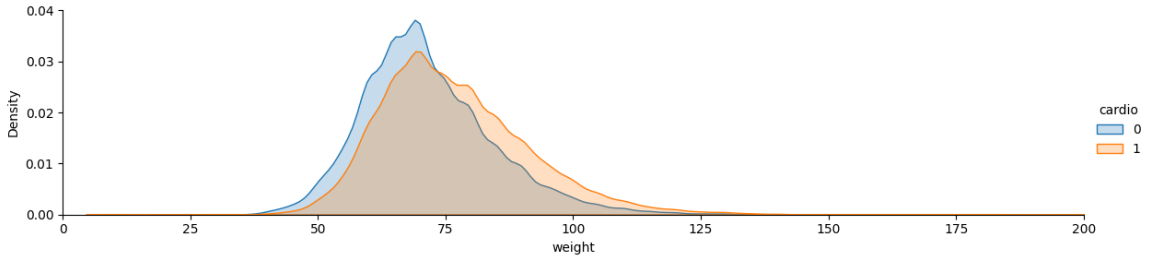
Hình 8. Sự tương quan miền phân bố dữ liệu theo thuộc tính age



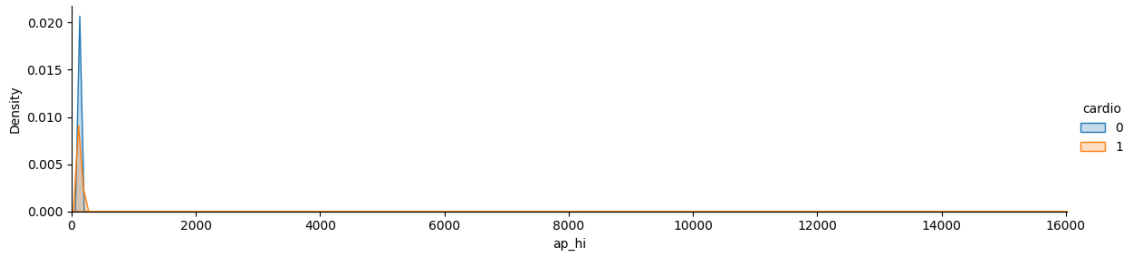
Hình 9. Sự tương quan phân bố dữ liệu theo thuộc tính gender



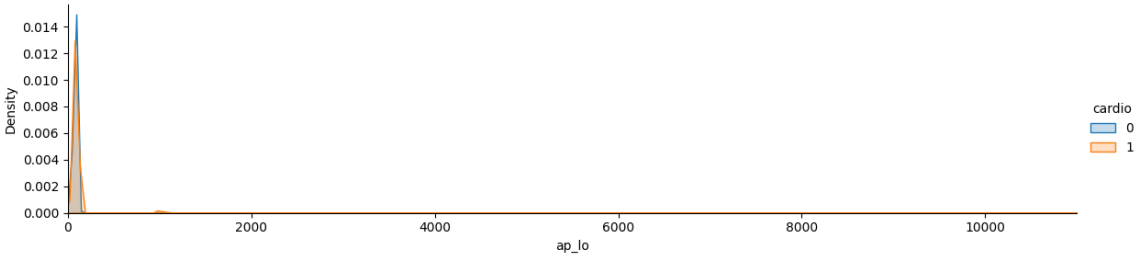
Hình 10. Sự tương quan phân bố dữ liệu theo thuộc tính height



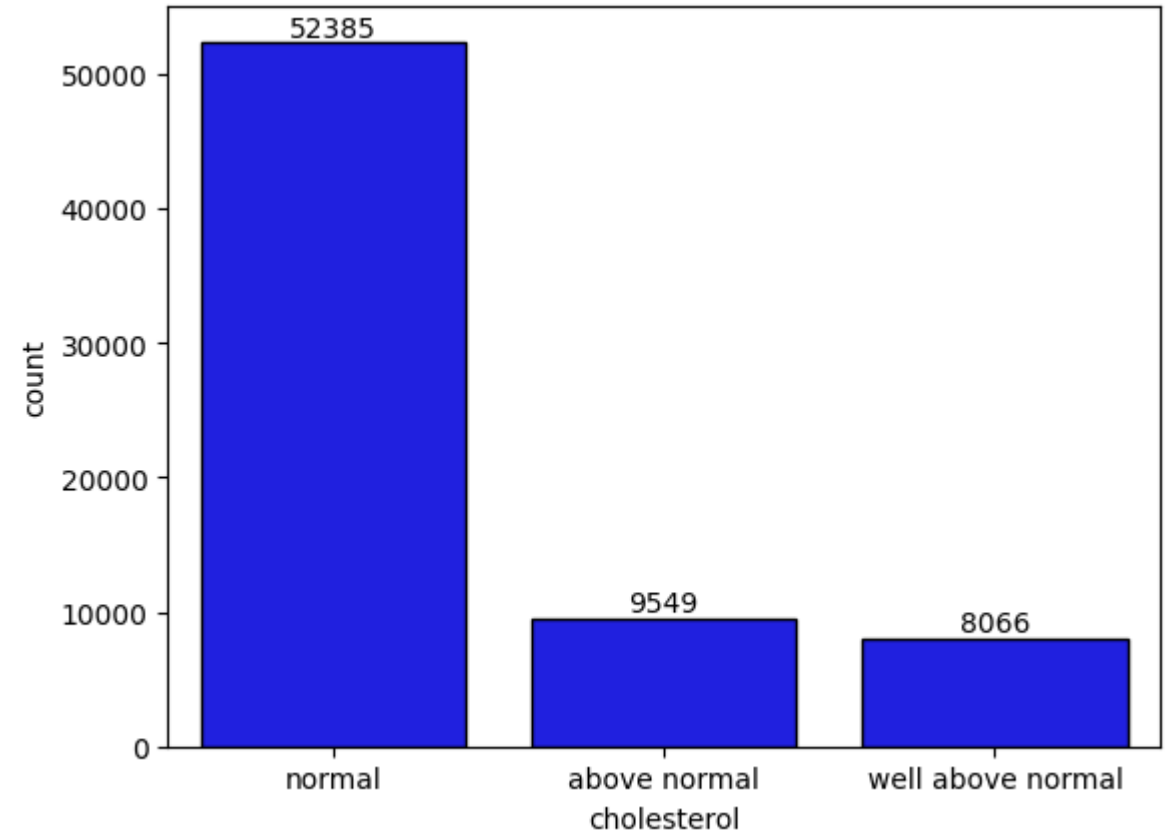
Hình 11. Sự tương quan phân bố dữ liệu theo thuộc tính weight



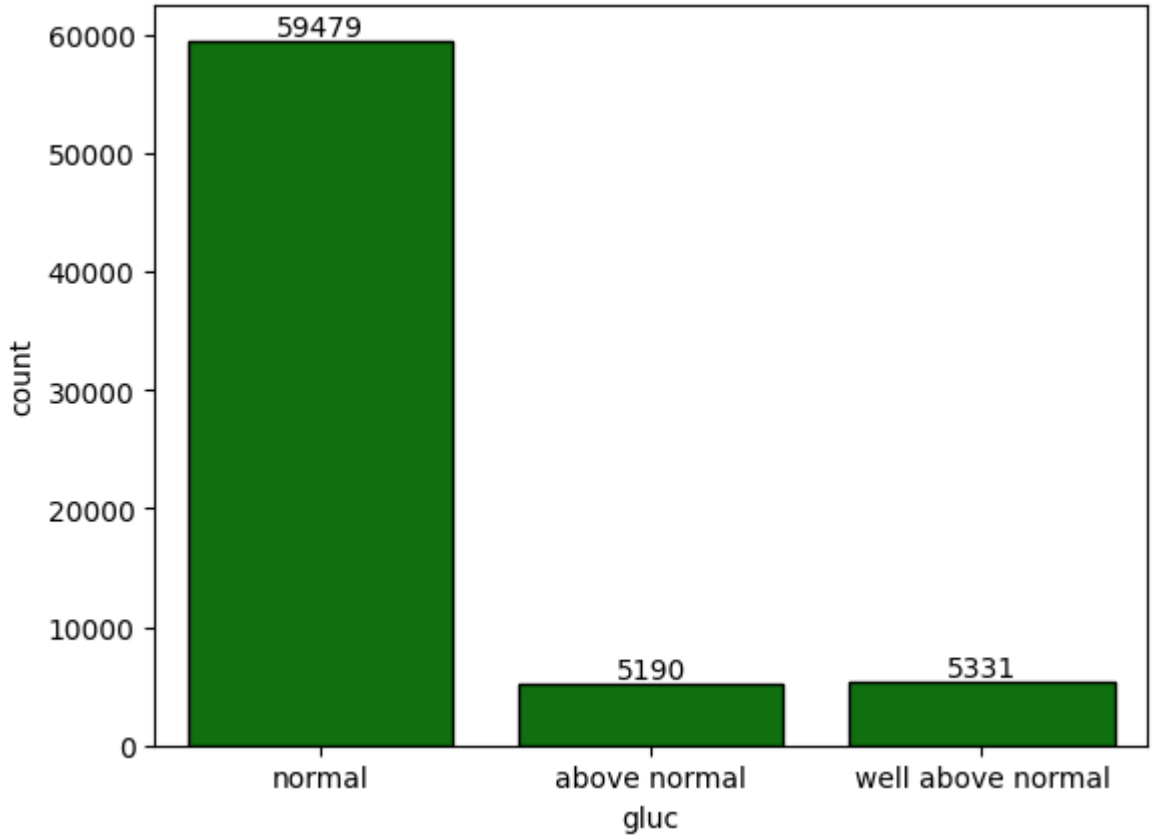
Hình 12. Sự tương quan phân bố dữ liệu theo thuộc tính ap_hi



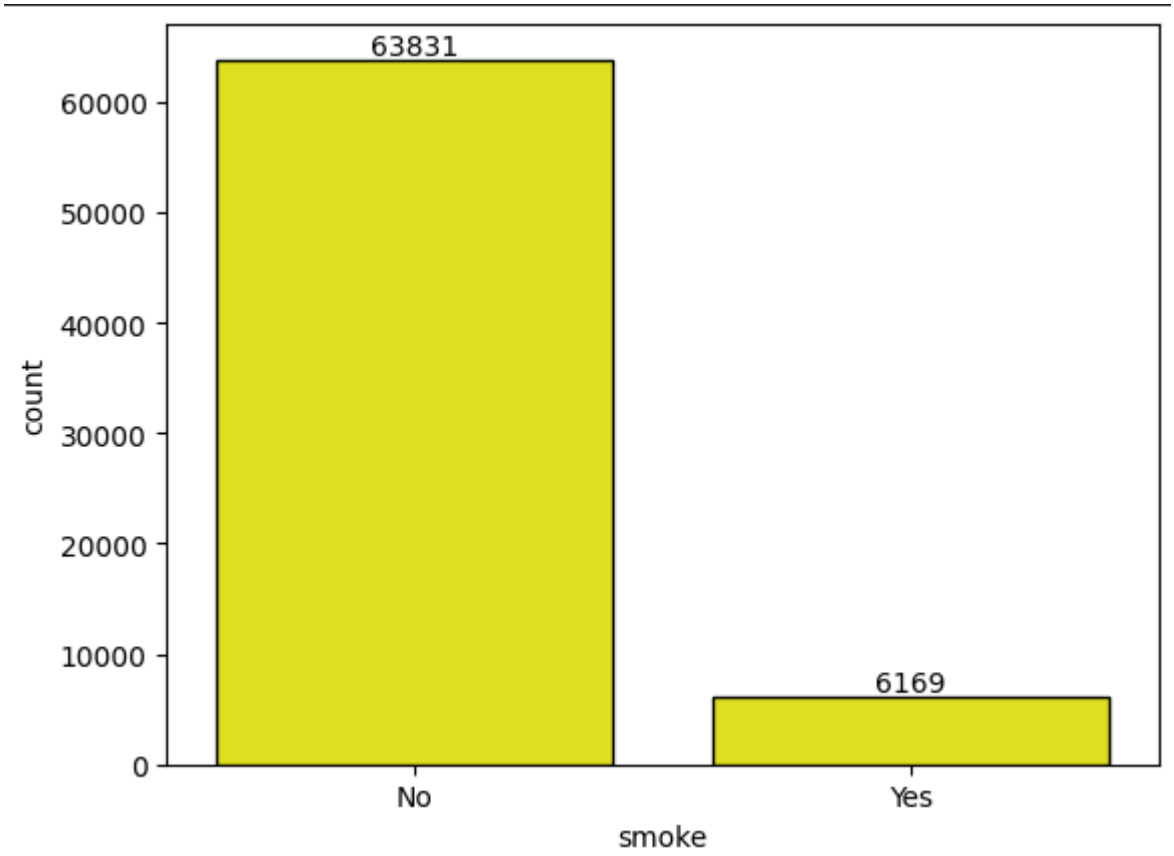
Hình 13. Sự tương quan phân bố dữ liệu theo thuộc tính ap_lo



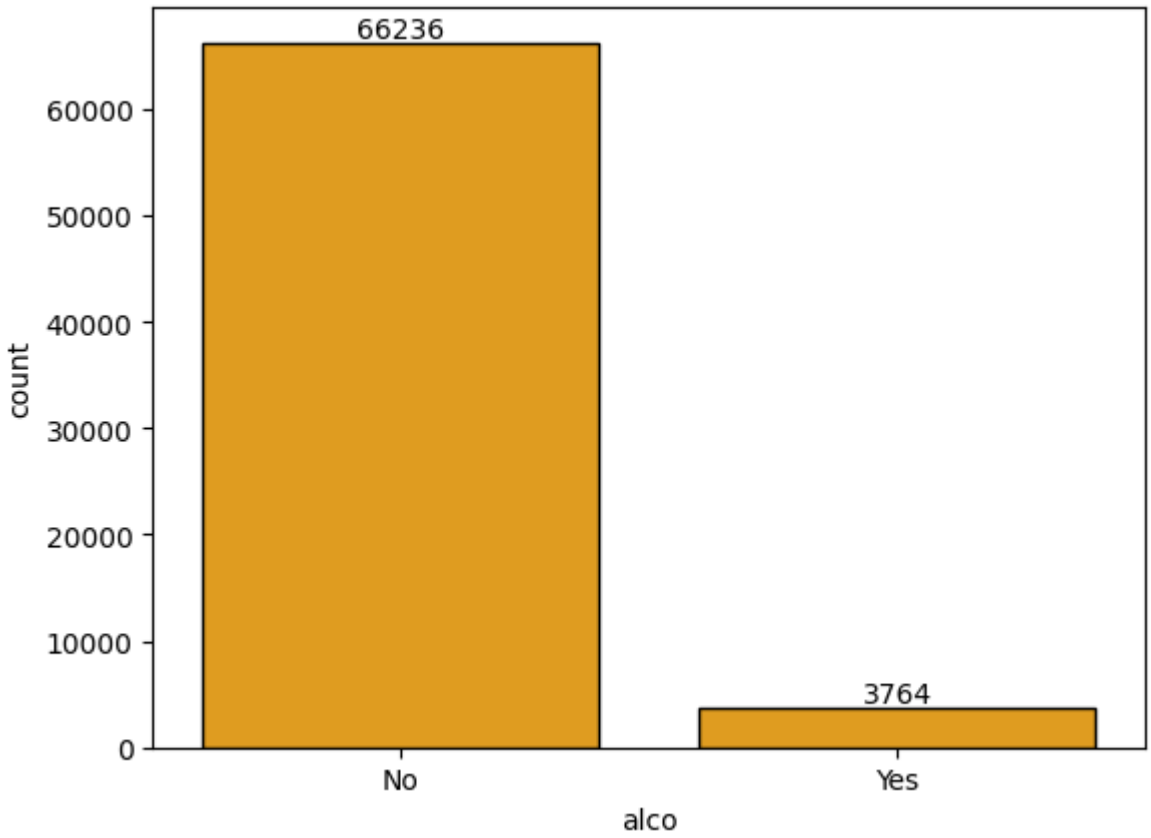
Hình 14. Sự tương quan phân bố dữ liệu theo thuộc tính cholesterol



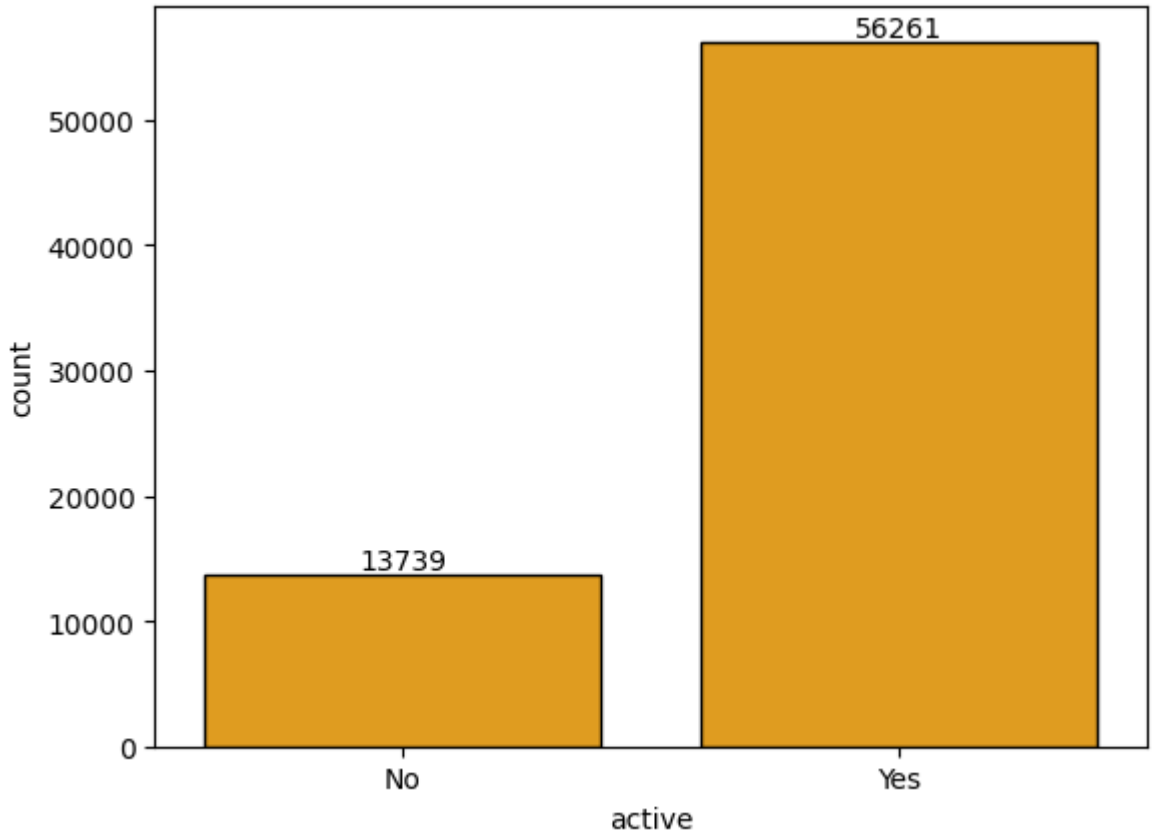
Hình 15. Sự tương quan phân bố dữ liệu theo thuộc tính gluc



Hình 16. Sự tương quan phân bố dữ liệu theo thuộc tính smoke



Hình 17. Sự tương quan phân bố dữ liệu theo thuộc tính alco



Hình 18. Sự tương quan phân bố dữ liệu theo thuộc tính active

Tiền xử lí dữ liệu

Khả năng mắc bệnh (nhãn) là một dữ liệu rời rạc vì thế cần chuyển đổi về dạng số để có thể làm việc được với mô hình phân lớp Random Forest triển khai

- No tương ứng với 0
- Yes tương ứng với 1

5.2.4. Cách thức đo độ chính xác của mô hình

Bước 1: Phân dữ liệu ngẫu nhiên theo hai tập huấn luyện và kiểm tra

Training DataFrame has 56000 rows

Test DataFrame has 14000 rows

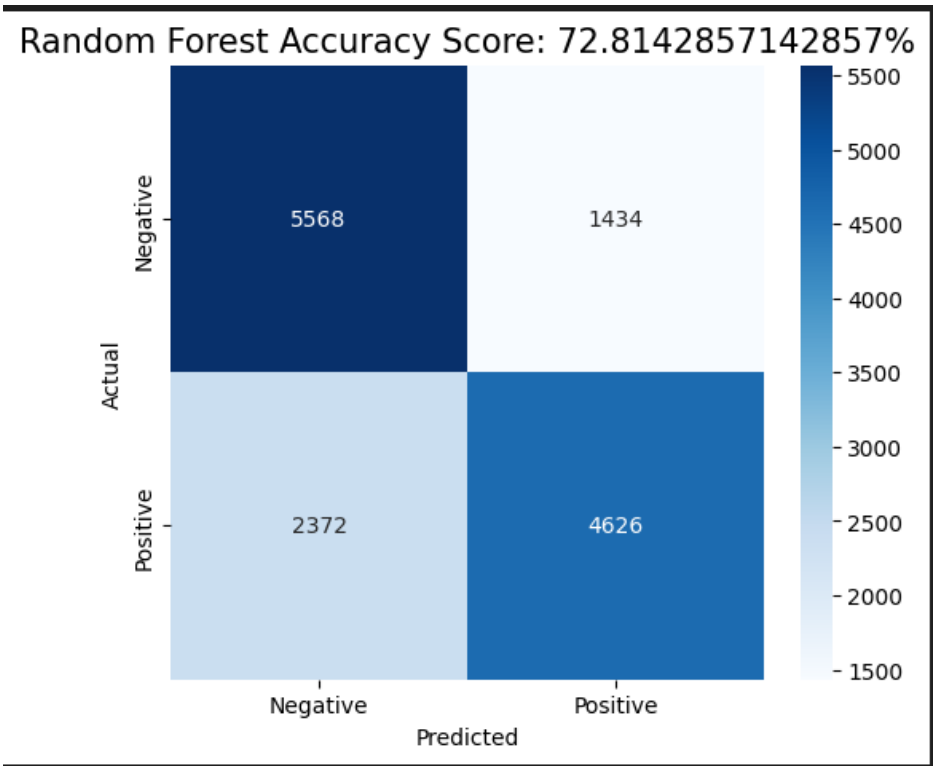
Bước 2: Đánh giá mô hình qua tập kiểm tra

```
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.80      0.75      7002
           1       0.76      0.66      0.71      6998

    accuracy                           0.73     14000
   macro avg       0.73      0.73      0.73     14000
weighted avg       0.73      0.73      0.73     14000
```

Hình 19. Classification Report

```
True Positives (TP): 4626
False Positives (FP): 1434
True Negatives (TN): 5568
False Negatives (FN): 2372
Accuracy (calculated): 72.81%
```

Hình 20. Result



Hình 21. Confusion Matrix

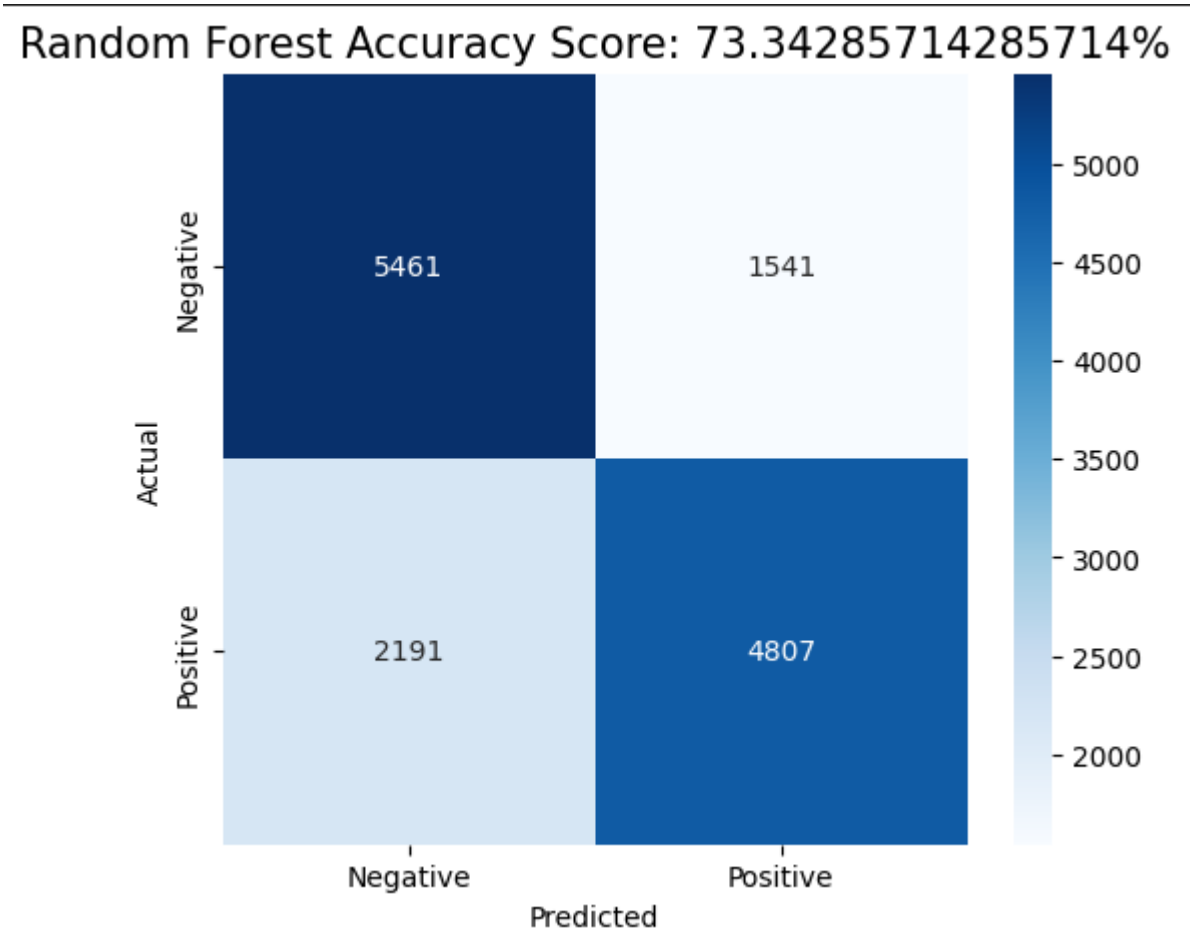
6. SO SÁNH KẾT QUẢ VỚI THƯ VIỆN SKLEARN

	precision	recall	f1-score	support
0	0.71	0.78	0.75	7002
1	0.76	0.69	0.72	6998
accuracy			0.73	14000
macro avg	0.74	0.73	0.73	14000
weighted avg	0.74	0.73	0.73	14000

Hình 22. Classification Report theo thư viện sklearn

```
True Positives (TP): 4807
False Positives (FP): 1541
True Negatives (TN): 5461
False Negatives (FN): 2191
Accuracy (calculated): 73.34%
```

Hình 23. Result theo thư viện sklearn



Hình 24. Confusion theo thư viện sklearn

7. KẾT LUẬN

Ưu điểm của thuật toán Random Forest

- Hiệu suất cao và mạnh mẽ: Random Forest giảm thiểu vấn đề overfitting mà các cây quyết định đơn lẻ thường gặp phải, hoạt động tốt trên cả bài toán phân loại (classification) và hồi quy (regression).
- Xử lý dữ liệu phi tuyến tính: có thể nắm bắt các mối quan hệ phi tuyến tính phức tạp giữa các biến đầu vào và đầu ra.
- Tính tổng quát hóa tốt: do sử dụng kỹ thuật bagging (bootstrap aggregating) và lấy mẫu ngẫu nhiên, nó có khả năng dự đoán tốt trên dữ liệu chưa từng thấy (dữ liệu kiểm tra).
- Khả năng xử lý dữ liệu đa dạng: Random Forest hoạt động tốt với cả dữ liệu có biến định tính (categorical) và biến liên tục (continuous).
- Tự động xử lý giá trị bị thiếu: Random Forest có thể xử lý một số giá trị bị thiếu (missing values) mà không cần tiền xử lý phức tạp.
- Đánh giá tầm quan trọng của đặc trưng: Thuật toán cung cấp thông tin về mức độ quan trọng của từng biến đầu vào đối với kết quả dự đoán, giúp chọn lọc đặc trưng tốt hơn.
- Ít yêu cầu về tham số: Random Forest không cần tinh chỉnh tham số quá phức tạp như một số mô hình khác (ví dụ, SVM hoặc neural networks)..

Nhược điểm của thuật toán Random Forest

- Thời gian tính toán lớn: Random Forest cần xây dựng nhiều cây quyết định (đặc biệt với dữ liệu lớn), do đó tiêu tốn tài nguyên và thời gian.
- Khó giải thích: Mặc dù Random Forest hoạt động tốt, nhưng nó thiếu tính trực quan so với một cây quyết định đơn lẻ, dẫn đến khó hiểu hơn cho người dùng.
- Không tối ưu cho bài toán dữ liệu hiếm (sparse data): Hiệu suất có thể

không tốt bằng một số thuật toán khác, như gradient boosting, trên dữ liệu rất hiếm hoặc không cân bằng.

- Không tốt với dữ liệu có quá nhiều đặc trưng nhiễu: Nếu dữ liệu có nhiều biến không liên quan, Random Forest có thể chọn phải một số đặc trưng nhiễu, ảnh hưởng đến độ chính xác của mô hình.
- Tốn bộ nhớ: Do phải lưu trữ nhiều cây quyết định và dữ liệu huấn luyện cho từng cây con, Random Forest yêu cầu bộ nhớ lớn hơn so với các thuật toán đơn giản như Logistic Regression.
- Khó xử lý dữ liệu cao chiều và số lượng lớp lớn: Khi số chiều hoặc số lớp (classes) tăng, thời gian xây dựng mô hình cũng tăng theo và có thể ảnh hưởng đến hiệu suất dự đoán..

8. MÃ NGUỒN

Giải thuật KNN được nhóm triển khai trên môi trường Google Colab.

Link mã nguồn: ***NguyenBaoZ/Random-Forest-Heart-Prediction***

8.1. Cài đặt các thư viện cần thiết

Một số thư viện cần thiết được dùng trong bài gồm:

- Thư viện pandas dùng để load dữ liệu dưới dạng bảng (csv).
- Thư viện numpy dùng để hỗ trợ tính toán ma trận.
- Thư viện sklearn dùng để kiểm tra so sánh kết quả với giải thuật hiện thực.
- Thư viện matplotlib và seaborn cho trực quan dữ liệu.
- Thư viện joblib để thực thi song song để tối ưu thuật toán

```
import pandas as pd
import numpy as np
from pandas import DataFrame,Series
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.metrics import classification_report
from collections import Counter
from joblib import Parallel, delayed
import time
%matplotlib inline
```

8.2. Xây dựng lớp mô hình Random Forest

Nhóm đã xây dựng một lớp *CustomDecisionTree* và *CustomRandomForest*

Đối với lớp *CustomDecisionTree*

- __init__(self, max_depth=None, min_samples_split=2):

```
def __init__(self, max_depth=None, min_samples_split=2):
    self.max_depth = max_depth
    self.min_samples_split = min_samples_split
    self.tree = None
```

  - max_depth: Độ sâu tối đa của cây (nếu không đặt giới hạn, cây sẽ tiếp

tục chia nhánh đến khi đạt điều kiện dừng khác).

- min_samples_split: Số lượng mẫu nhỏ nhất cần thiết để tiếp tục chia nhánh.
- tree: Lưu cấu trúc của cây sau khi được huấn luyện.

- fit(self, X, y, depth=0):

```
def fit(self, X, y, depth=0):
    y = y.flatten()
    if len(set(y)) == 1 or len(y) < self.min_samples_split or (self.max_depth is not None and depth >= self.max_depth):
        return Counter(y).most_common(1)[0][0]
    best_split = self._best_split(X, y)
    if not best_split:
        return Counter(y).most_common(1)[0][0]
    left_tree = self.fit(X[best_split['left_indices']], y[best_split['left_indices']], depth + 1)
    right_tree = self.fit(X[best_split['right_indices']], y[best_split['right_indices']], depth + 1)
    return {'split': best_split, 'left': left_tree, 'right': right_tree}
```

Huấn luyện cây quyết định:

- Điều kiện dừng chia nhánh:
  - Tất cả nhãn trong tập y là giống nhau.
  - Số lượng mẫu ít hơn min_samples_split.
  - Độ sâu cây đạt giới hạn max_depth. Nếu bất kỳ điều kiện nào thỏa mãn, trả về nhãn phổ biến nhất trong y.
- Tìm điểm chia tốt nhất:
  - Gọi hàm _best_split để xác định cách chia tốt nhất tại nút hiện tại.
  - Nếu không thể chia thêm, trả về nhãn phổ biến nhất.
- Đệ quy: Áp dụng đệ quy để huấn luyện cây con bên trái và phải, sử dụng các tập con được chia bởi điểm chia.

Kết quả: Trả về cấu trúc cây quyết định dưới dạng từ điển với các nhánh split, left, và right.

- predict(self, X):

```
def predict(self, X):
    return np.array([self._predict_single(x, self.tree) for x in X])
```

Dự đoán nhãn cho một tập dữ liệu X: Áp dụng hàm _predict_single để dự đoán cho từng mẫu x trong X.

- _predict_single(self, x, tree):

```python
def _predict_single(self, x, tree):
    if isinstance(tree, dict):
        if x[tree['split']['feature_index']] <= tree['split']['threshold']:
            return self._predict_single(x, tree['left'])
        else:
            return self._predict_single(x, tree['right'])
    else:
        return tree
```

Dự đoán nhãn cho một mẫu duy nhất x dựa trên cấu trúc cây tree:

1. Nếu nút hiện tại là một từ điển (leaf): So sánh giá trị của đặc trưng với threshold:
    - Nếu nhỏ hơn hoặc bằng, đi tiếp cây con bên trái.
    - Nếu lớn hơn, đi tiếp cây con bên phải.
2. Nếu nút hiện tại là leaf (label), trả về nhãn đó.

- _best_split(self, X, y):

```python
def _best_split(self, X, y):
    m, n = X.shape
    if m <= 1:
        return None
    best_gini = 1.0 - sum((np.sum(y == c) / m) ** 2 for c in np.unique(y))
    best_split = None
    for feature_index in range(n):
        thresholds = np.unique(X[:, feature_index])
        for threshold in thresholds:
            left_indices = X[:, feature_index] <= threshold
            right_indices = X[:, feature_index] > threshold
            if len(y[left_indices]) == 0 or len(y[right_indices]) == 0:
                continue
            gini_left = 1.0 - sum((np.sum(y[left_indices] == c) / len(y[left_indices])) ** 2 for c in np.unique(y))
            gini_right = 1.0 - sum((np.sum(y[right_indices] == c) / len(y[right_indices])) ** 2 for c in np.unique(y))
            gini = (len(y[left_indices]) * gini_left + len(y[right_indices]) * gini_right) / m
            if gini < best_gini:
                best_gini = gini
                best_split = {
                    'feature_index': feature_index,
                    'threshold': threshold,
                    'left_indices': left_indices,
                    'right_indices': right_indices
                }
    return best_split
```

Tìm cách chia dữ liệu tối ưu tại một nút dựa trên chỉ số Gini:

1. **Chỉ số Gini**:
    - Đo lường mức độ không đồng nhất trong tập dữ liệu.
    - Gini càng thấp, dữ liệu càng đồng nhất.
2. **Lặp qua các đặc trưng và giá trị phân ngưỡng**:
    - Thử nghiệm chia tập dữ liệu tại từng giá trị phân ngưỡng (threshold) của mỗi đặc trưng.
    - Tính chỉ số Gini cho hai tập con sau khi chia (trái và phải).
    - Lưu lại cách chia có chỉ số Gini nhỏ nhất.
    -

3. **Kết quả**:
    - Trả về cách chia tốt nhất, bao gồm: đặc trưng, ngưỡng, và các chỉ số của tập con.

Đối với lớp *CustomRandomForest*

- __init__(self, n_estimators=100, max_features=None, max_depth=None, min_samples_split=2, bootstrap=1.0, n_jobs=-1):

```
def __init__(self, n_estimators=100, max_features=None, max_depth=None, min_samples_split=2, bootstrap=1.0, n_jobs=-1):
    self.n_estimators = n_estimators
    self.max_features = max_features
    self.max_depth = max_depth
    self.min_samples_split = min_samples_split
    self.bootstrap = bootstrap
    self.n_jobs = n_jobs
    self.trees = []
```

    - n_estimators: Số lượng cây trong rừng.
    - max_features: Số lượng đặc trưng tối đa được sử dụng tại mỗi lần phân chia (nếu không đặt, sử dụng toàn bộ đặc trưng).
    - max_depth: Độ sâu tối đa của mỗi cây.
    - min_samples_split: Số lượng mẫu tối thiểu để chia nút.
    - bootstrap: Tỷ lệ mẫu bootstrap để huấn luyện từng cây (giá trị từ 0 đến 1, biểu thị phần trăm mẫu).
    - n_jobs: Số luồng thực thi song song (nếu -1, sử dụng toàn bộ CPU).
    - trees: Danh sách chứa các cây được huấn luyện trong rừng..

- fit(self, X, y):

```
def fit(self, X, y):
    X = np.array(X)
    y = np.array(y).flatten()
    n_samples, n_features = X.shape
    n_bootstrap_samples = int(self.bootstrap * n_samples)
    self.trees = Parallel(n_jobs=self.n_jobs)(delayed(self._fit_tree)(X, y, n_bootstrap_samples, n_features) for _ in range(self.n_estimators))
```

(variable) _: int

Huấn luyện random forest:

- Chuyển đổi dữ liệu:
    - Chuyển X và y thành mảng NumPy để dễ dàng thao tác.
    - Tính số mẫu (n_samples) và số đặc trưng (n_features).
- Tính số mẫu bootstrap:
    - n_bootstrap_samples được xác định dựa trên tỷ lệ bootstrap.
- Huấn luyện song song: Sử dụng joblib.Parallel để huấn luyện nhiều cây cùng lúc, mỗi cây được xây dựng bởi hàm _fit_tree.

Kết quả: Lưu danh sách các cây (và các đặc trưng tương ứng) vào self.trees.

- _fit_tree(self, X, y, n_bootstrap_samples, n_features):

```
def _fit_tree(self, X, y, n_bootstrap_samples, n_features):
    indices = np.random.choice(range(len(X)), size=n_bootstrap_samples, replace=True)
    X_subset = X[indices]
    y_subset = y[indices]
    features = np.random.choice(n_features, size=self.max_features, replace=False) if self.max_features else range(n_features)
    tree = CustomDecisionTree(max_depth=self.max_depth, min_samples_split=self.min_samples_split)
    tree.tree = tree.fit(X_subset[:, features], y_subset)
    return (tree, features)
```

Huấn luyện 1 tree đơn lẻ:

- Tạo tập bootstrap:
  - Chọn ngẫu nhiên n_bootstrap_samples từ tập dữ liệu ban đầu (cho phép trùng lặp).
- Chọn đặc trưng ngẫu nhiên:
  - Nếu max_features được đặt, chọn ngẫu nhiên số lượng đặc trưng tương ứng
  - Nếu không, sử dụng toàn bộ đặc trưng.
- Huấn luyện cây:
  - Khởi tạo một cây CustomDecisionTree với các tham số max_depth và min_samples_split
  - Huấn luyện cây với tập con của dữ liệu (X_subset và y_subset).

Kết quả: Trả về cặp (tree, features), trong đó tree là cây được huấn luyện và features là danh sách đặc trưng được sử dụng..

- predict(self, X):

```
def predict(self, X):
    X = np.array(X)
    tree_preds = Parallel(n_jobs=self.n_jobs)(delayed(self._predict_tree)(tree, features, X) for tree, features in self.trees)
    tree_preds = np.swapaxes(np.array(tree_preds), 0, 1)
    final_preds = [Counter(tree_preds[i]).most_common(1)[0][0] for i in range(tree_preds.shape[0])]
    return np.array(final_preds)
```

- Dự đoán song song:
  - Sử dụng joblib.Parallel để gọi hàm _predict_tree cho từng cây trong self.trees.
  - Tính số mẫu (n_samples) và số đặc trưng (n_features).
- Tổng hợp dự đoán:
  - Tập hợp dự đoán từ tất cả các cây (dạng ma trận: số mẫu × số cây).
  - Với mỗi mẫu, lấy nhãn được dự đoán nhiều nhất (bỏ phiếu đa số).
- Kết quả: Trả về mảng các nhãn dự đoán cuối cùng.

- _predict_tree(self, tree, features, X):

```
def _predict_tree(self, tree, features, X):
    return tree.predict(X[:, features])
```

  - Sử dụng chỉ những đặc trưng đã chọn khi huấn luyện cây (features).
  - Gọi hàm predict của cây đó để dự đoán nhãn.

8.3. Áp dụng mô hình Random Forest vào bộ dữ liệu Iris

8.3.1. Tải bộ dữ liệu

```
heart_df = pd.read_csv('cardio_train.csv', delimiter=";")
heart_df.head()
```

8.3.2. Tiền xử lý dữ liệu

- *Quy đổi tuổi:*

```
heart_df['age_by_year'] = (heart_df['age'] // 365)
```

- *Chia tập dữ liệu thành tập train và test tỉ lệ test_size = 20%*

```
# Splitting the DataFrame into training and test sets
heart_df_train, heart_df_test = train_test_split(heart_df, test_size=0.2, random_state=17)

# Show the number of records in each split
print(f"Training DataFrame has {len(heart_df_train)} rows")
print(f"Test DataFrame has {len(heart_df_test)} rows")
```

8.3.3. Xây dựng mô hình Random Forest

```
model2 = CustomRandomForest(n_estimators=100, max_features=5, max_depth=10)
```

```
# Measure the time to fit the model
start_time = time.time()
model2.fit(X_train, Y_train)
fit_time = time.time() - start_time
print(f"Time to fit the model: {fit_time:.2f} seconds")
```

```
Time to fit the model: 19.22 seconds
```

```
# Predictions
start_time = time.time()
predictions_custom = model2.predict(X_test)
predict_time = time.time() - start_time
print(f"Time to predict: {predict_time:.2f} seconds")
```

```
Time to predict: 133.75 seconds
```

```
# Evaluation
print("Classification Report:")
print(classification_report(Y_test, predictions_custom))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.80      0.75      7002
           1       0.76      0.66      0.71      6998

    accuracy                           0.73     14000
   macro avg       0.73      0.73      0.73     14000
weighted avg       0.73      0.73      0.73     14000
```

```python
# Compute confusion matrix
cm = confusion_matrix(y_test_data, predictions_custom)

# Extract TP, FP, TN, FN
TN, FP, FN, TP = cm.ravel()

print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")

# Compute accuracy for verification
accuracy = (TP + TN) / (TP + FP + TN + FN)
print(f"Accuracy (calculated): {accuracy * 100:.2f}%")
```

```
True Positives (TP): 4626
False Positives (FP): 1434
True Negatives (TN): 5568
False Negatives (FN): 2372
Accuracy (calculated): 72.81%
```

```python
# Compute confusion matrix
cm = confusion_matrix(y_test_data, predictions_custom)

# Extract TP, FP, TN, FN for reference (optional)
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]
TP = cm[1, 1]

# Compute metrics
accuracy = accuracy_score(y_test_data, predictions_custom) * 100  # Multiply by 100 for percentage
precision = precision_score(y_test_data, predictions_custom, average='binary') * 100
recall = recall_score(y_test_data, predictions_custom, average='binary') * 100
f1 = f1_score(y_test_data, predictions_custom, average='binary') * 100

# Print metrics
print(f'Accuracy: {accuracy:.2f}%')
print(f'Precision: {precision:.2f}%')
print(f'Recall: {recall:.2f}%')
print(f'F1-score: {f1:.2f}%')
```

[39]

```
Accuracy: 72.81%
Precision: 76.34%
Recall: 66.10%
F1-score: 70.85%
```

8.4. So sánh với kết quả của thư viện sklearn

```
model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=17)
model.fit(x_training_data,y_training_data)
predictions = model.predict(x_test_data)
print(x_test_data.shape)
```

```
(14000, 12)
```

```
print(classification_report(y_test_data, predictions))
```

	precision	recall	f1-score	support
0	0.71	0.78	0.75	7002
1	0.76	0.69	0.72	6998
accuracy			0.73	14000
macro avg	0.74	0.73	0.73	14000
weighted avg	0.74	0.73	0.73	14000

```python
# Compute confusion matrix
cm = confusion_matrix(y_test_data, predictions)

# Extract TP, FP, TN, FN
TN, FP, FN, TP = cm.ravel()

print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")

# Compute accuracy for verification
accuracy = (TP + TN) / (TP + FP + TN + FN)
print(f"Accuracy (calculated): {accuracy * 100:.2f}%")
```

[27]

```
True Positives (TP): 4807
False Positives (FP): 1541
True Negatives (TN): 5461
False Negatives (FN): 2191
Accuracy (calculated): 73.34%
```

```python
# Compute confusion matrix
cm = confusion_matrix(y_test_data, predictions)

# Extract TP, FP, TN, FN for reference (optional)
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]
TP = cm[1, 1]

# Compute metrics
accuracy = accuracy_score(y_test_data, predictions) * 100  # Multiply by 100 for percentage
precision = precision_score(y_test_data, predictions, average='binary') * 100
recall = recall_score(y_test_data, predictions, average='binary') * 100
f1 = f1_score(y_test_data, predictions, average='binary') * 100

# Print metrics
print(f'Accuracy: {accuracy:.2f}%')
print(f'Precision: {precision:.2f}%')
print(f'Recall: {recall:.2f}%')
print(f'F1-score: {f1:.2f}%')
```

[28]

```
Accuracy: 73.34%
Precision: 75.72%
Recall: 68.69%
F1-score: 72.04%
```

- **Nhận xét:**

Kết quả hiện thực của gần giống với kết quả của thư viện sklearn.

9. TÀI LIỆU THAM KHẢO

[1] PGS.TS Dương Tuấn Anh, Chương 5 – Decision trees, Học Máy và Ứng Dụng, Khoa Khoa Học và Kỹ Thuật Máy Tính, Đại Học Bách Khoa TPHCM.

[2] Vũ Hữu Tiệp, Blog Machine Learning Cơ Bản, https://machinelearningcoban.com/2017/01/08/knn/

[3] RandomForestClassifier — scikit-learn 1.5.2 documentation

[4] How to Develop a Random Forest Ensemble in Python - MachineLearningMastery.com