

AI Course

Capstone Project

Final Report

For students (instructor review required)

©2023 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of this document.

This document is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this document other than the curriculum of Samsung Innovation Campus, you must receive written consent from copyright holder.

CHUẨN ĐOÁN BỆNH TIỂU ĐƯỜNG SỬ DỤNG MỘT SỐ MÔ HÌNH AI

< Date (DD/MM/YY) >

Giảng viên hướng dẫn: Trần Hùng Cường

Nhóm 4

Nguyễn Long Vũ –HAUI-AI1- SIC2100

Lăng Nhật Minh-HAUI-AI1- SIC2105

Hán Chí Vũ –HAUI-AI1-SIC2101

Mục lục

MỞ ĐẦU	5
Chương 1. Giới thiệu	6
1.1 Thông tin chung	6
1.2 Động lực và mục tiêu	6
1.3 Thành viên và phân công vai trò	6
1.4 Lịch trình và các mốc thời gian quan trọng	6
Chương 2. Thực hiện dự án.....	8
2.1 Thu thập dữ liệu	8
2.2 Phương pháp.....	9
2.3 Quy trình làm việc	10
2.4 Sơ đồ hệ thống.....	10
Chương 3. Kết quả	11
3.1 Phân tích dữ liệu (exploratory data analysis -EDA)	11
3.1.1. Pregnancies.....	12
3.1.2. Glucose	14
3.1.3. BloodPressure.....	16
3.1.4. SkinThickness.....	17
3.1.5. Insulin.....	18
3.1.6. BMI.....	19
3.1.7. DiabetesPedigreeFunction	21
3.1.8. Age.....	22

3.1.9. Outcome	23
3.1.10. Ma trận tương quan	24
3.2 Tiền xử lý dữ liệu	25
3.2.1. Dữ liệu ban đầu	25
3.2.2. Xử lý giá trị bị thiếu	28
3.2.3. Cân bằng dữ liệu	30
3.3 Mô hình hóa	33
3.3.1. Logistics	33
3.3.2. Random Forest	38
3.3.3. Mạng nơ ron nhân tạo.....	42
3.4 Giao diện người dùng	50
3.5 Thử nghiệm và cải tiến	54
3.5.1. So sánh các mô hình	54
3.5.2. Cải tiến mô hình	56
3.5.3. Thử nghiệm	65
Chương 4. Tác động dự kiến	72
4.1 Thành tựu và lợi ích.....	72
4.2 Những cải tiến trong tương lai	74
Chương 5. Đánh giá và nhận xét của thành viên nhóm	76
Chương 6. Giảng viên nhận xét.....	77
TÀI LIỆU THAM KHẢO	78

MỞ ĐẦU

Trong thời đại công nghệ 4.0, ứng dụng trí tuệ nhân tạo (AI) và học máy (Machine Learning) đang ngày càng phát triển mạnh mẽ, mang lại những đột phá to lớn trong nhiều lĩnh vực, đặc biệt là y học. Việc sử dụng các mô hình học máy để dự đoán và chẩn đoán bệnh tật không chỉ giúp cải thiện độ chính xác trong chẩn đoán mà còn giảm thiểu thời gian và chi phí trong quá trình điều trị.

Dự án của nhóm 4 tập trung vào việc xây dựng các mô hình học máy để dự đoán bệnh tiểu đường - một trong những căn bệnh mãn tính phổ biến nhất trên thế giới hiện nay. Bằng cách áp dụng các mô hình Logistic Regression, Random Forest và Mạng Nơ ron nhân tạo, nhóm 4 mong muốn tạo ra một công cụ hỗ trợ đắc lực cho các chuyên gia y tế trong việc phát hiện sớm căn bệnh này.

Báo cáo này sẽ trình bày chi tiết quá trình thu thập dữ liệu, xây dựng mô hình, thử nghiệm, và cải tiến các mô hình học máy để đạt được kết quả dự đoán tốt nhất. Hy vọng rằng những kết quả đạt được sẽ không chỉ thể hiện sự nỗ lực của nhóm mà còn đóng góp một phần nhỏ vào công cuộc ứng dụng công nghệ hiện đại trong y học.

Chương 1. Giới thiệu

1.1 Thông tin chung

Bệnh tiểu đường là một trong những thách thức lớn nhất đối với sức khỏe toàn cầu hiện nay. Với tốc độ gia tăng không ngừng, bệnh tiểu đường đã trở thành một vấn đề cấp bách, ảnh hưởng sâu sắc đến chất lượng cuộc sống của hàng triệu người. Bệnh này không chỉ gây ra các biến chứng nguy hiểm như bệnh tim mạch, tổn thương thận, mù lòa, mà còn làm tăng nguy cơ tử vong ở các bệnh nhân mắc bệnh. Chính vì vậy, việc nghiên cứu và phát triển các phương pháp chẩn đoán sớm, chính xác và hiệu quả bệnh tiểu đường là một nhiệm vụ quan trọng đối với cả cộng đồng y tế và khoa học.

1.2 Động lực và mục tiêu

Việc tìm ra phương pháp chẩn đoán chính xác và hiệu quả nhất cho bệnh tiểu đường không chỉ giúp giảm tải áp lực cho các hệ thống y tế mà còn nâng cao chất lượng chăm sóc sức khỏe cho bệnh nhân. Mục tiêu của báo cáo này là đánh giá và so sánh hiệu suất của ba mô hình học máy - hồi quy tuyến tính, mạng nơron nhân tạo và Random Forest - trong việc dự đoán bệnh tiểu đường. Thông qua việc phân tích và so sánh kết quả, báo cáo sẽ xác định mô hình nào có khả năng đưa ra dự đoán chính xác nhất. Từ đó, có thể đưa ra đề xuất về việc áp dụng mô hình phù hợp trong thực tiễn y tế, góp phần vào việc nâng cao chất lượng chẩn đoán và điều trị bệnh tiểu đường.

1.3 Thành viên và phân công vai trò

Lăng Nhật Minh: Tìm hiểu và áp dụng mô hình mạng nơ ron nhân tạo.

Hán Chí Vũ: Tìm hiểu và phát triển mô hình random forest.

Nguyễn Long Vũ: Tìm hiểu và phát triển mô hình hồi quy logicstic.

1.4 Lịch trình và các mốc thời gian quan trọng

25/7/2024 – 30/7/2024: Lựa chọn đề tài và tìm hiểu về các mô hình machine learning và deep learning.

1/8/2024 – 10/8/2024: Phân công nhiệm vụ, các thành viên tìm hiểu học tập về

mô hình đã được giao. Tổ chức họp thảo luận về đề tài trên google meet

11/8/2024 – 15/8/2024: Thử nghiệm các mô hình và thực hiện so sánh giữa các mô hình.

15/8/2024 đến nay: Tổng hợp kết quả, hoàn thành báo cáo và chuẩn bị bài thuyết trình.

Chương 2. Thực hiện dự án

2.1 Thu thập dữ liệu

Nguồn dữ liệu:

Bộ dữ liệu về tiểu đường này ban đầu đến từ Viện Quốc gia về Bệnh tiểu đường, Tiêu hóa và Thận. Mục tiêu là dự đoán dựa trên các phép đo chẩn đoán xem bệnh nhân có bị tiểu đường hay không.

Bộ dữ liệu này có một hạn chế đó là tất cả bệnh nhân ở đây đều là phụ nữ ít nhất 21 tuổi có nguồn gốc là người da đỏ Pima.

Mô tả dữ liệu:

- Pregnancies: Số lần mang thai
- Glucose: Nồng độ glucose huyết tương sau 2 giờ trong xét nghiệm dung nạp glucose đường uống.
- BloodPressure: Huyết áp tâm trương (mm Hg)
- SkinThickness: Độ dày nếp gấp da cơ tam đầu (mm)
- Insulin: Insulin huyết thanh 2 giờ (mu U/ml)
- BMI: Chỉ số khối cơ thể (cân nặng tính bằng kg/ (chiều cao tính bằng m)²)
- DiabetesPedigreeFunction: Chỉ số truyền bệnh tiểu đường, phản ánh mức độ di truyền của bệnh.
- Age: Tuổi (năm)
- Outcome: Biến lớp (0 hoặc 1)

Xử lý dữ liệu:

- Xử lý giá trị bị thiếu: thay thế giá trị bị thiếu bằng giá trị xuất hiện nhiều nhất.
- Xử lý dữ liệu không hợp lý: loại bỏ hoặc thay thế các giá trị không hợp lý trong dữ liệu,
- Chuẩn hóa dữ liệu: Dữ liệu với đầu ra là 1 (mắc bệnh) và 0(không mắc bệnh) là hợp lý và không cần chuẩn hóa.

2.2 Phương pháp

Giới thiệu mô hình Logistic Regression:

Một mô hình hồi quy sử dụng hàm sigmoid để dự đoán xác suất của một lớp (ở đây là bệnh tiểu đường).

Ưu điểm: Đơn giản, dễ giải thích, hiệu quả với các bài toán phân loại nhị phân.

Nhược điểm: Có thể không hoạt động tốt với dữ liệu phi tuyến tính và dữ liệu lớn.

Giới thiệu mô hình Mạng Nơ-ron Nhân tạo (ANN):

Mô tả cơ bản về cấu trúc của một mạng nơ-ron, các tầng ẩn, hàm kích hoạt, và cách huấn luyện mạng.

Ưu điểm: Khả năng xử lý tốt dữ liệu phi tuyến tính và có khả năng học các mối quan hệ phức tạp.

Nhược điểm: Cần nhiều dữ liệu để huấn luyện, đòi hỏi tài nguyên tính toán cao, và khó giải thích.

Giới thiệu mô hình Random Forest:

Là một tập hợp (ensemble) của nhiều cây quyết định, mỗi cây được xây dựng trên một tập con khác nhau của dữ liệu.

Ưu điểm: Khả năng xử lý tốt với dữ liệu phức tạp, giảm overfitting, và cung cấp ước tính độ quan trọng của các đặc trưng.

Nhược điểm: Thời gian huấn luyện và dự đoán có thể dài, và khó giải thích so với các mô hình đơn giản.

Lý do chọn các mô hình này:

- Logistic Regression là một mô hình cơ bản, dễ hiểu, là tiêu chuẩn vàng cho nhiều bài toán phân loại.
- Mạng Nơ-ron Nhân tạo được chọn vì khả năng học được các mối quan hệ phi tuyến tính trong dữ liệu.
- Random Forest được chọn vì khả năng xử lý tốt dữ liệu phức tạp và sự cân bằng giữa độ chính xác và tính linh hoạt.

2.3 Quy trình làm việc

Bước 1: Chia tách dữ liệu

Dữ liệu được chia thành hai tập: tập huấn luyện (training set) và tập kiểm tra (test set). Bạn có thể nêu rõ tỷ lệ chia (ví dụ: 80%-20%).

Bước 2: Tiền xử lý

Các bước chuẩn bị dữ liệu như đã mô tả trong phần 2.1: xử lý giá trị bị thiếu, chuẩn hóa hoặc chuẩn hóa dữ liệu, mã hóa các biến phân loại nếu có.

Bước 3: Huấn luyện mô hình

- Logistic Regression: Mô tả các bước huấn luyện Logistic Regression, bao gồm việc chọn tham số (solver, max_iter, v.v.), và sử dụng hàm mất mát (loss function) nào.
- Mạng Nơ-ron Nhân tạo: Mô tả kiến trúc của mạng (số lượng tầng, số lượng nơ-ron mỗi tầng), hàm kích hoạt, phương pháp tối ưu (optimizer), số lượng epoch, batch size.
- Random Forest: Mô tả các tham số quan trọng của mô hình như số lượng cây (n_estimators), độ sâu tối đa của cây (max_depth), và cách chọn tham số này.

Bước 4: Đánh giá mô hình

Sử dụng các chỉ số: Độ chính xác (accuracy), Độ nhạy (recall), Độ đặc hiệu (specificity), Độ đo F1 (F1-score).

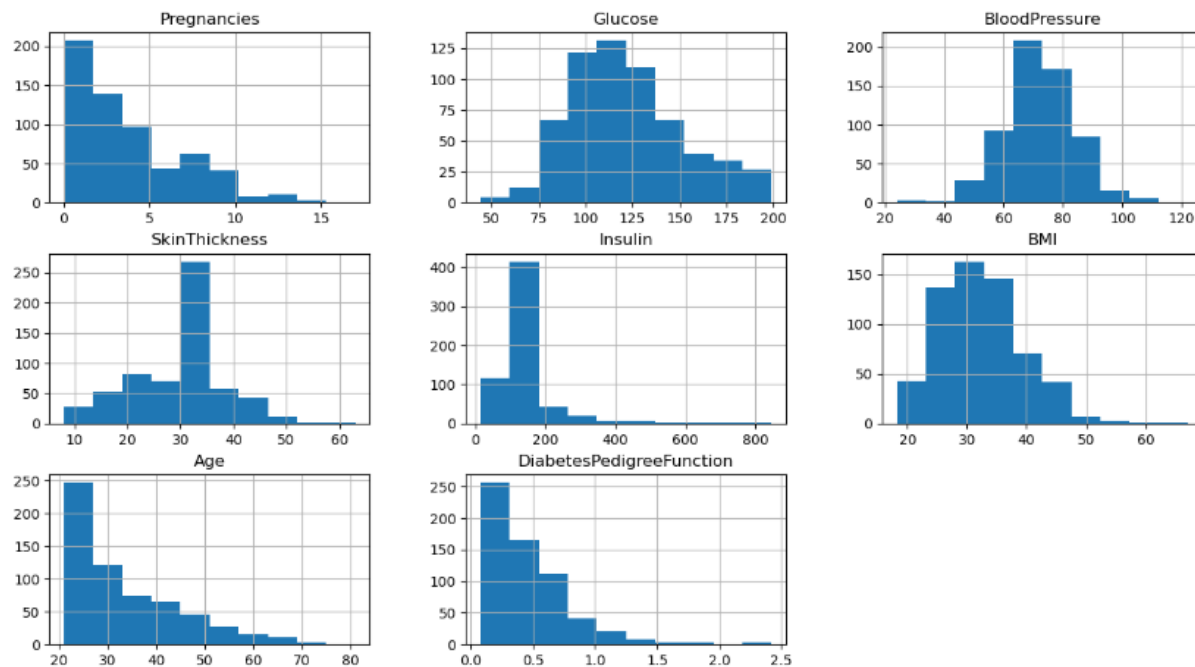
Kết quả: So sánh kết quả của ba mô hình dựa trên các chỉ số đã chọn. Trình bày kết quả bằng biểu đồ.

Bước 5: Tối ưu hóa và điều chỉnh mô hình

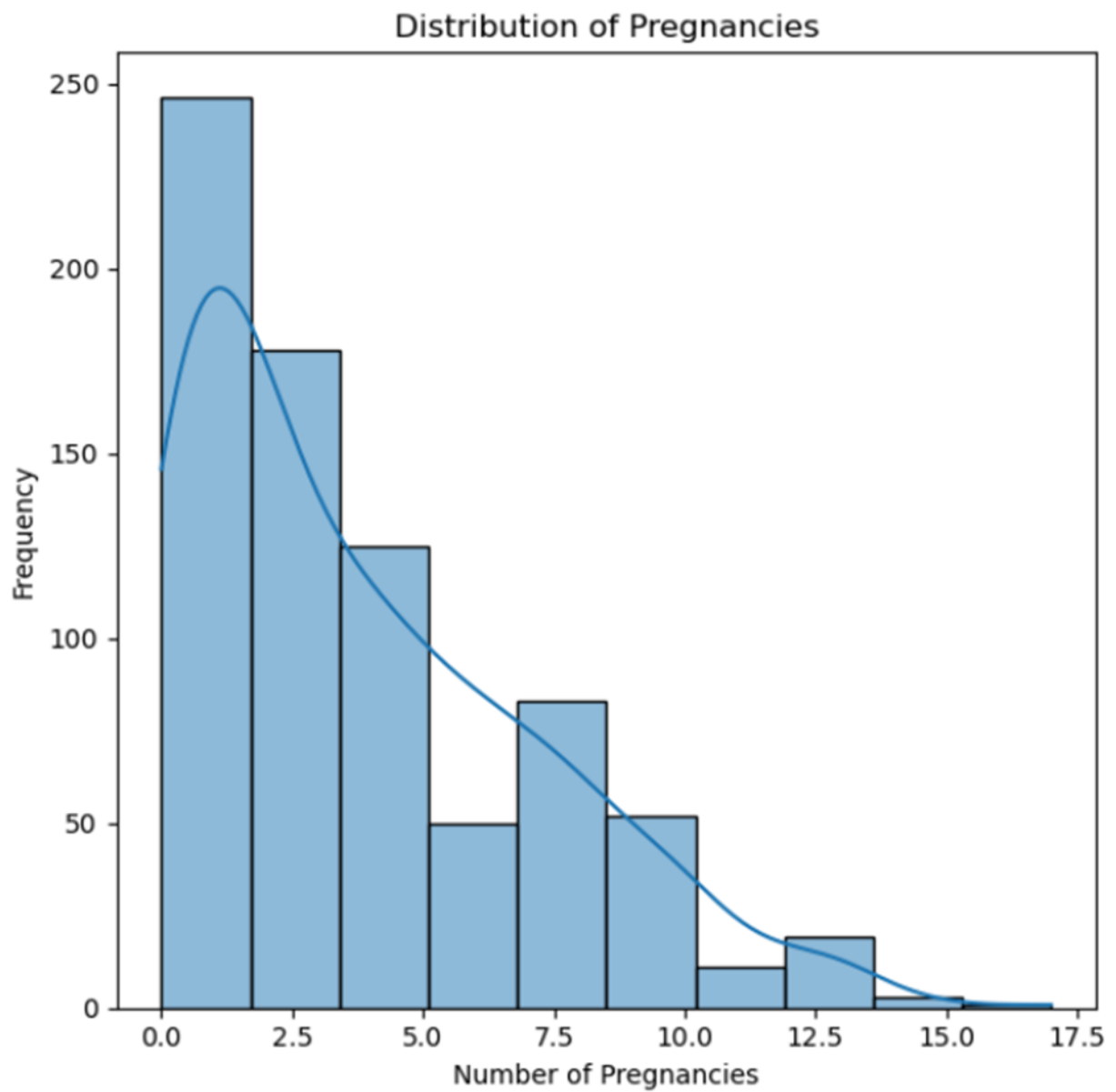
2.4 Sơ đồ hệ thống

Chương 3. Kết quả

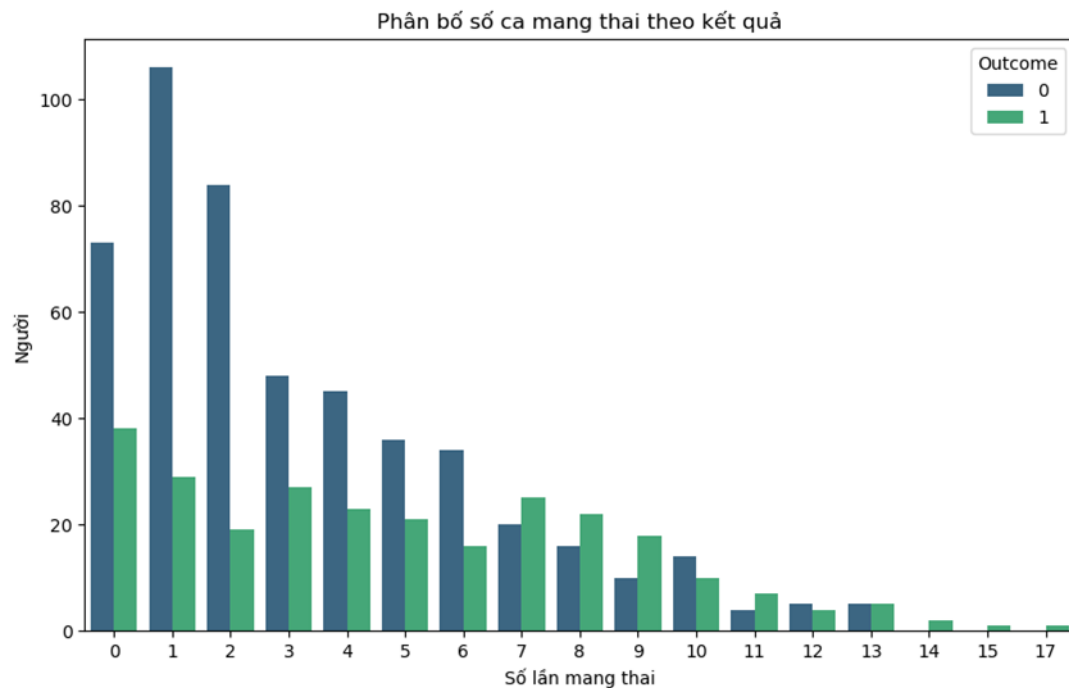
3.1 Phân tích dữ liệu (exploratory data analysis -EDA)



3.1.1. Pregnancies



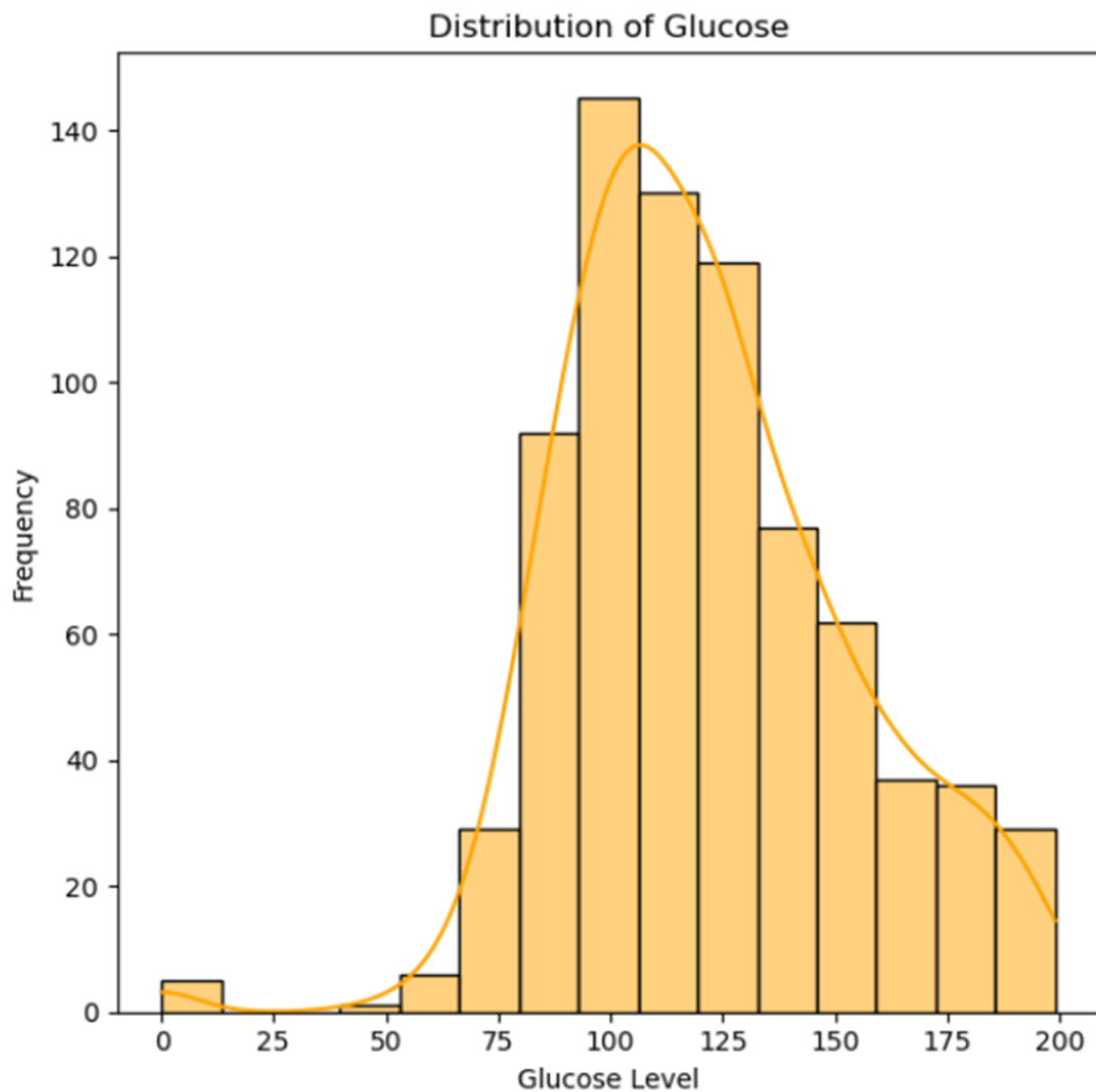
Tần suất mang thai từ 0 đến 5 lần là chiếm đa số.



Ta thấy số lần mang thai từ 0 đến 2 lần là tỉ lệ bị bệnh tiểu đường ít nhất, những người mang thai từ 0 đến 2 lần rất nhiều và cũng bị tiểu đường rất ít.

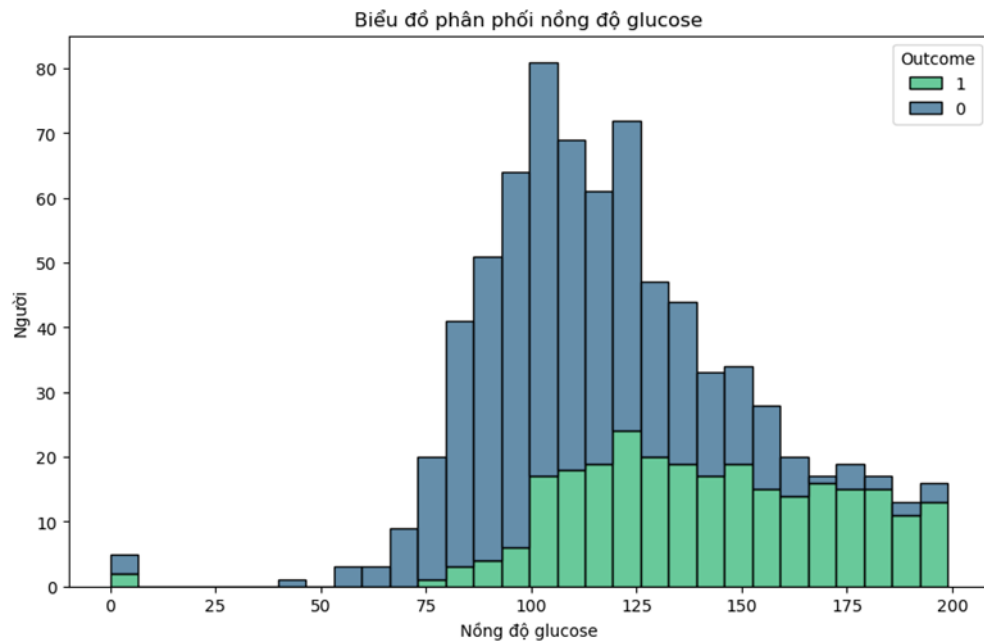
Từ 7 lần mang thai trở lên thì tỷ lệ mắc bệnh tiểu đường cao hơn.

3.1.2. Glucose



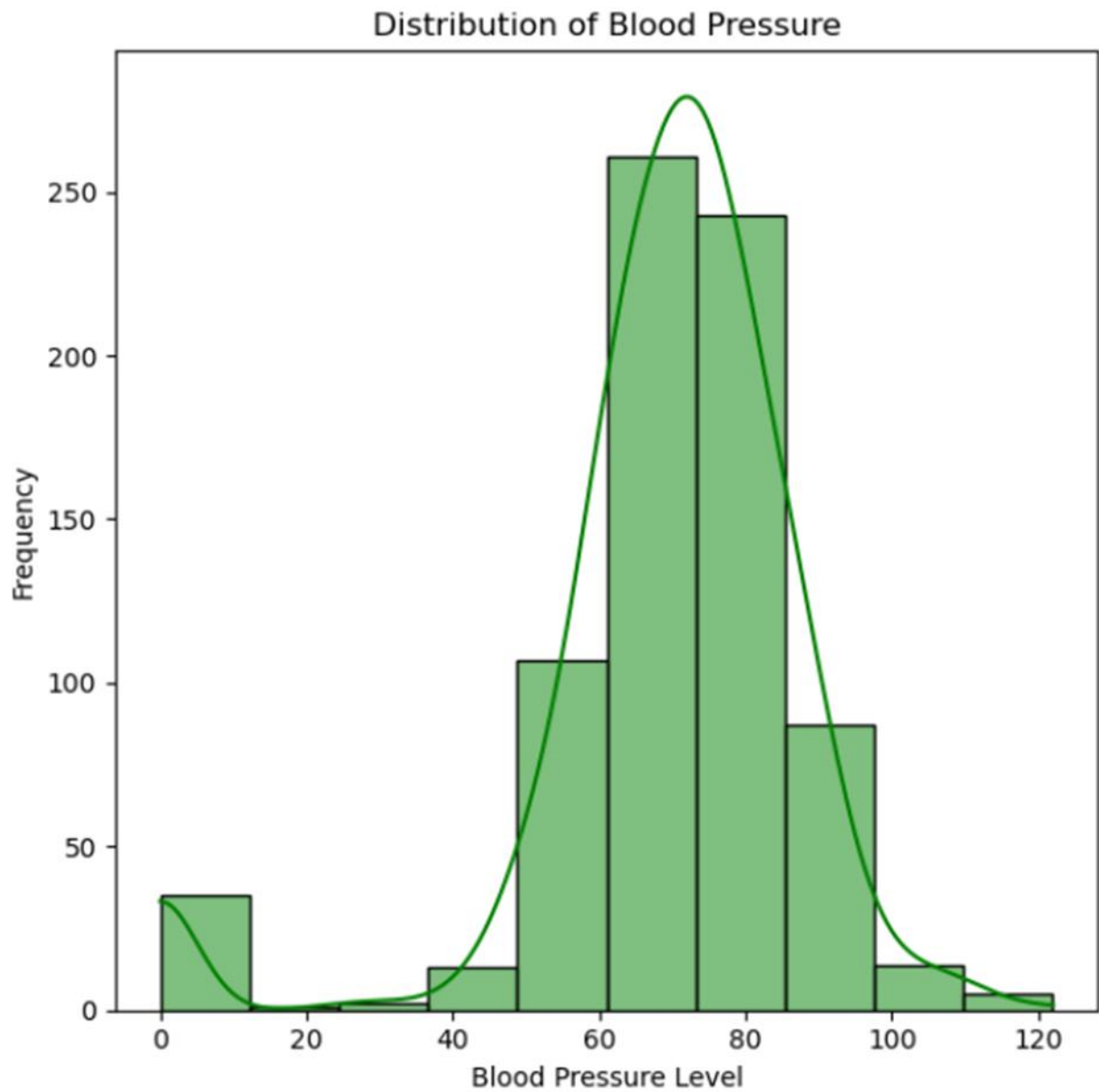
Người có nồng độ glucose từ 100 đến 125 chiếm đa số.

Theo biểu đồ, nồng độ glucose từ 25 đến 75 không bị tiểu đường. Từ 75 đến 125 thì số người không bị tiểu đường chiếm đa số so với số người bị tiểu đường. Từ 125 đến 150 tỷ lệ bị bệnh và không bị bệnh sẽ ngang nhau, khi vượt ngưỡng 150 thì sẽ có nhiều nguy cơ bị bệnh tiểu đường hơn.

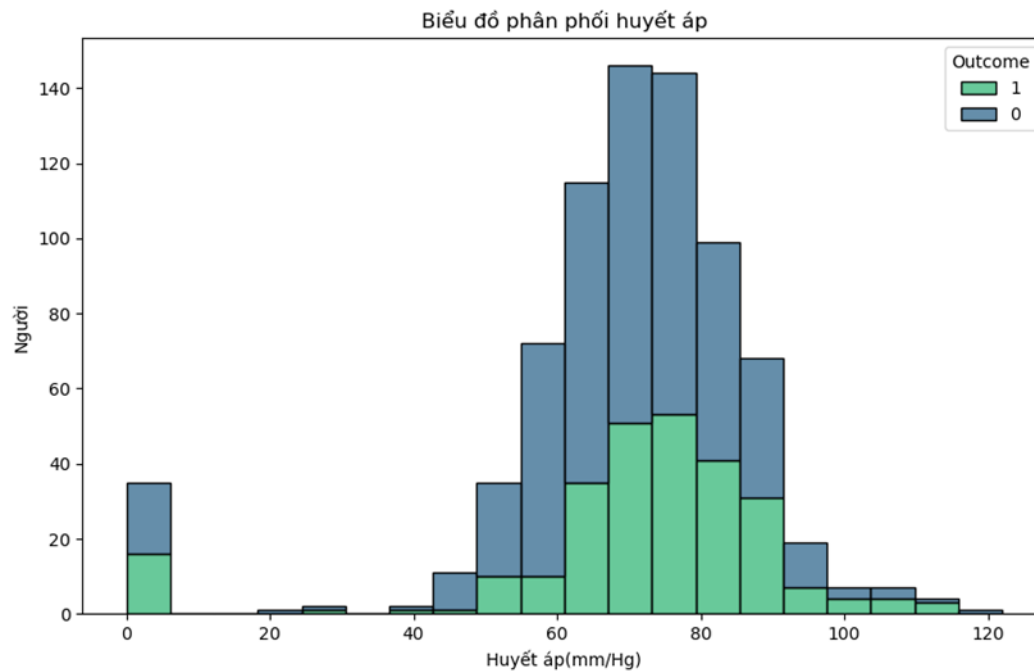


Theo biểu đồ, nồng độ glucose từ 25 đến 75 không bị tiểu đường. Từ 75 đến 125 thì số người không bị tiểu đường chiếm đa số so với số người bị tiểu đường. Từ 125 đến 150 tỷ lệ bị bệnh và không bị bệnh sẽ ngang nhau, khi vượt ngưỡng 150 thì sẽ có nhiều nguy cơ bị bệnh tiểu đường hơn.

3.1.3. BloodPressure



Người có huyết áp từ 60 đến 80 chiếm đa số.

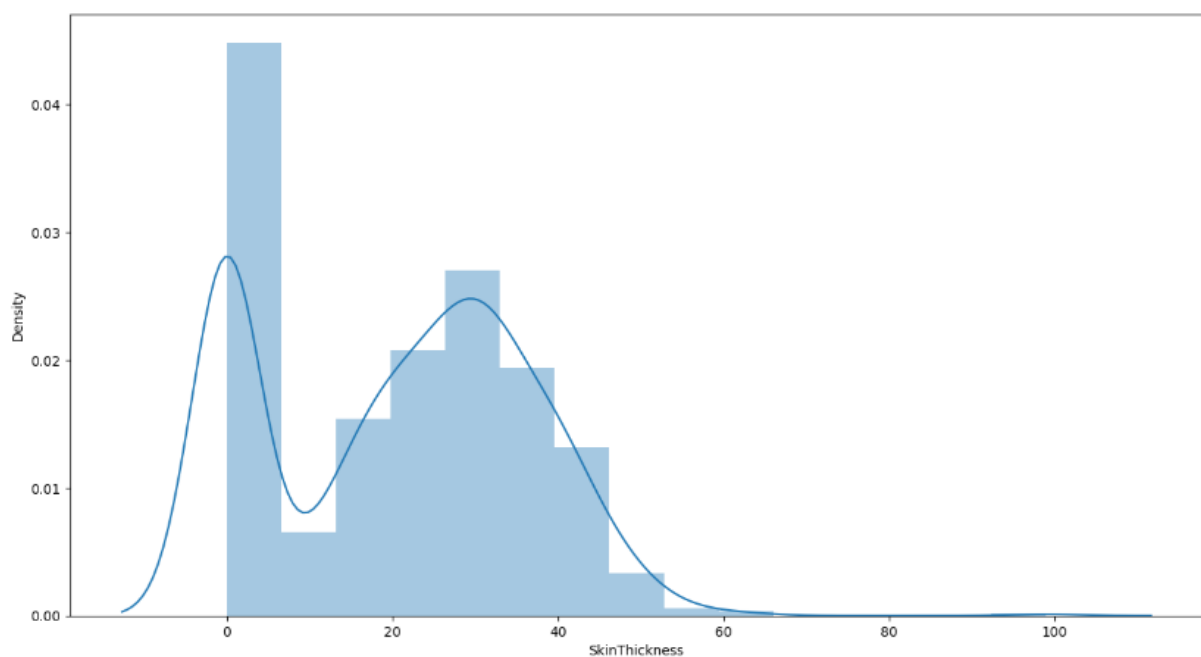


Theo biểu đồ, huyết áp từ 20 đến 50 mm/Hg sẽ không bị tiểu đường, từ 50 đến 60 mm/Hg sẽ có ít tỷ lệ bị bệnh tiểu đường.

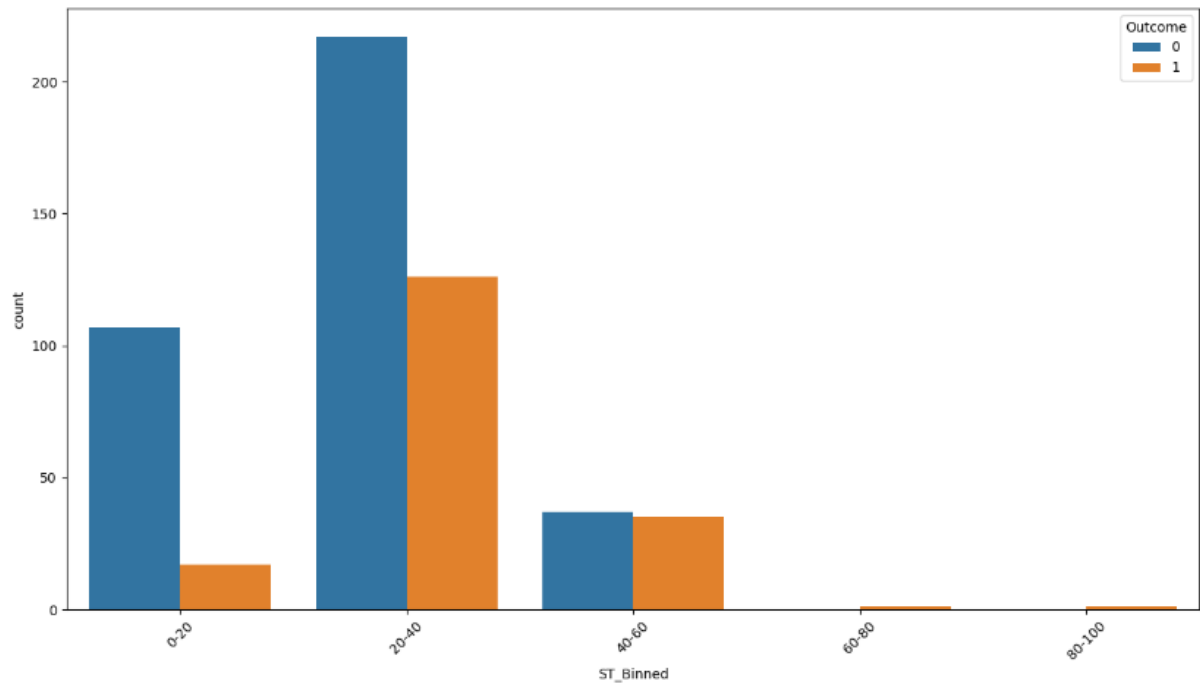
Từ 60 đến 80 mm/Hg tỷ lệ bị bệnh tiểu đường sẽ tăng lên nhưng tỷ lệ không bị vẫn cao hơn.

Từ 80 mm/Hg trở lên tỷ lệ bị bệnh tiểu đường sẽ cao hơn.

3.1.4. SkinThickness

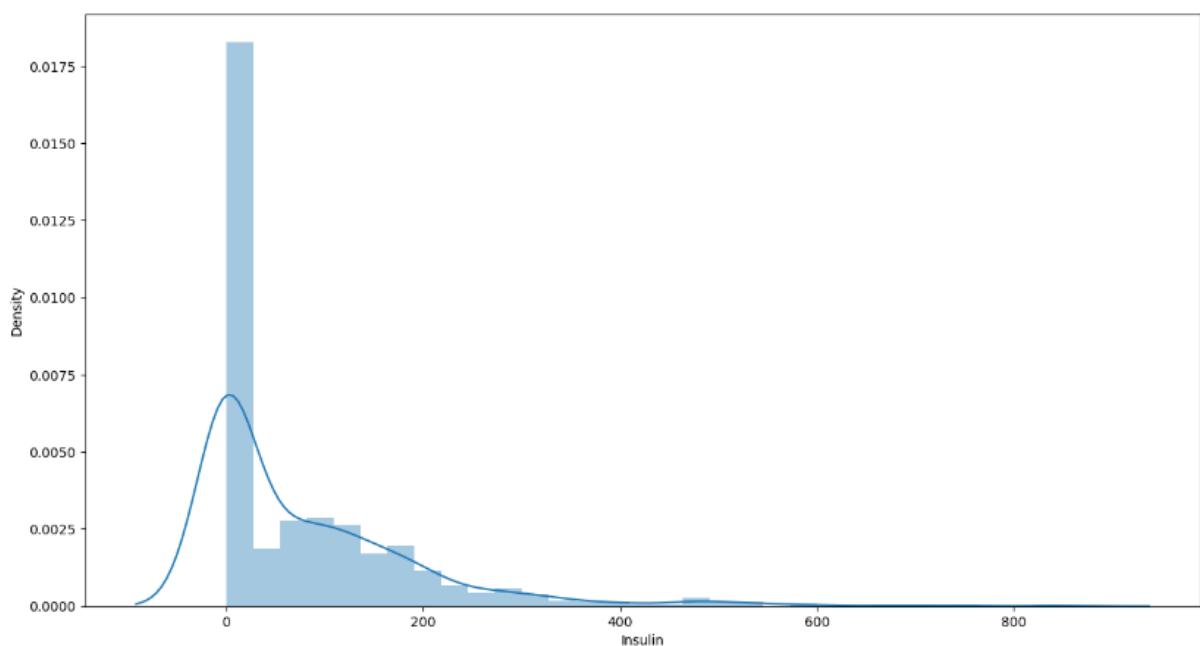


Phân phối gần như chuẩn: Phân phối của Skin Thickness gần giống với phân phối chuẩn, với một đỉnh chính ở giữa và các giá trị giảm dần về hai phía. Điều này cho thấy độ dày da của nhóm đối tượng nghiên cứu có sự phân bố tương đối đồng đều.



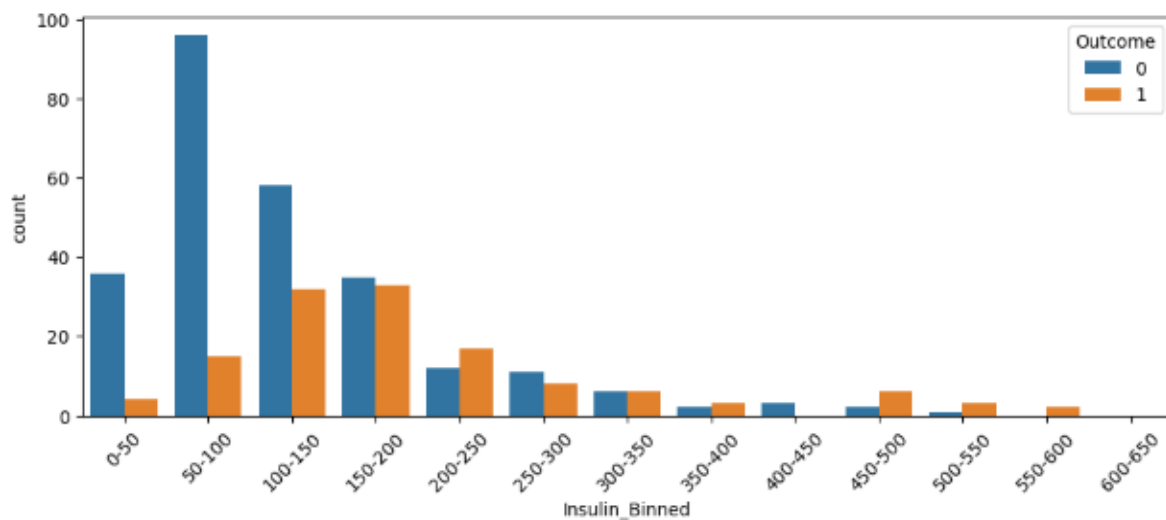
Giá trị này càng cao thì tỉ lệ mắc bệnh càng lớn

3.1.5. Insulin

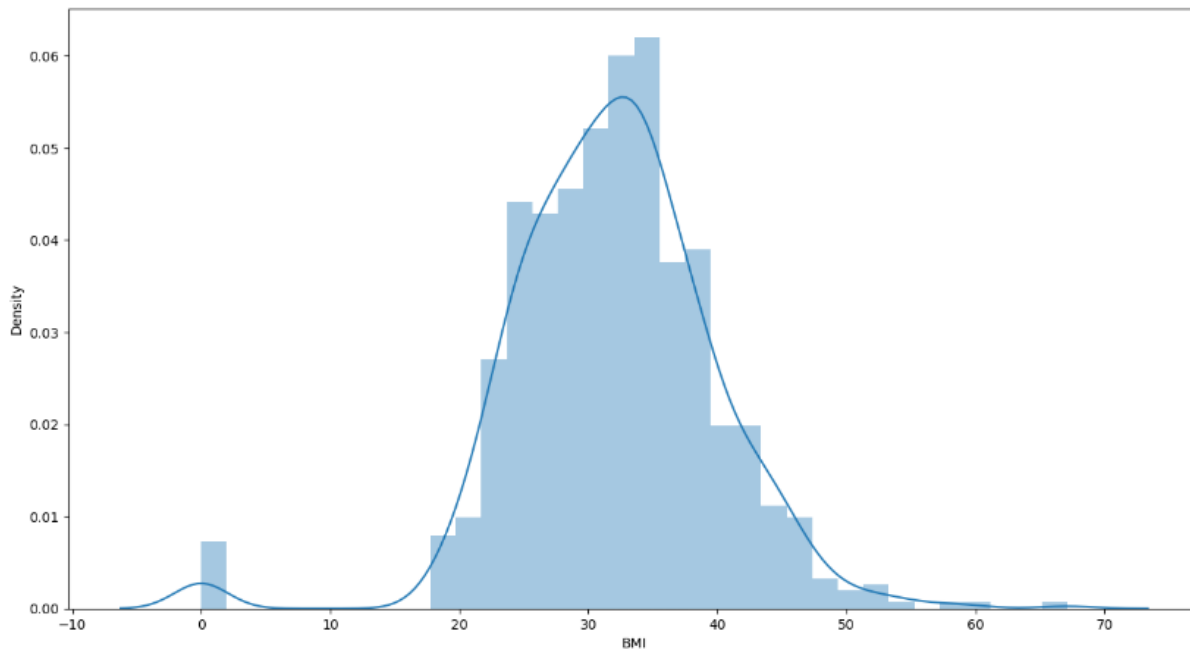


Phân phối lệch phải: phân phối của Insulin lệch phải. Điều này cho thấy đa số người có mức insulin tương đối thấp, nhưng có một số ít người có mức insulin rất cao.

Có thể có nhiều nhóm: Đồ thị cho thấy có thể có nhiều nhóm khác nhau về mức insulin. Điều này có thể liên quan đến các yếu tố khác như tuổi tác, chế độ ăn uống, hoặc tình trạng sức khỏe.

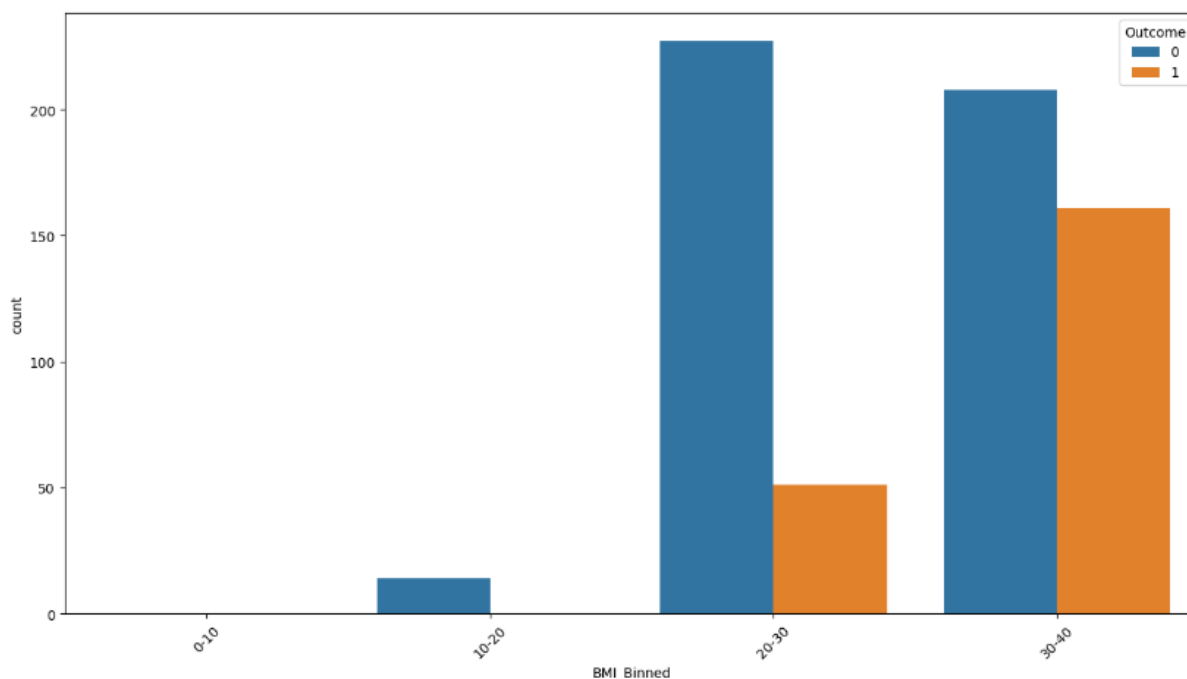


3.1.6. BMI



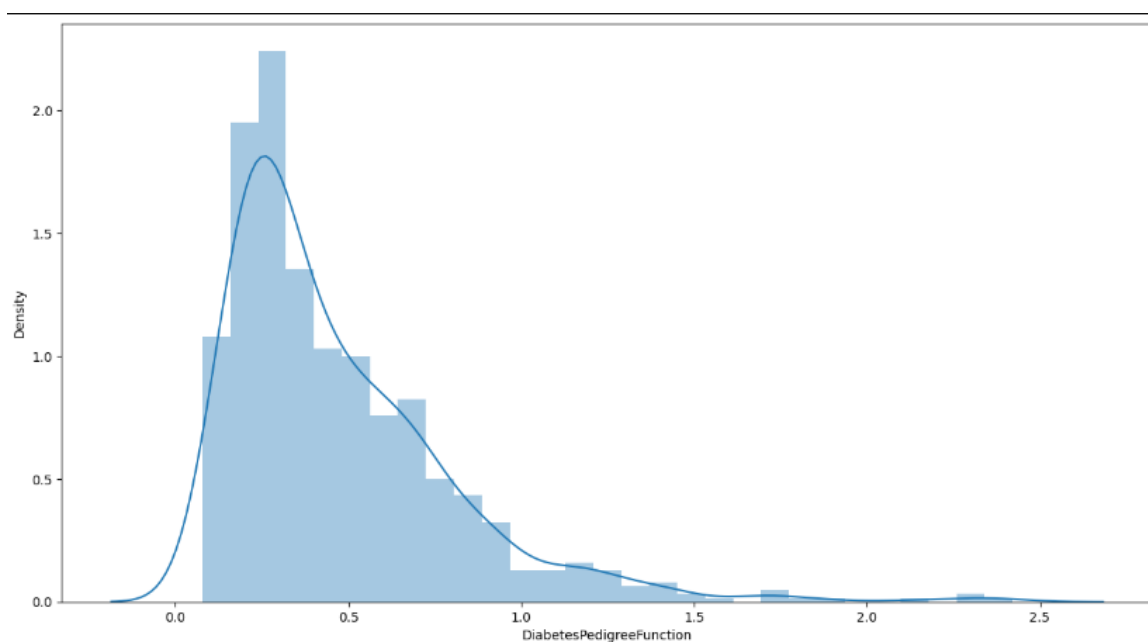
Phân phối lệch phải: Hầu hết các giá trị đều tập trung dưới 30, nhưng có một số giá trị rất lớn mở rộng hơn về bên phải. Điều này chỉ ra rằng có một số cá nhân có BMI rất cao so với những người còn lại.

Các giá trị ngoại lệ có thể xảy ra: Các giá trị BMI rất cao ở phía bên phải có thể được coi là các giá trị ngoại lệ, cần được xem xét kỹ hơn để đánh giá tác động của chúng đối với phân tích.

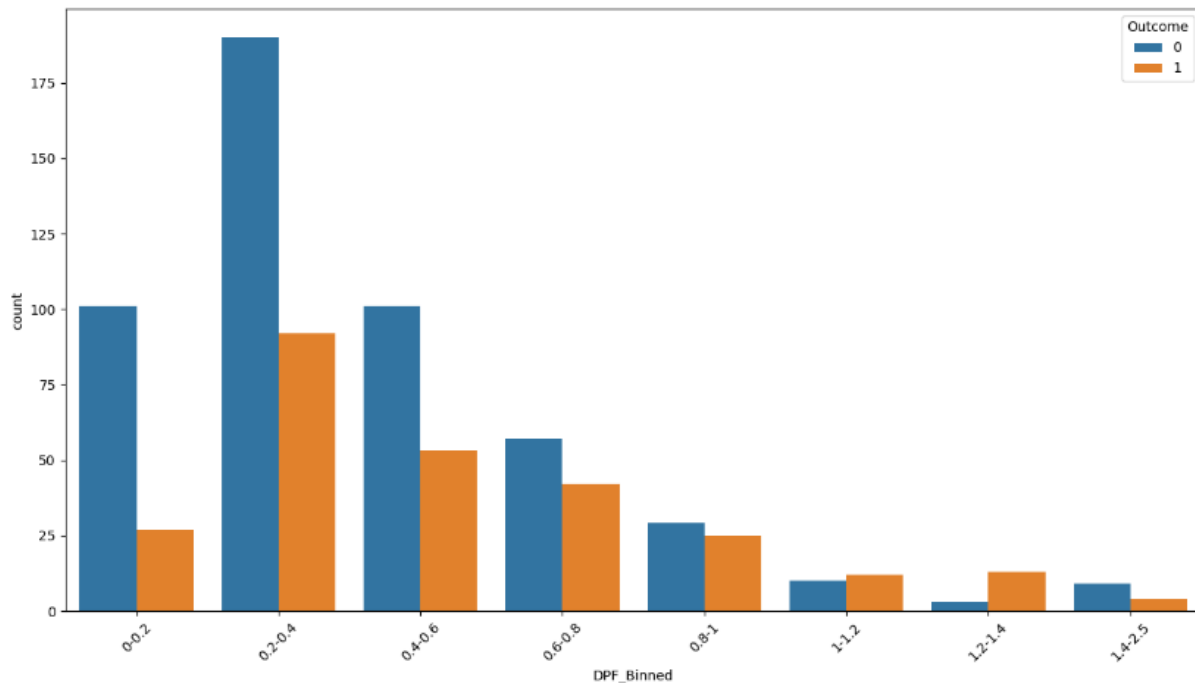


BMI càng thấp thì tỉ lệ mắc bệnh càng thấp.

3.1.7. DiabetesPedigreeFunction

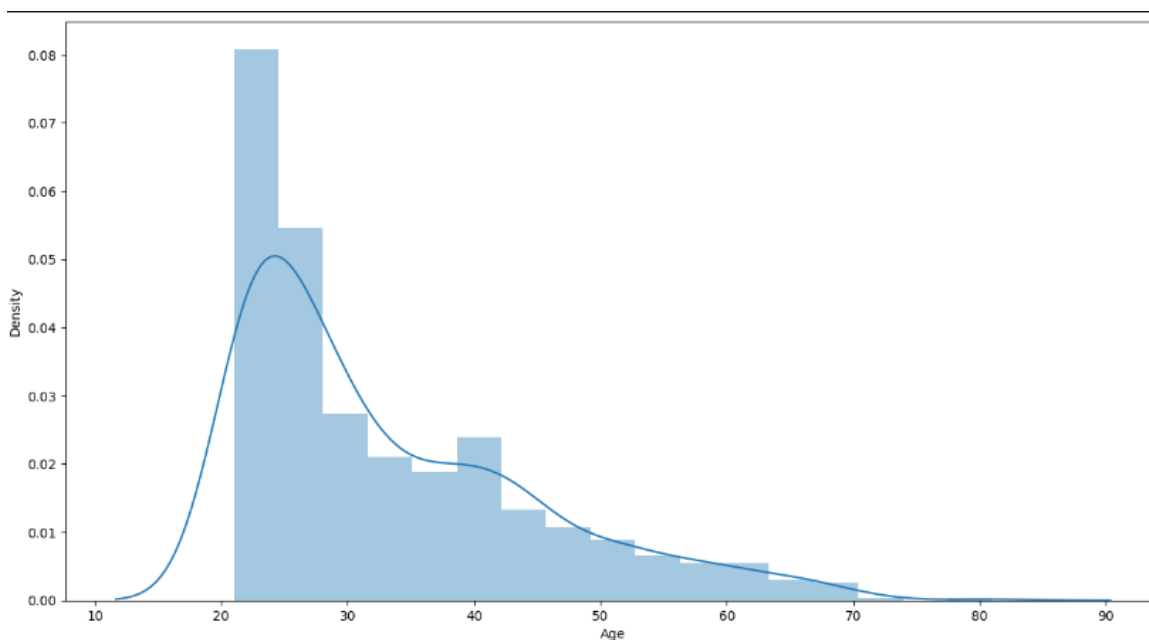


Hệ số di truyền này dao động nhiều từ 0.1 đến 1.0 và xuất hiện thưa thớt ở 1.5 đến 2.5 .



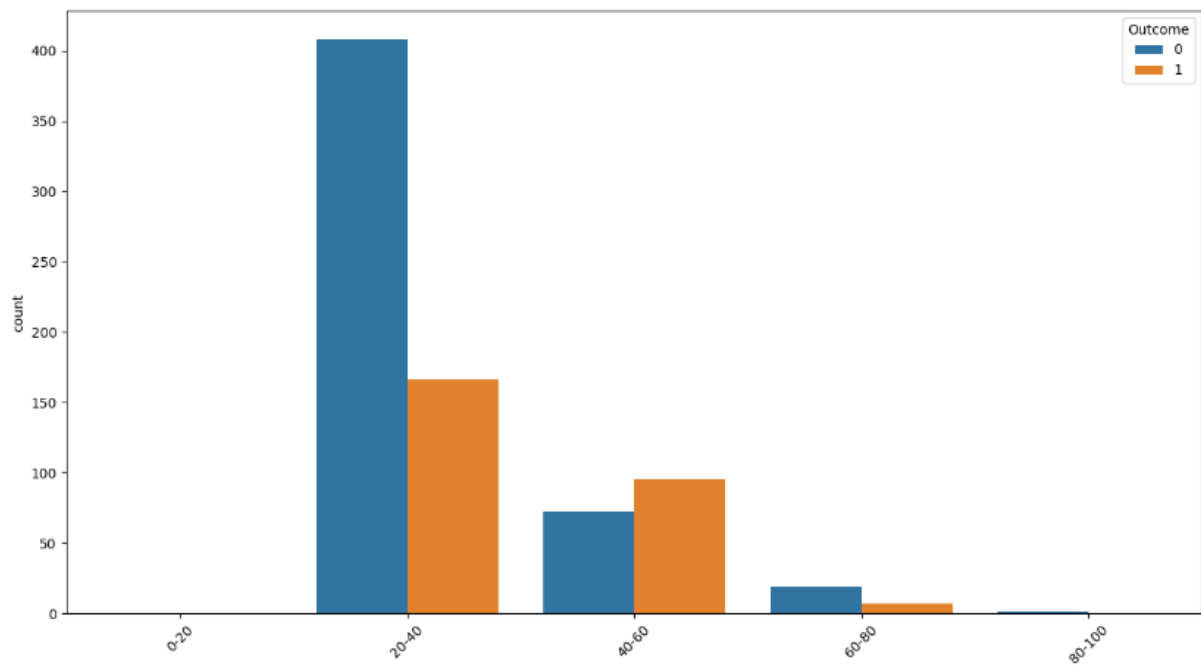
Dựa vào biểu đồ ta thấy giá trị này không ảnh hưởng nhiều đến tỉ lệ mắc bệnh. Tuy vậy có vẻ giá trị này càng cao thì càng dễ mắc bệnh. Giá trị cực thấp vẫn có khả năng mắc bệnh nhưng không nhiều.

3.1.8. Age



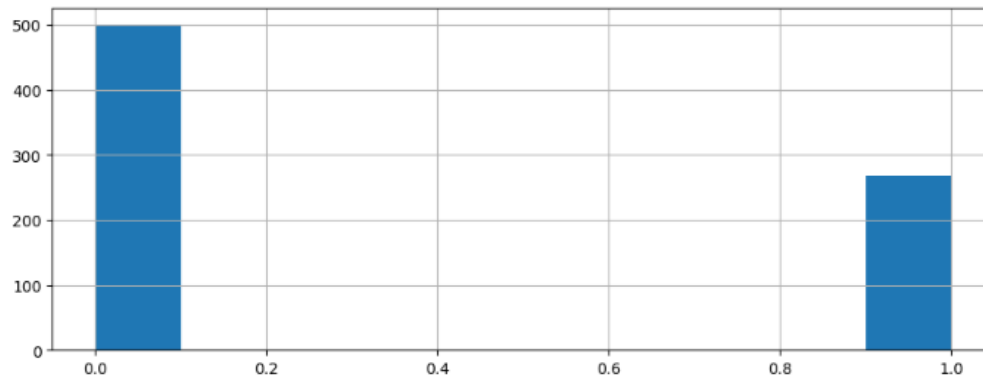
Trong bộ dữ liệu này, độ tuổi phổ biến nhất là từ 20 đến 30 và giảm dần cho đến

70 tuổi.



Dựa vào biểu đồ ta thấy, tỉ lệ mắc bệnh tiểu đường của người trẻ thấp hơn nhiều so với những người trung niên và cao tuổi. Ở những người trung niên và cao tuổi tỉ lệ người mắc bệnh đôi khi còn cao hơn so với tỉ lệ những người không mắc bệnh.

3.1.9. Outcome



```
pd.DataFrame(y.value_counts())
```

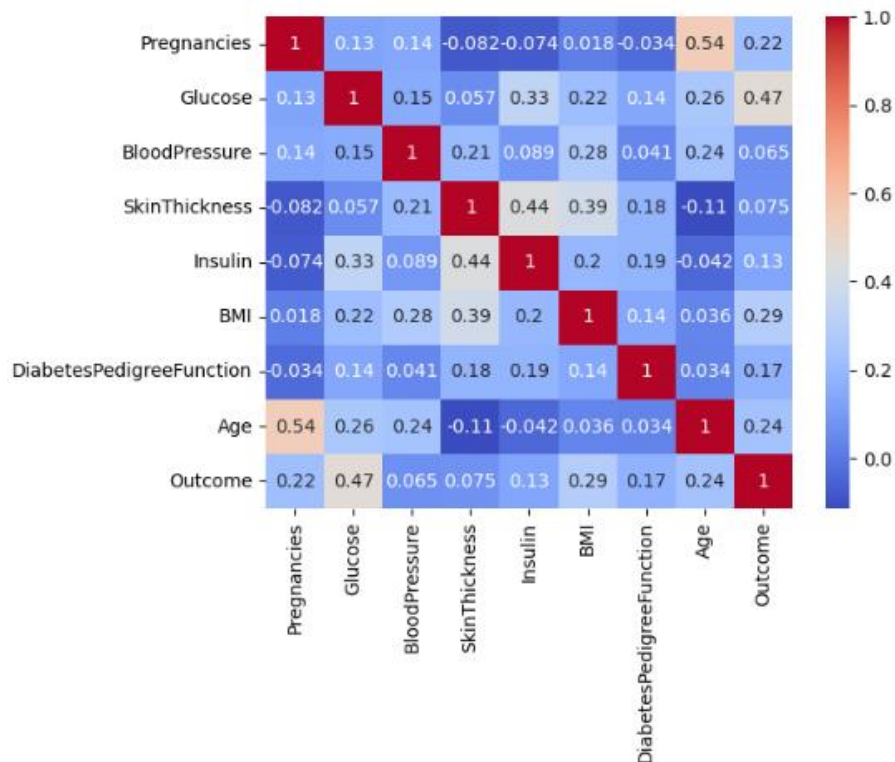
count	
Outcome	
0	500
1	268

Giá trị đầu ra được phân thành hai giá trị:

- 0: Không mắc bệnh
- 1: Mắc bệnh

Bộ dữ liệu có 500 negative cases và 268 postive cases. Cho thấy bộ dữ liệu này bị mất cân bằng giữa các lớp.

3.1.10. Ma trận tương quan



Theo ma trận tương quan, Glucose ảnh hưởng tới Outcome nhiều nhất với hệ số tương quan là 0.47, tiếp đến là BMI với hệ số tương quan là 0.29, Age với hệ số tương quan là 0.24... Thấp nhất là BloodPressure với hệ số tương quan là 0.065.

3.2 Tiền xử lý dữ liệu

3.2.1. Dữ liệu ban đầu

Dữ liệu sau khi thu thập sẽ được đưa vào bảng có đuôi .csv

diabetes.csv - Excel

FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW

Clipboard Font Alignment Number Conditional Formatting

B1 Glucose

	A	B	C	D	E	F	G	H	I
1	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0

diabetes

Bảng có 9 cột: 8 cột đầu vào và một cột đầu ra

Cột Pregnancies đưa ra số lần mang thai.

Cột Glucose đưa ra mức glucose trong huyết tương (mg/dL).

Cột BloodPressure đưa ra huyết áp tâm trương (mmHg).

Cột SkinThickness đưa ra độ dày nếp gấp da (mm).

Cột Insulin đưa ra mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose).

Cột BMI đưa ra chỉ số khối cơ thể (được tính bằng cân nặng/ chiều cao bình phương).

Cột DiabetesPedigreeFunction đưa ra chỉ số truyền bệnh tiểu đường, phản ánh mức độ di truyền của bệnh.

Lấy file dữ liệu:

```
data = pd.read_csv('diabetes.csv')
```

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
print(data.shape)
```

```
(768, 9)
```

File này có 768 hàng và 9 cột

Tách dữ liệu thành đầu ra và đầu vào

```
X = data.drop('Outcome', axis=1)
y = data['Outcome']
```

Chia dữ liệu thành tập train và tập test

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Với tỉ lệ 80% train và 20% test

```
print(X_train.shape, X_test.shape)
```

```
(614, 8) (154, 8)
```

```
print(y_train.shape, y_test.shape)
```

```
(614,) (154,)
```

Ta loại bỏ khoảng trống không cần thiết trong tập dữ liệu

```
] : #Loại bỏ khoảng trống không cần thiết
```

```
] : X_train=X_train.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

```
] : X_test=X_test.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

3.2.2. Xử lý giá trị bị thiếu

```
print(data.isna().sum())
```

```
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age               0
Outcome           0
dtype: int64
```

```
data.isin(['?']).sum()
```

```
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age               0
Outcome           0
dtype: int64
```

Sau khi tìm kiếm thì không hề có giá trị bị thiếu hoặc giá trị bị thay thế bởi dấu ‘?’.

Ta thử với giá trị bằng 0:

```
print((X_train==0).sum())
```

```
Pregnancies      91
Glucose           5
BloodPressure     24
SkinThickness     176
Insulin           290
BMI               7
DiabetesPedigreeFunction  0
Age               0
dtype: int64
```

Ta thấy có rất nhiều giá trị bằng 0 không có nghĩa như là của glucose, bloodpressure, skin thickness, insulin và bmi. Cột pregnancies có khoảng 91 giá trị bằng 0 nhưng hoàn toàn hợp lý (vì có thể mang thai 0 lần).

Ta xử lý những giá trị này:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="most_frequent")#thay thế bằng giá trị xuất hiện nhiều nhất
```

```
imputer.fit(X_train[['Glucose']])
imputer.fit(X_train[['BloodPressure']])
imputer.fit(X_train[['SkinThickness']])
imputer.fit(X_train[['Insulin']])
imputer.fit(X_train[['BMI']])
```

Đầu tiên ta thay những giá trị bằng 0 không có nghĩa thành giá trị nan.

```
X_train[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = X_train[['Glucose',
    'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0, np.nan)

imputer.fit(X_train[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']])

# Sau đó, sử dụng transform để thay thế giá trị missing
X_train[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = imputer.transform(X_train[['Glucose',
    'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']])
```

Sau đó thay những giá trị NaN này thành giá trị xuất hiện nhiều nhất.

Tương tự với X_test

```
: imputer.fit(X_test[['Glucose']])
imputer.fit(X_test[['BloodPressure']])
imputer.fit(X_test[['SkinThickness']])
imputer.fit(X_test[['Insulin']])
imputer.fit(X_test[['BMI']])
```

```
: SimpleImputer
SimpleImputer(strategy='most_frequent')
```

```
: X_test[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = X_test[['Glucose',
    'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0, np.nan)

imputer.fit(X_test[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']])

# Sau đó, sử dụng transform để thay thế giá trị missing
X_test[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = imputer.transform(X_test[['Glucose',
    'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']])
```

Sau khi thay thế:

```
[37]: print((X_test==0).sum())
```

```
Pregnancies      20
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age               0
dtype: int64
```

```
[38]: print((X_train==0).sum())
```

```
Pregnancies      91
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age               0
dtype: int64
```

```
[ ]: #như vậy đã loại bỏ được các giá trị bằng 0 không có nghĩa trong X_train và X_test
```

Giá trị bằng 0 đã được thay thế bằng các giá trị được xuất hiện nhiều nhất.

3.2.3. Cân bằng dữ liệu

Phân tích dữ liệu:

```
[42]: #Y_train
print('Tổng số quan sát : {}'.format(len(y_train)))

print('Số người mắc bệnh : {}'.format(round(((y_train==1).sum()/len(y_train))*100, 2)))

print('Số người không mắc bệnh : {}'.format(round(((y_train==0).sum()/len(y_train))*100, 2)))
```

```
Tổng số quan sát : 614
```

```
Số người mắc bệnh : 34.69%
```

```
Số người không mắc bệnh : 65.31%
```

```
[43]: #Y_test
print('Tổng số quan sát : {}'.format(len(y_test)))
|
print('Số người mắc bệnh : {}'.format(round(((y_test==1).sum()/len(y_test))*100, 2)))

print('Số người không mắc bệnh : {}'.format(round(((y_test==0).sum()/len(y_test))*100, 2)))
```

```
Tổng số quan sát : 154
```

```
Số người mắc bệnh : 35.71%
```

```
Số người không mắc bệnh : 64.29%
```

Dữ liệu bị mất cân bằng với số người mắc bệnh chiếm khoảng 35.7% và số người không mắc bệnh chiếm khoảng 64.3%.


```

j: pd.DataFrame(y.value_counts())
j:
      count
Outcome
0         500
1         268

```

Tổng cộng có 500 giá trị đầu ra bằng 0 và 268 giá trị đầu ra bằng 1.

Ta cần phải cân bằng dữ liệu bằng cách tăng mẫu của lớp thiểu số hoặc giảm mẫu của lớp đa số. Với bộ dữ liệu không nhiều (768 mẫu) thì giảm mẫu không phải là lựa chọn tốt. Vì thế ta phải tăng mẫu của lớp thiểu số.

Import thư viện:

```

from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled_train, y_resampled_train = smote.fit_resample(X_train, y_train)

```

imblearn: Đây là thư viện imbalanced-learn, một thư viện Python phổ biến cho việc xử lý dữ liệu mất cân bằng.

SMOTE: Là một lớp trong imblearn, dùng để thực hiện việc oversampling dữ liệu cho lớp thiểu số bằng cách tạo ra các mẫu tổng hợp mới.

Khởi tạo:

SMOTE (random_state=42): Khởi tạo một đối tượng SMOTE với random_state=42 để đảm bảo tính ngẫu nhiên có thể tái hiện được, random_state giúp ta có được cùng một kết quả khi chạy lại mã, đảm bảo tính nhất quán.

Cân bằng dữ liệu:

smote.fit_resample (X_train, y_train): Phương thức fit_resample của SMOTE được sử dụng để thực hiện quá trình oversampling.

Tương tự với các file test:

```

from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled_test, y_resampled_test = smote.fit_resample(X_test, y_test)

```


Sau khi cân bằng:

```
print('Tổng số quan sát : {}'.format(len(y_resampled_train)))
print('Số người mắc bệnh : {}'.format(round(((y_resampled_train==1).sum()/len(y_resampled_train))*100, 2)))
print('Số người không mắc bệnh : {}'.format(round(((y_resampled_train==0).sum()/len(y_resampled_train))*100, 2)))
Tổng số quan sát : 802
Số người mắc bệnh : 50.0%
Số người không mắc bệnh : 50.0%

print('Tổng số quan sát : {}'.format(len(y_resampled_test)))
print('Số người mắc bệnh : {}'.format(round(((y_resampled_test==1).sum()/len(y_resampled_test))*100, 2)))
print('Số người không mắc bệnh : {}'.format(round(((y_resampled_test==0).sum()/len(y_resampled_test))*100, 2)))
Tổng số quan sát : 198
Số người mắc bệnh : 50.0%
Số người không mắc bệnh : 50.0%
```

Số người mắc bệnh và không mắc bệnh bằng nhau, mẫu dữ liệu đã được cân bằng.

```
: pd.DataFrame(X_resampled_train).to_csv('X_train.csv', index=False)
pd.DataFrame(X_resampled_test).to_csv('X_test.csv', index=False)

pd.DataFrame(y_resampled_train).to_csv('y_train.csv', index=False, header=True)
pd.DataFrame(y_resampled_test).to_csv('y_test.csv', index=False, header=True)
```

Ta xuất dữ liệu đã tiền xử lý sang các file để dễ dàng sử dụng.

3.3 Mô hình hóa

3.3.1. Logistics

3.3.1.1. Khái niệm

Hồi quy logistics (logistics regression) là một thuật toán học máy có giám sát, phân tích dữ liệu nhằm mô hình hóa mối quan hệ giữa một hoặc nhiều biến đầu vào (các yếu tố dữ liệu) và một biến đầu ra nhị phân (có hai giá trị, thường là 0 hoặc 1). Thay vì dự đoán một giá trị cụ thể của biến đầu ra, hồi quy logistics dự đoán xác suất mà biến đầu ra thuộc về một trong hai nhóm hoặc lớp (chẳng hạn, "có" hoặc "không").

Ưu điểm của thuật toán này là đơn giản, tương đối dễ thực hiện, huấn luyện nhanh và linh hoạt hơn hồi quy tuyến tính. Tuy nhiên đây chưa được coi là một trong

những thuật toán đem lại sự phân loại chính xác nhất, dễ ảnh hưởng bởi nhiễu và không hiệu quả với dữ liệu phức tạp hoặc phi tuyến tính.

Hồi quy Logistics sử dụng hàm tuyến tính và hàm sigmoid.

Trước tiên, hồi quy logistic tính toán một tổ hợp tuyến tính của các biến đầu vào:

$$z = w^T x + b$$

- w là vector trọng số (weights), thể hiện mức độ ảnh hưởng của từng đặc trưng x đến kết quả đầu ra.
- x là vector các giá trị đặc trưng của mẫu đầu vào.
- b là bias, một hằng số được thêm vào để điều chỉnh mô hình.
- Giá trị z có thể là bất kỳ số thực nào, bao gồm cả số dương và âm, tùy thuộc vào các giá trị của x và w .

Hàm sigmoid:

Hồi quy logistic sử dụng hàm sigmoid để chuyển đổi giá trị tuyến tính z thành một xác suất nằm trong khoảng từ 0 đến 1:

$$P(y=1|x) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Hàm sigmoid nhận giá trị z và nén nó vào khoảng $[0, 1]$. Điều này rất hữu ích cho việc dự đoán xác suất, vì xác suất phải là một giá trị giữa 0 và 1.

Sau khi áp dụng hàm sigmoid, kết quả đầu ra $\sigma(z)$ đại diện cho xác suất mà mẫu đầu vào thuộc về lớp 1 (hoặc có thể là kết quả "có" trong một bài toán nhị phân). Giá trị xác suất này sẽ so sánh với ngưỡng nào đó (thường là 50%) nếu $P(y=1|x)$ thì giá trị sẽ trả về là có mắc bệnh (1) và không mắc bệnh (0) nếu thấp hơn 50%.

3.3.1.2. Huấn luyện mô hình

Import thư viện cần thiết

```
[20]: import numpy as np
import pandas as pd
import pickle
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report
# Visualizations
import matplotlib.pyplot as plt

import seaborn as sns
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore');
```

- `import numpy as np`: Nhập thư viện NumPy, một thư viện phổ biến để xử lý các mảng số học.
- `import pandas as pd`: Nhập thư viện Pandas, sử dụng để thao tác với dữ liệu dạng bảng (DataFrame).
- `import pickle`: Nhập thư viện pickle, dùng để lưu trữ và tải lại các đối tượng Python (như mô hình học máy) dưới dạng file nhị phân.
- `from sklearn.linear_model import LogisticRegression`: Nhập lớp Logistic Regression từ thư viện scikit-learn để sử dụng mô hình hồi quy logistic.
- `from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score`: Nhập các hàm để tính các chỉ số đánh giá hiệu suất của mô hình như độ chính xác, độ chính xác (precision), độ nhạy (recall), điểm F1, và diện tích dưới đường cong ROC (AUC).
- `from sklearn.metrics import confusion_matrix, classification_report`: Nhập các hàm để tạo ma trận nhầm lẫn và báo cáo phân loại cho mô hình.
- `import seaborn as sns`: Nhập thư viện Seaborn, một thư viện mạnh mẽ để trực quan hóa dữ liệu.
- `import matplotlib.pyplot as plt`: Nhập Matplotlib để tạo biểu đồ.
- `%matplotlib inline`: Lệnh này giúp hiển thị các biểu đồ ngay trong notebook.
- `import warnings` và `warnings.filterwarnings('ignore')`: Tắt các cảnh báo để tránh bị làm phiền bởi các thông báo không cần thiết trong quá trình chạy.

code.

Đọc các file train, test đã lưu từ tiền xử lý:

```
[5]: X_train = pd.read_csv('X_train.csv')
[6]: X_test = pd.read_csv('X_test.csv')
[7]: y_train = pd.read_csv('y_train.csv')
[8]: y_test = pd.read_csv('y_test.csv')
```

Huấn luyện mô hình từ các file train:

```
[17]: LRModel=LogisticRegression(solver='lbfgs', max_iter=7600)
      LRModel.fit(X_train,y_train)
```

```
[17]: LogisticRegression
      LogisticRegression(max_iter=7600)
```

Khởi tạo mô hình hồi quy logistic với solver lbfgs và số lần lặp tối đa là 7600.

- lbfgs (Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm) là một thuật toán tối ưu hóa thường được sử dụng để huấn luyện các mô hình hồi quy logistic và các mô hình học máy khác. lbfgs là một biến thể của phương pháp Newton, nhưng thay vì tính toán toàn bộ ma trận Hessian (là ma trận đạo hàm bậc hai của hàm mất mát), nó chỉ lưu trữ một số giới hạn các phép tính để tiết kiệm bộ nhớ, do đó gọi là "limited-memory". Thuật toán này rất hiệu quả khi làm việc với các vấn đề có số lượng tham số lớn vì nó yêu cầu ít bộ nhớ hơn so với phương pháp Newton tiêu chuẩn và cũng có xu hướng hội tụ nhanh.
- max_iter là tham số chỉ định số lần lặp tối đa mà thuật toán lbfgs sẽ thực hiện trước khi dừng. Nếu mô hình không hội tụ (nghĩa là không đạt được sự ổn định trong các tham số) trong số lần lặp này, quá trình huấn luyện sẽ dừng lại. Ở đây ta dùng max_iter = 7600 không quá nhỏ cũng không quá lớn giúp mô hình hội tụ, đạt được độ chính xác mong muốn và đồng thời không tốn quá nhiều thời gian.

LRModel.fit(X_train, y_train): Huấn luyện mô hình LRModel trên dữ liệu huấn

luyện `X_train` và nhãn `y_train`.

Dự đoán:

```
y_pred = LRModel.predict(X_test)
```

Mô hình `LRModel` dự đoán nhãn cho dữ liệu kiểm tra `X_test` và lưu kết quả dự đoán vào biến `y_pred`.

Tính toán độ chính xác của mô hình:

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, LRModel.predict_proba(X_test)[:, 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
print(f"ROC-AUC: {roc_auc}")

Accuracy: 0.6818181818181818
Precision: 0.6875
Recall: 0.6666666666666666
F1-Score: 0.676923076923077
ROC-AUC: 0.7951229466380982
```

- `accuracy = accuracy_score(y_test, y_pred)`: Tính độ chính xác của mô hình, tức là tỷ lệ dự đoán đúng.
- `precision = precision_score(y_test, y_pred)`: Tính độ chính xác của các dự đoán "Bị tiểu đường".
- `recall = recall_score(y_test, y_pred)`: Tính tỷ lệ dự đoán đúng trong số các trường hợp thực sự "Bị tiểu đường".
- `f1 = f1_score(y_test, y_pred)`: Tính điểm F1, một thước đo kết hợp giữa độ chính xác và độ nhạy.
- `roc_auc = roc_auc_score(y_test, LRModel.predict_proba(X_test)[:, 1])`: Tính diện tích dưới đường cong ROC (AUC), một thước đo quan trọng khác của mô hình.

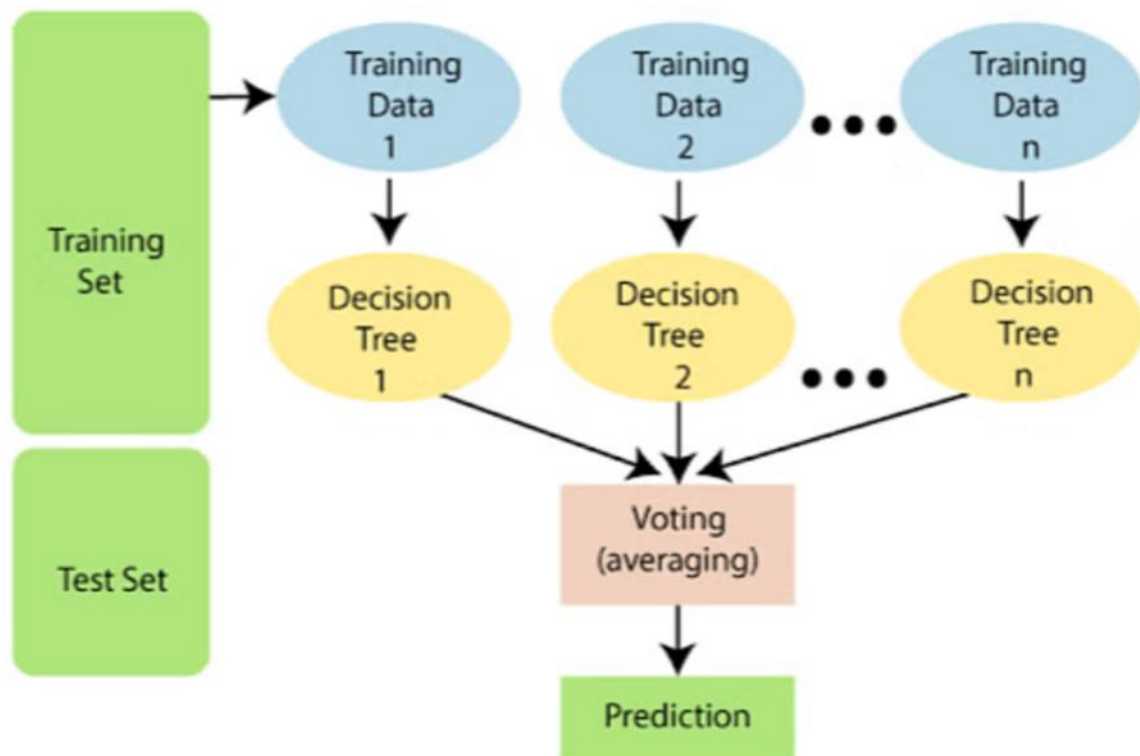
Độ chính xác của mô hình này là xấp xỉ 68%.

3.3.2. Random Forest

3.3.2.1. Khái niệm

Random forest là thuật toán supervised learning, có thể giải quyết cả bài toán regression và classification.

Random là ngẫu nhiên, Forest là rừng, nên ở thuật toán Random Forest mình sẽ xây dựng nhiều cây quyết định bằng thuật toán Decision Tree, tuy nhiên mỗi cây quyết định sẽ khác nhau (có yếu tố random). Sau đó kết quả dự đoán được tổng hợp từ các cây quyết định.



Các bước sau đây giải thích cách thức hoạt động của Thuật toán Rừng ngẫu nhiên:

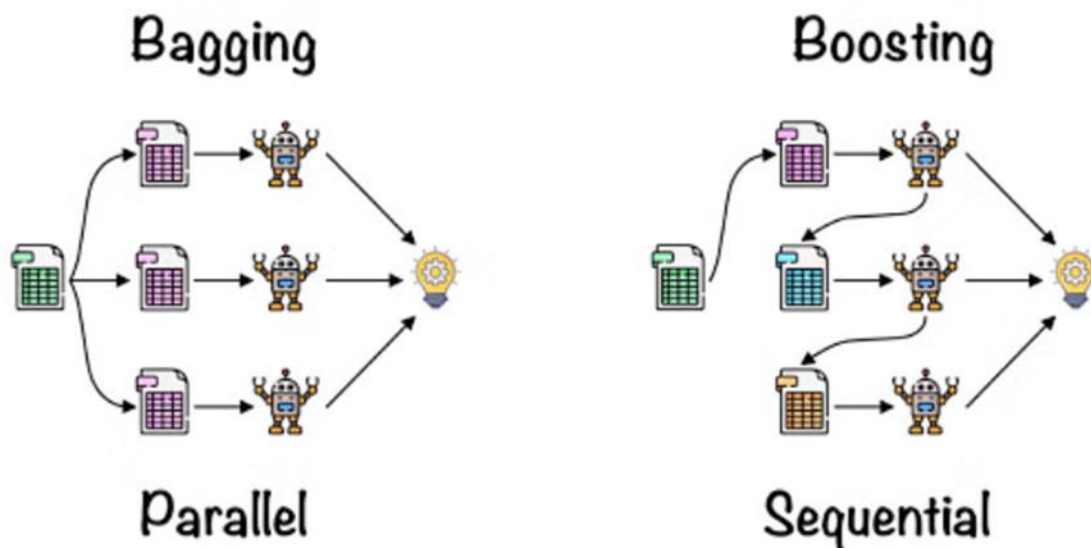
- Bước 1: Chọn mẫu ngẫu nhiên từ dữ liệu hoặc tập huấn luyện nhất định.
- Bước 2: Thuật toán này sẽ xây dựng một cây quyết định cho mọi dữ liệu đào tạo.
- Bước 3: Việc bỏ phiếu sẽ diễn ra bằng cách tính trung bình của cây quyết

định.

- Bước 4: Cuối cùng, chọn kết quả dự đoán được bình chọn nhiều nhất làm kết quả dự đoán cuối cùng.

Sự kết hợp của nhiều mô hình này được gọi là Ensemble. Ensemble sử dụng hai phương pháp:

- Bagging: Tạo một tập hợp con đào tạo khác từ dữ liệu đào tạo mẫu có thay thế được gọi là Bagging. Đầu ra cuối cùng dựa trên biểu quyết đa số.
- Boosting: Kết hợp những người học yếu thành những người học mạnh bằng cách tạo ra các mô hình tuần tự sao cho mô hình cuối cùng có độ chính xác cao nhất được gọi là boosting. Ví dụ: ADA BOOST, XG BOOST.



Bagging: Từ nguyên lý nêu trên, chúng ta có thể hiểu Random forest sử dụng mã Bagging. Bây giờ, chúng ta hãy cùng tìm hiểu khái niệm này một cách chi tiết. Bagging còn được gọi là Bootstrap Aggregation được random forest sử dụng. Quá trình này bắt đầu với bất kỳ dữ liệu ngẫu nhiên gốc nào. Sau khi sắp xếp, nó được tổ chức thành các mẫu được gọi là Bootstrap Sample. Quá trình này được gọi là Bootstrapping. Hơn nữa, các mô hình được đào tạo riêng lẻ, tạo ra các kết quả khác nhau được gọi là Aggregation. Ở bước cuối cùng, tất cả các kết quả được kết hợp và đầu ra được tạo ra dựa trên biểu quyết đa số. Bước này được gọi là

Bagging và được thực hiện bằng cách sử dụng Ensemble Classifier.

Các siêu tham số sau đây được sử dụng để tăng cường khả năng dự đoán:

- `n_estimators`: Số cây được thuật toán xây dựng trước khi tính trung bình các tích.
- `max_features`: Số lượng tính năng tối đa mà rừng ngẫu nhiên sử dụng trước khi xem xét việc tách một nút.
- `mini_sample_leaf`: Xác định số lượng lá tối thiểu cần thiết để tách một nút bên trong.

Các siêu tham số sau đây được sử dụng để tăng tốc độ của mô hình:

- `n_jobs`: Truyền đạt cho engine số lượng bộ xử lý được phép sử dụng. Nếu giá trị là 1, engine chỉ có thể sử dụng một bộ xử lý, nhưng nếu giá trị là -1, thì không có giới hạn.
- `random_state`: Kiểm soát tính ngẫu nhiên của mẫu. Mô hình sẽ luôn tạo ra cùng một kết quả nếu nó có giá trị xác định của trạng thái ngẫu nhiên và nếu nó được cung cấp cùng một siêu tham số và cùng một dữ liệu đào tạo.
- `oob_score`: OOB (Out Of the Bag) là phương pháp xác thực chéo rừng ngẫu nhiên. Trong phương pháp này, một phần ba mẫu không được sử dụng để đào tạo dữ liệu mà để đánh giá hiệu suất của nó.

3.3.2.2. Huấn luyện mô hình

Đọc các file train, test đã lưu từ tiền xử lý:

```
] X_train = pd.read_csv('X_train.csv')
] X_test = pd.read_csv('X_test.csv')
] y_train = pd.read_csv('y_train.csv')
] y_test = pd.read_csv('y_test.csv')
```



```
#importing the randomforest classifier from sklearn ensemble.
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# random forest model creation
rfc = RandomForestClassifier()
# training the model
rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
metrics.accuracy_score(y_test,rfc_predict)

0.7222222222222222
```

- `rfc = RandomForestClassifier()`: Tạo một instance của mô hình Random Forest.
- `rfc.fit(X_train, y_train)`: Huấn luyện mô hình trên tập dữ liệu `X_train` (các đặc trưng đầu vào) và `y_train` (nhãn mục tiêu).
- `rfc_predict = rfc.predict(X_test)`: Dự đoán nhãn mục tiêu cho tập dữ liệu kiểm tra `X_test` dựa trên mô hình đã được huấn luyện.
- `metrics.accuracy_score(y_test, rfc_predict)`: Tính toán và trả về độ chính xác của mô hình bằng cách so sánh giữa nhãn thực sự (`y_test`) và nhãn dự đoán (`rfc_predict`).

Độ chính xác là 72.2%.

Ta in ra ma trận:

```
# importing the classification report and confusion matrix from sklearn metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

# printing the confusion matrix statistics and classification report
print(confusion_matrix(y_test, rfc_predict))
print(classification_report(y_test, rfc_predict))

[[87 12]
 [43 56]]
      precision    recall  f1-score   support

     0       0.67       0.88       0.76         99
     1       0.82       0.57       0.67         99

 accuracy          0.72         198
 macro avg       0.75       0.72       0.72         198
 weighted avg    0.75       0.72       0.72         198
```



3.3.3. Mạng nơ ron nhân tạo

3.3.3.1. Khái niệm

Mạng nơ ron nhân tạo (Artificial Neural Network) là một loại hệ thống máy học sâu, được lấy cảm hứng từ cách thức hoạt động của bộ não con người.

Mạng nơ ron nhân tạo bao gồm nhiều nơ ron nhân tạo, được kết nối với nhau theo một cấu trúc phân lớp. Mỗi nơ ron nhân tạo là một hàm toán học, có chức năng thu thập và phân loại thông tin dựa theo các trọng số và ngưỡng.

Mạng nơ ron nhân tạo có thể học hỏi và thích ứng với các dữ liệu phi tuyến tính và phức tạp, giải quyết các bài toán khó như nhận dạng hình ảnh, giọng nói, văn

bản và dự báo.

Một mạng NR sẽ có 3 kiểu tầng:

- Tầng vào (input layer): Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
- Tầng ra (output layer): Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
- Tầng ẩn (hidden layer): Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng. Lưu ý rằng, một NR chỉ có 1 tầng vào và 1 tầng ra nhưng có thể có nhiều tầng ẩn.

Các hình tròn được gọi là các node, ngoài ra còn có hàm Loss dùng để đo lường sự khác biệt giữa giá trị dự đoán của mô hình và giá trị thực tế.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer – 1 (Không tính input layer).

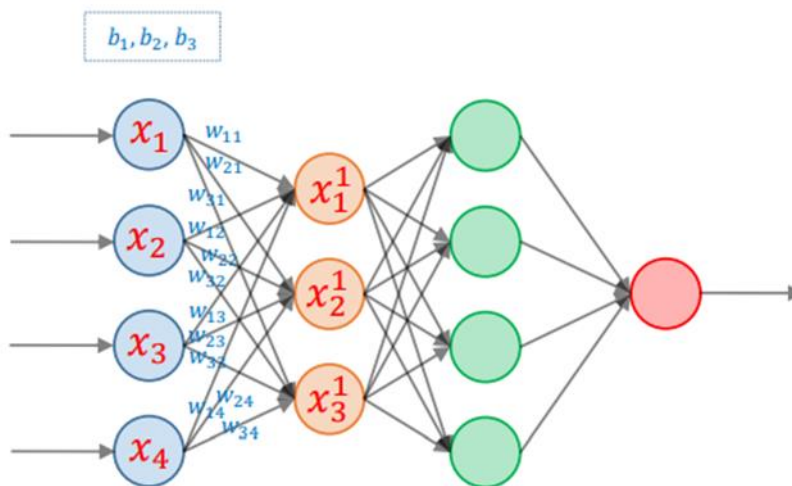
Thiết kế lớp ẩn:

- Weight(W): Mỗi kết nối giữa các neuron được điều chỉnh bằng trọng số, thể hiện mức độ quan trọng của đầu vào đối với kết quả của neuron
- Bias (b): Ngưỡng là một giá trị độc lập được thêm vào, cho phép điều chỉnh độ lệch của đầu ra của neuron.
- Activation function: một hàm phi tuyến như sigmoid, tanh, ReLU giúp mô hình học các mối liên hệ phức tạp hơn.

Phương trình lớp ẩn:

- $X(\text{input})$
- $\text{Output} = \text{activation}(XW+b)$

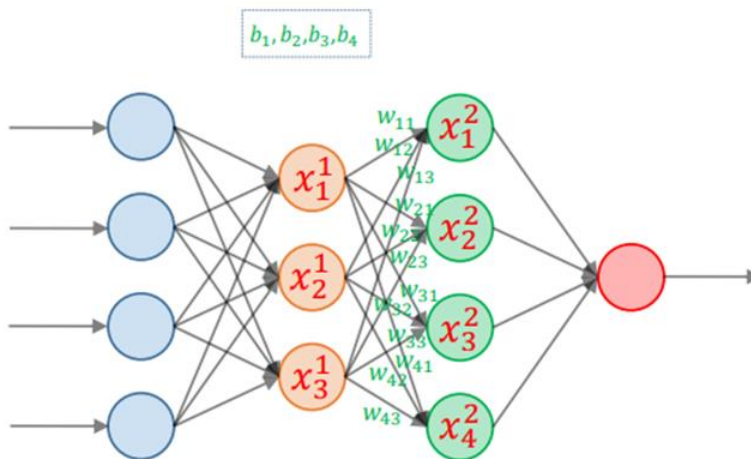
Ví dụ:



Áp dụng các trọng số và truyền dữ liệu đến lớp tiếp theo:

$$\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} = \text{Activation} \left(\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Tiếp tục:



$$\begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{bmatrix} = \text{Activation} \left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \cdot \begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \right)$$

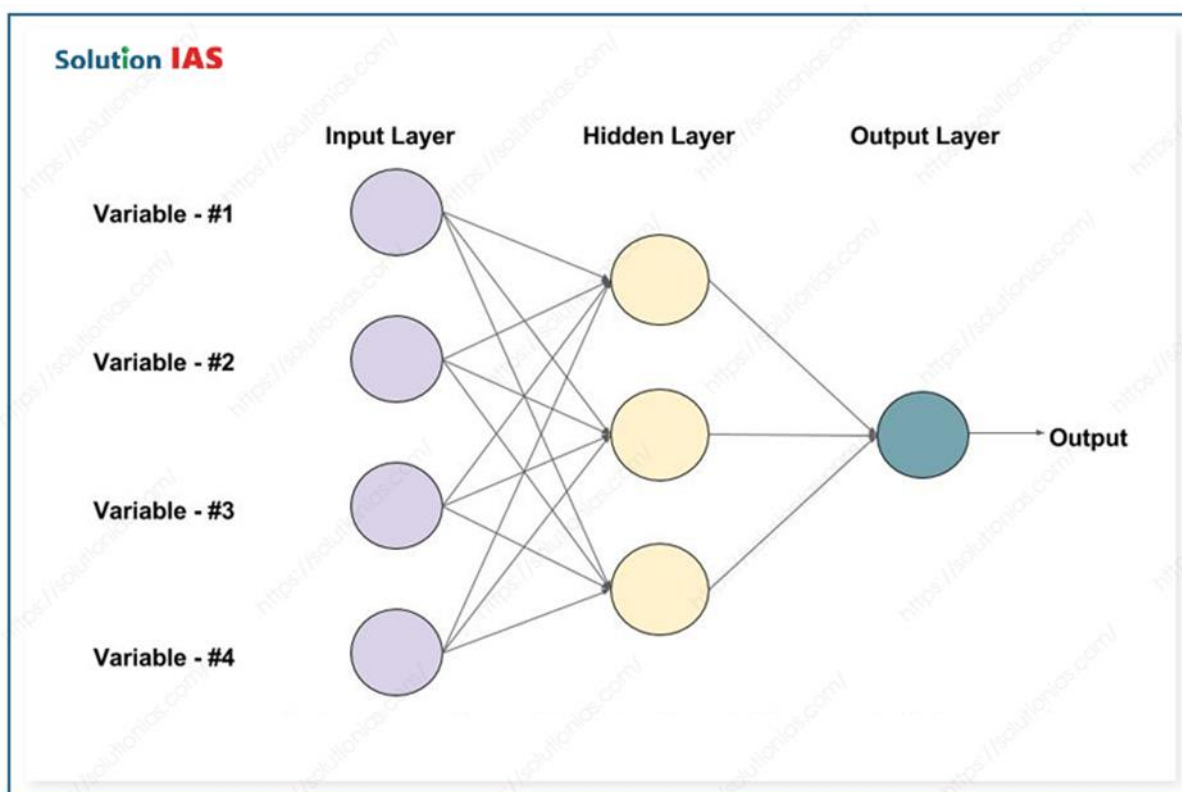
Mô hình mạng nơ ron nhân tạo

Có nhiều loại mô hình mạng nơ-ron nhân tạo khác nhau, tùy thuộc vào cách sắp xếp và kết nối các nơ-ron nhân tạo với nhau. Một số loại mô hình mạng nơ-ron nhân tạo phổ biến là:

Mạng nơ-ron truyền thẳng (feedforward neural network)

Là loại mạng nơ-ron nhân tạo đơn giản nhất, trong đó các nơ-ron nhân tạo được sắp xếp thành các lớp và chỉ có kết nối từ lớp trước sang lớp sau. Không có kết nối ngược lại hay giữa các nơ-ron trong cùng một lớp.

- Ưu điểm: đơn giản, dễ hiểu và dễ huấn luyện.
- Nhược điểm: không thể xử lý được các dữ liệu có tính thời gian hoặc có sự phụ thuộc lẫn nhau giữa các phần tử.



Ví dụ: perceptron, mạng nơ-ron tích chập (convolutional neural network), mạng nơ-ron đa lớp (multilayer perceptron).

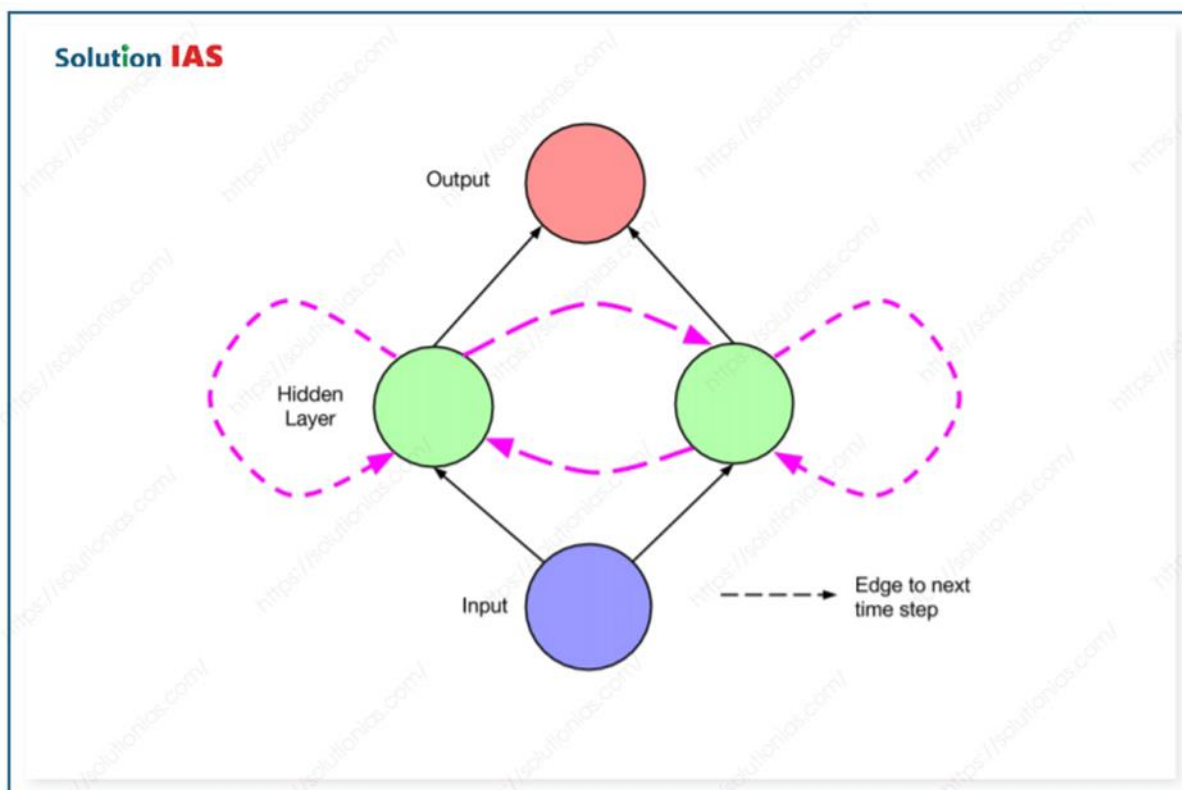
Mạng nơ-ron hồi quy (recurrent neural network)

Mạng nơ-ron hồi quy có khả năng ghi nhớ trạng thái trước đó và sử dụng chúng để ảnh hưởng đến trạng thái hiện tại. Có các kết nối ngược lại giữa các lớp hoặc giữa các bước thời gian. Thường được sử dụng cho các bài toán xử lý chuỗi hoặc dữ liệu có tính thời gian.

- Ưu điểm: có thể xử lý được các dữ liệu chuỗi hoặc có tính thời gian, bằng

cách sử dụng bộ nhớ ngắn hạn để lưu trữ trạng thái trước đó.

- Nhược điểm: khó huấn luyện, dễ bị vấn đề biến mất hoặc bùng nổ gradient.

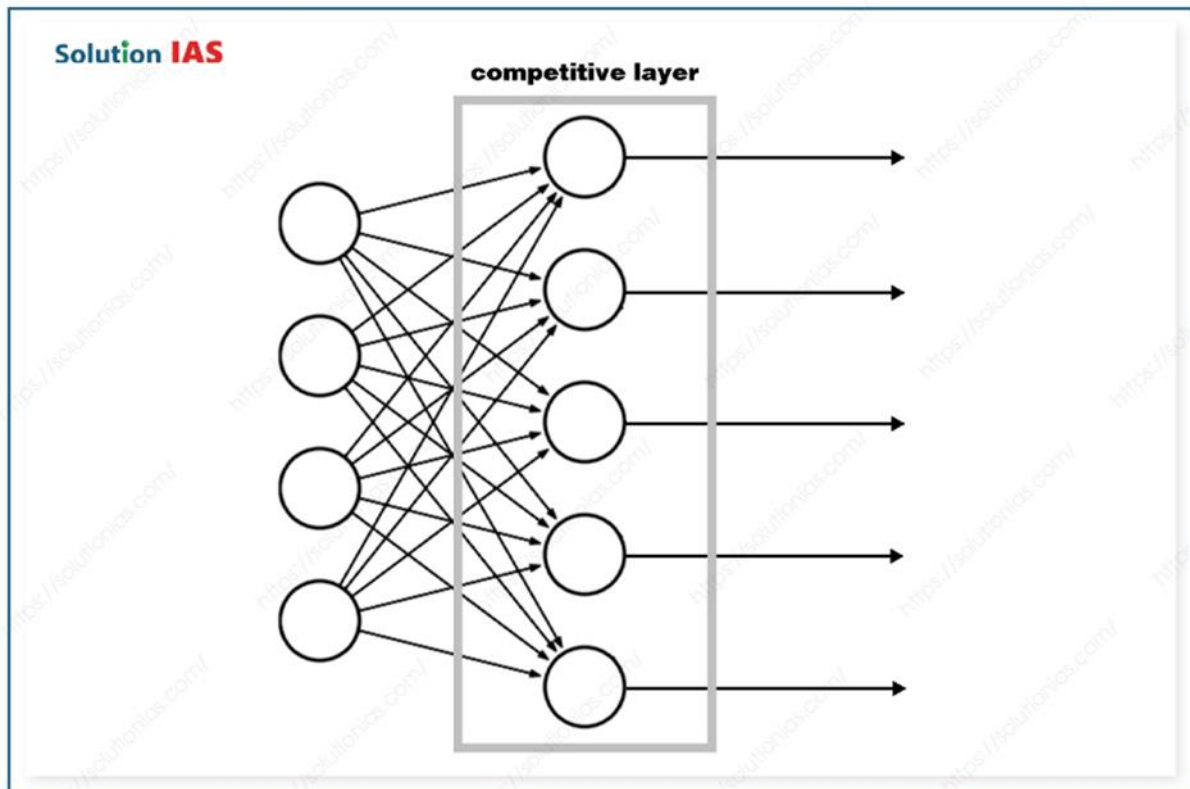


Ví dụ: mạng hồi quy đơn giản (simple recurrent network), mạng bộ nhớ ngắn hạn dài (long short-term memory), mạng cổng hồi quy (gated recurrent unit).

Mạng nơ-ron cạnh tranh (competitive neural network)

Là loại mạng nơ-ron nhân tạo có khả năng tự tổ chức và phân cụm dữ liệu vào các nhóm khác nhau. Có sự cạnh tranh giữa các nơ-ron để trở thành người chiến thắng khi nhận đầu vào. Thường được sử dụng cho các bài toán phân tích dữ liệu không giám sát.

- Ưu điểm: có thể tự tổ chức và phân cụm dữ liệu vào các nhóm khác nhau mà không cần nhãn.
- Nhược điểm: khó xác định số lượng và kích thước của các nhóm, cũng như khó diễn giải kết quả.



Ví dụ: bản đồ tự tổ chức (self-organizing map), mạng học cạnh tranh (learning vector quantization).

3.3.3.2. Huấn luyện mô hình

Khởi tạo mô hình

- import các thư viện cần thiết:

```
import tensorflow as tf
import pandas as pd
import numpy as np
```

Sử dụng thư viện keras: Keras là một thư viện mạng thần kinh mã nguồn mở được viết bằng Python. Nó có khả năng chạy trên TensorFlow, Microsoft Cognitive Toolkit, Theano hoặc MXNet. Được thiết kế để cho phép thử nghiệm nhanh với mạng lưới thần kinh sâu, nó tập trung vào việc thân thiện với người dùng, mô-đun và có thể mở rộng.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```


Đọc dữ liệu file đã tiền xử lý:

```
X_train = pd.read_csv('X_train.csv')
y_train = pd.read_csv('y_train.csv')
X_test = pd.read_csv('X_test.csv')
y_test = pd.read_csv('y_test.csv')
```

Tạo mô hình tuần tự (Sequential), thêm lớp đầu vào và các lớp tiếp, sử dụng các mô hình sau để biên dịch:

- optimizer = 'adam' sử dụng bộ tối ưu hóa Adam, một thuật toán tối ưu hóa phổ biến trong học sâu.
- loss = 'binary_crossentropy' chỉ định hàm mất mát (loss function) là entropy nhị phân, thường được sử dụng cho bài toán phân loại nhị phân.
- metrics = ['accuracy'] yêu cầu theo dõi độ chính xác (accuracy) trong quá trình huấn luyện và đánh giá mô hình.

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dropout(.2))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Đưa ra cái nhìn tổng quan về cấu trúc của mô hình mạng nơ-ron mà ta đã vừa xây dựng:

- Total params: 253: Tổng số tham số của mô hình là 253.
- Trainable params: 253: Tất cả các tham số đều có thể được huấn luyện (không có tham số không thể huấn luyện).
- Non-trainable params: 0: Không có tham số nào không thể huấn luyện.


```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12)	108
dense_1 (Dense)	(None, 8)	104
dropout (Dropout)	(None, 8)	0
dense_2 (Dense)	(None, 4)	36
dense_3 (Dense)	(None, 1)	5

Total params: 253 (1012.00 B)
 Trainable params: 253 (1012.00 B)
 Non-trainable params: 0 (0.00 B)

Đào tạo mô hình

```
model.fit(X_train, y_train, epochs = 300, batch_size=15, validation_data=(X_test, y_test))
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test loss: {loss}')
print(f'Test accuracy: {accuracy}')
```

- `model.fit (...)` là phương thức dùng để huấn luyện mô hình với dữ liệu đào tạo.
- `X_train`: Đây là dữ liệu đầu vào dùng để huấn luyện mô hình. `X_train` chứa các mẫu dữ liệu huấn luyện.
- `y_train`: Đây là nhãn mục tiêu tương ứng với `X_train`, `y_train` chứa các giá trị mục tiêu (labels) cho các mẫu dữ liệu huấn luyện.
- `epochs = 300`: Số lượng lần lặp lại qua toàn bộ dữ liệu huấn luyện. Một epoch là một lượt qua toàn bộ dữ liệu huấn luyện. Ở đây, mô hình sẽ được huấn luyện trong 300 epochs.
- `batch_size = 15`: Kích thước của từng lô dữ liệu (batch) được sử dụng trong mỗi lần cập nhật trọng số. Mỗi lô dữ liệu chứa 15 mẫu.
- `validation_data = (X_test, y_test)`: Cung cấp dữ liệu kiểm tra (validation data) để đánh giá mô hình sau mỗi epoch. `X_test` và `y_test` là dữ liệu và nhãn của tập kiểm tra. Điều này cho phép theo dõi hiệu suất của mô hình trên dữ liệu chưa thấy trong quá trình huấn luyện.
- `model.evaluate (...)` là phương thức dùng để đánh giá mô hình trên dữ liệu kiểm tra.

Kết quả sau huấn luyện:

```
Epoch 150/150
54/54 ----- 0s 3ms/step - accuracy: 0.6893 - loss: 0.5796 - val_accuracy: 0.7121 - val_loss: 1.0249
7/7 ----- 0s 2ms/step - accuracy: 0.7293 - loss: 1.0033
Test loss: 1.0248651504516602
Test accuracy: 0.7121211886405945
```

Độ chính xác là 71%

3.4 Giao diện người dùng

Import thư viện cần thiết:

```
import tkinter as tk
from tkinter import ttk, messagebox
```

Load các model đã lưu:

```
with open('logistic_model.pkl', 'rb') as file:
    LRModel = pickle.load(file)

with open('random_forest_model.pkl', 'rb') as file:
    RFModel = pickle.load(file)

from keras.models import load_model

noron_model = load_model('model.h5')
```

Sử dụng các mô hình đã tạo ở giai đoạn trước và đưa vào các biến mô hình LRModel, RFModel và noron_model.

Tạo cột dữ liệu khớp với các mô hình:

```
# Các cột dữ liệu phải khớp với dữ liệu đầu vào của mô hình
x = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
```

Hàm LR_predict (chuẩn đoán bằng hồi quy logistic):

Lấy dữ liệu từ bàn phím và chuẩn đoán dữ liệu đã nhập.

```
prediction = LRModel.predict(patient_data)
```

Nếu giá trị này bằng 1 thì mắc bệnh tiểu đường, bằng 0 thì không mắc bệnh tiểu đường.

```
def LR_predict():
    try:
        pregnancies = float(entry_pregnancies.get())
        glucose = float(entry_glucose.get())
        blood_pressure = float(entry_blood_pressure.get())
        skin_thickness = float(entry_skin_thickness.get())
        insulin = float(entry_insulin.get())
        bmi = float(entry_bmi.get())
        dpf = float(entry_dpf.get())
        age = float(entry_age.get())

        data = [pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dpf, age]
        patient_data = pd.DataFrame([data], columns=x)

        prediction = LRModel.predict(patient_data)

        if prediction[0] == 1:
            messagebox.showinfo("Kết quả", "Mắc bệnh tiểu đường")
        else:
            messagebox.showinfo("Kết quả", "Không mắc bệnh tiểu đường")
    except ValueError as e:
        messagebox.showerror("Lỗi", f"Vui lòng nhập đúng định dạng số.\n{e}")
    except Exception as e:
        messagebox.showerror("Lỗi", f"Có lỗi xảy ra: {e}")
```

Hàm *NR_predict* (chuẩn đoán bằng mạng nơ ron nhân tạo):

Lấy dữ liệu từ bàn phím và chuẩn đoán dữ liệu đã nhập.

`prediction = noron_model.predict(patient_data)`

Nếu giá trị này lớn hơn 0.5 thì mắc bệnh tiểu đường, thấp hơn hoặc bằng 0.5 thì không mắc bệnh tiểu đường.

```
def NR_predict():
    try:
        pregnancies = float(entry_pregnancies.get())
        glucose = float(entry_glucose.get())
        blood_pressure = float(entry_blood_pressure.get())
        skin_thickness = float(entry_skin_thickness.get())
        insulin = float(entry_insulin.get())
        bmi = float(entry_bmi.get())
        dpf = float(entry_dpf.get())
        age = float(entry_age.get())

        input_data = np.array([[pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dpf, age]])
        prediction = noron_model.predict(input_data)
        result = 'Mắc bệnh tiểu đường.' if prediction[0][0] > 0.5 else 'Không mắc bệnh tiểu đường.'

        messagebox.showinfo("Kết quả dự đoán", f'Xác suất mắc bệnh tiểu đường: {prediction[0][0]:.2f}\nDự đoán: {result}')
    except ValueError:
        messagebox.showerror("Lỗi", "Vui lòng nhập đúng định dạng số.")
    except Exception as e:
        messagebox.showerror("Lỗi", f"Có lỗi xảy ra: {e}")
```

Hàm *RF_predict* (chuẩn đoán bằng rừng ngẫu nhiên):

Lấy dữ liệu từ bàn phím và chuẩn đoán dữ liệu đã nhập.

`prediction = RFModel.predict(patient_data)`

Nếu giá trị này bằng 1 thì mắc bệnh tiểu đường, bằng 0 thì không mắc bệnh tiểu đường.

```
def RF_predict():
    try:
        pregnancies = float(entry_pregnancies.get())
        glucose = float(entry_glucose.get())
        blood_pressure = float(entry_blood_pressure.get())
        skin_thickness = float(entry_skin_thickness.get())
        insulin = float(entry_insulin.get())
        bmi = float(entry_bmi.get())
        dpf = float(entry_dpf.get())
        age = float(entry_age.get())

        input_data = np.array([[pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dpf, age]])
        prediction = RFModel.predict(input_data)[0]

        if prediction == 1:
            messagebox.showinfo("Kết quả", " Mắc bệnh tiểu đường.")
        else:
            messagebox.showinfo("Kết quả", " Không mắc bệnh tiểu đường.")
    except ValueError as e:
        messagebox.showerror("Lỗi", f"Vui lòng nhập đúng định dạng số.\n{e}")
    except Exception as e:
        messagebox.showerror("Lỗi", f"Có lỗi xảy ra: {e}")
```

Thiết kế giao diện:

```
# Tạo cửa sổ chính
root = tk.Tk()
root.title("Dự đoán tiểu đường")
root.geometry("400x600")
root.config(bg="#f0f0f0")

# Sử dụng Canvas để thêm hình nền
canvas = tk.Canvas(root, width=400, height=600)
canvas.pack(fill="both", expand=True)

# Tạo Gradient background
gradient = canvas.create_rectangle(0, 0, 400, 600, outline="", fill="lightblue")

frame_inputs = ttk.Frame(canvas, padding="10", style="TFrame")
frame_inputs.place(relx=0.5, rely=0.2, anchor="center")

# Các trường nhập liệu
labels = ['Bạn đã mang thai bao nhiêu lần', 'Mức glucose trong huyết tương (mg/dL)', 'Huyết áp tâm trương (mmHg)',
          'Độ dày nếp gấp da (mm)', 'Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)', 'Chỉ số khối cơ thể (BMI)',
          'Tiền sử mắc bệnh của gia đình (DPF)', 'Tuổi']

entries = []

for label in labels:
    row = ttk.Frame(frame_inputs)
    row.pack(pady=5, fill="x")
    ttk.Label(row, text=label, font=("Arial", 12, "bold")).pack(side="left", padx=5)
    entry = ttk.Entry(row, font=("Arial", 12))
    entry.pack(side="right", fill="x", expand=True)
    entries.append(entry)
```

```
# Gán sự kiện Enter để chuyển sang ô nhập liệu kế tiếp
def focus_next_widget(event):
    event.widget.tk_focusNext().focus()
    return "break"

for entry in entries:
    entry.bind("<Return>", focus_next_widget)

entry_pregnancies, entry_glucose, entry_blood_pressure, entry_skin_thickness, entry_insulin, entry_bmi, entry_dpf, entry_age = entries

# Nút dự đoán
button_frame = tk.Frame(root, bg="#f0f0f0")
button_frame.place(relx=0.5, rely=0.85, anchor="center")

LR_button = tk.Button(button_frame, text="Logistic", command=LR_predict, bg="#4CAF50", fg="ffffff", font=("Arial", 12, "bold"), padx=10, pady=5)
LR_button.grid(row=0, column=0, padx=10)

NR_button = tk.Button(button_frame, text="Neural Network", command=NR_predict, bg="#008CBA", fg="ffffff", font=("Arial", 12, "bold"), padx=10, pady=5)
NR_button.grid(row=0, column=1, padx=10)

RF_button = tk.Button(button_frame, text="Random Forest", command=RF_predict, bg="#f44336", fg="ffffff", font=("Arial", 12, "bold"), padx=10, pady=5)
RF_button.grid(row=0, column=2, padx=10)

# Chạy ứng dụng
root.mainloop()
```

Giao diện hoàn chỉnh:

Bạn đã mang thai bao nhiêu lần

Mức glucose trong huyết tương (mg/dL)

Huyết áp tâm trương (mmHg)

Độ dày nếp gấp da (mm)

Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)

Chỉ số khối cơ thể (BMI)

Tiền sử mắc bệnh của gia đình (DPF)

Tuổi

Logistic

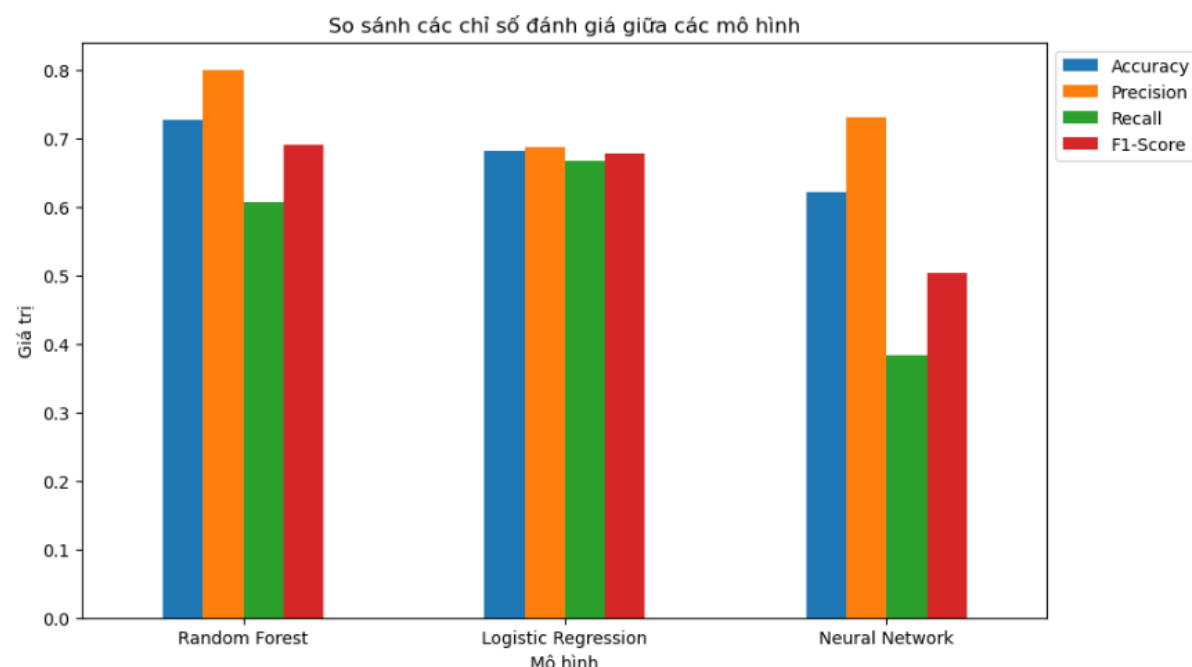
Neural Network

Random Forest

3.5 Thử nghiệm và cải tiến

3.5.1. So sánh các mô hình

7/7	0s 11ms/step				
	Model	Accuracy	Precision	Recall	F1-Score
0	Random Forest	0.727273	0.800000	0.606061	0.689655
1	Logistic Regression	0.681818	0.687500	0.666667	0.676923
2	Neural Network	0.621212	0.730769	0.383838	0.503311



1. Random Forest

Precision là cao nhất trong số ba mô hình, điều này cho thấy Random Forest có khả năng tốt trong việc giảm thiểu các kết quả dương tính giả (false positives).

Accuracy cũng khá cao, chỉ đứng sau Precision, cho thấy mô hình hoạt động ổn định và đúng đắn trong hầu hết các trường hợp.

Recall có phần thấp hơn so với Precision, điều này có thể gợi ý rằng mô hình có xu hướng bỏ sót một số lượng nhất định các mẫu dương tính thật sự (true positives).

F1-Score nằm ở mức trung bình giữa Precision và Recall, phản ánh một sự cân

bằng hợp lý giữa hai chỉ số này.

2. Logistic Regression

Cả Accuracy, Precision, Recall, và F1-Score đều có giá trị tương đối đồng đều và khá ổn định, không có chỉ số nào quá vượt trội so với các chỉ số khác.

Điều này cho thấy Logistic Regression là một mô hình tương đối ổn định và không có sự thiên lệch nhiều giữa việc xác định đúng các mẫu dương tính và âm tính.

Tuy nhiên, các chỉ số này đều thấp hơn một chút so với Random Forest, đặc biệt là về Precision và Accuracy, gợi ý rằng Logistic Regression có thể không phải là lựa chọn tốt nhất trong trường hợp này.

3. Neural Network

Precision là cao nhất trong số tất cả các chỉ số của Neural Network, tương tự như Random Forest, điều này cũng gợi ý rằng Neural Network có khả năng giảm thiểu các kết quả dương tính giả tốt.

Recall lại có xu hướng thấp, thậm chí thấp hơn so với Random Forest và Logistic Regression, điều này có nghĩa là mô hình này có thể bỏ sót nhiều mẫu dương tính thật sự hơn.

F1-Score không quá cao, cho thấy rằng mặc dù Precision tốt, nhưng do Recall thấp, nên sự cân bằng giữa Precision và Recall chưa thực sự tốt.

Accuracy nằm ở mức tương đối ổn định nhưng không quá nổi bật so với Random Forest.

Nhận xét:

Random Forest là mô hình tốt nhất, đặc biệt với Precision cao và Accuracy ổn định. Tuy nhiên, nếu mục tiêu của bạn là phát hiện tất cả các mẫu dương tính (tăng Recall), thì có lẽ cần cân nhắc cải thiện mô hình này.

Logistic Regression cho thấy một sự cân bằng giữa tất cả các chỉ số, nhưng không có chỉ số nào thực sự vượt trội.

Neural Network có Precision tốt nhưng cần cải thiện Recall để tránh bỏ sót các

mẫu quan trọng, điều này cũng sẽ giúp tăng F1-Score.

3.5.2. Cải tiến mô hình

3.5.2.1. Hồi quy logistics

Import thư viện

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Khởi tạo mô hình Logistic Regression
log_reg = LogisticRegression()

# Định nghĩa các giá trị siêu tham số cần thử nghiệm
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'],
    'max_iter': [100, 500, 1000]
}

# Khởi tạo GridSearchCV
grid_search = GridSearchCV(estimator=log_reg, param_grid=param_grid, cv=5, scoring='accuracy')

# Huấn luyện mô hình với các siêu tham số
grid_search.fit(X_train, y_train)

# In ra các siêu tham số tốt nhất
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-validation Score:", grid_search.best_score_)
```

Để tìm siêu tham số ta dùng phương pháp GridSearch. Thử nghiệm với param_grid ta tìm:

- C: Tham số điều chỉnh mức độ regularization. Giá trị nhỏ hơn làm tăng mức độ regularization. (mức độ này dùng để ngăn chặn overfitting)
- penalty: Loại penalty áp dụng (L1, L2).
- solver: Thuật toán tối ưu hóa.
- max_iter: Số lần lặp tối đa cho thuật toán tối ưu hóa.

Khởi tạo GridSearch:

- estimator: là mô hình cần tối ưu hóa, trong trường hợp này là log_reg.
- param_grid: là tập hợp các giá trị siêu tham số thử nghiệm.

- cv: là số lần chia cắt dữ liệu để thực hiện cross-validation. cv=5, nghĩa là dữ liệu sẽ được chia làm 5 phần, mỗi phần sẽ lần lượt được dùng làm tập kiểm tra, 4 phần còn lại sẽ làm tập huấn luyện.
- scoring: Phương pháp đánh giá mô hình, ta sử dụng accuracy, tức là độ chính xác của mô hình sẽ được dùng để đánh giá.

Ta tìm được các siêu tham số cho mô hình:

```
Best Hyperparameters: {'C': 1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
Best Cross-validation Score: 0.7593555900621118
```

```
from sklearn.linear_model import LogisticRegression
best_params = grid_search.best_params_
# Khởi tạo mô hình Logistic Regression với các siêu tham số tối ưu
LRModel_Hyper = LogisticRegression(C=best_params['C'],
                                   penalty=best_params['penalty'],
                                   solver=best_params['solver'],
                                   max_iter=best_params['max_iter']
                                   )

# Huấn luyện mô hình trên toàn bộ tập huấn luyện
LRModel_Hyper.fit(X_train, y_train)
```

LogisticRegression

LogisticRegression(C=1, solver='liblinear')

Sau đó ta dùng các siêu tham số để huấn luyện lại mô hình.

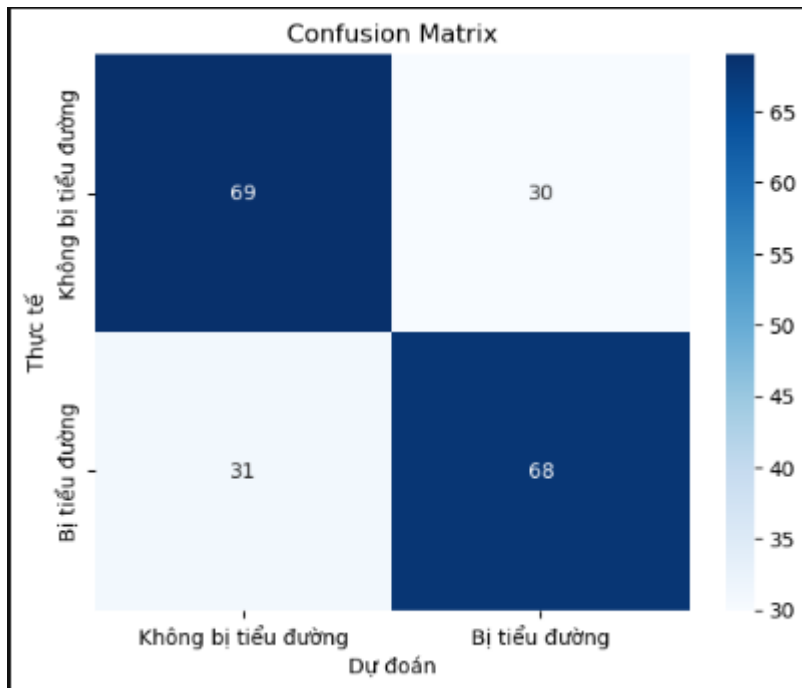
```
y_pred_Hyper = LRModel_Hyper.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_Hyper)
precision = precision_score(y_test, y_pred_Hyper)
recall = recall_score(y_test, y_pred_Hyper)
f1 = f1_score(y_test, y_pred_Hyper)
roc_auc = roc_auc_score(y_test, LRModel.predict_proba(X_test)[: , 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
print(f"ROC-AUC: {roc_auc}")

Accuracy: 0.6919191919191919
Precision: 0.6938775510204082
Recall: 0.6868686868686869
F1-Score: 0.6903553299492385
ROC-AUC: 0.7951229466380982
```

Độ chính xác tăng lên đến 69% tức là hiệu quả hơn so với ban đầu là 1%.



Ta lưu file đã tối ưu để dễ dàng sử dụng:

```
with open('logistic_model.pkl', 'wb') as file:
    pickle.dump(LRModel_Hyper, file)
```

3.5.2.2. Random Forest

Ở mô hình này ta cũng dùng GridSearch để tìm siêu tham số.

Import thư viện và khởi tạo các định nghĩa:

```
: from sklearn.model_selection import GridSearchCV
: from sklearn.ensemble import RandomForestClassifier
: from sklearn.datasets import load_iris
: from sklearn.model_selection import train_test_split

: # Định nghĩa mô hình
: rf = RandomForestClassifier(random_state=42)

: # Định nghĩa lưới các tham số để thử nghiệm
: param_grid = {
:     'n_estimators': [50, 100, 200],
:     'max_depth': [5, 10, 20, 30],
:     'min_samples_split': [2, 5, 10],
:     'min_samples_leaf': [1, 2, 4]
: }
```

Khởi tạo và huấn luyện mô hình với GridSearch:

```
# Khởi tạo GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Huấn luyện mô hình với GridSearchCV
grid_search.fit(X_train, y_train)
```

Fitting 5 folds for each of 108 candidates, totalling 540 fits

```
> GridSearchCV ① ?
> estimator: RandomForestClassifier
  > RandomForestClassifier ?
```

- estimator: Mô hình bạn muốn tối ưu hóa, ở đây là rf (RandomForestClassifier).
- param_grid: Lưới các tham số mà bạn đã định nghĩa trước đó để thử nghiệm.
- cv=5: Số lần cross-validation (5-fold CV), nghĩa là dữ liệu sẽ được chia thành 5 phần, mỗi phần sẽ lần lượt làm tập kiểm tra trong khi các phần còn lại làm tập huấn luyện.
- n_jobs=-1: Sử dụng tất cả các CPU cores có sẵn để tăng tốc quá trình tìm kiếm.
- verbose=2: Độ chi tiết khi in thông tin ra màn hình về quá trình thực hiện, giá trị 2 là mức độ vừa phải.
- scoring='accuracy': Phương pháp đánh giá là độ chính xác của mô hình.

Tìm ra siêu tham số và xem độ chính xác:

```
# Tham số tốt nhất
print("Best Parameters:", grid_search.best_params_)

# Độ chính xác tốt nhất
print("Best Accuracy:", grid_search.best_score_)

Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Accuracy: 0.8367158385093167

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Đánh giá mô hình
from sklearn import metrics
from sklearn.metrics import accuracy_score
print("Test Accuracy:", accuracy_score(y_test, y_pred))

Test Accuracy: 0.7323232323232324
```

Huấn luyện lại bằng các siêu tham số đã tìm được:

```
rfc1=RandomForestClassifier(random_state=42, n_estimators= 200, max_depth=20, min_samples_leaf= 2,min_samples_split = 2)

rfc1.fit(X_train, y_train)
```

RandomForestClassifier

RandomForestClassifier(max_depth=20, min_samples_leaf=2, n_estimators=200, random_state=42)

```
pred=rfc1.predict(X_test)

pred

array([1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0],
      dtype=int64)

print("=== Confusion Matrix ===")
print(confusion_matrix(y_test,pred))
print('\n')
print("=== Classification Report ===")
print(classification_report(y_test,pred))
print('\n')
#print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
print("Accuracy for Random Forest on CV data: ",metrics.accuracy_score(y_test,pred))
```

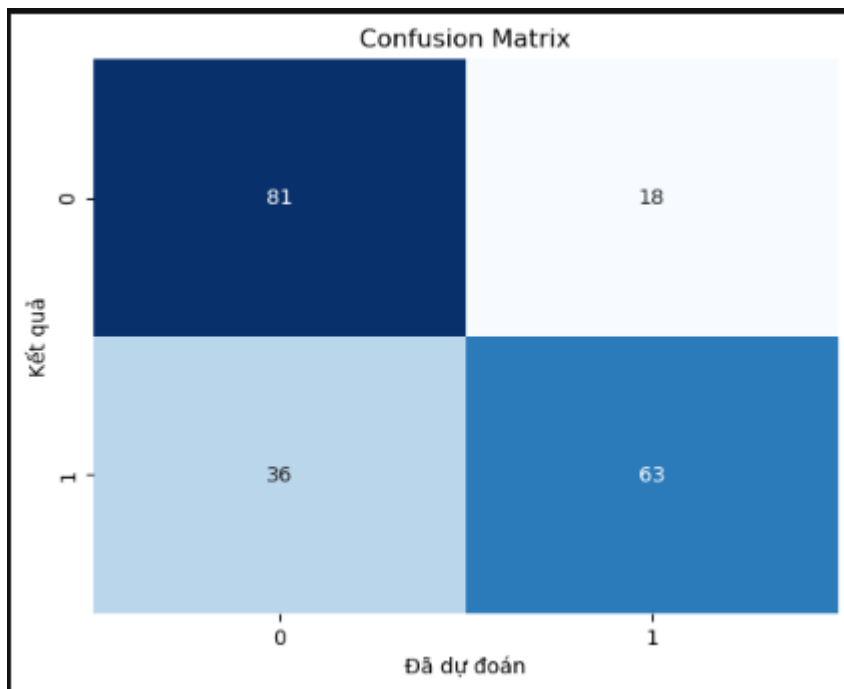
```

=== Classification Report ===

```

	precision	recall	f1-score	support
0	0.69	0.82	0.75	99
1	0.78	0.64	0.70	99
accuracy			0.73	198
macro avg	0.74	0.73	0.72	198
weighted avg	0.74	0.73	0.73	198

Độ chính xác là 73% cao hơn ban đầu 1%.



```

with open('Random_Forest.pkl', 'wb') as file:
    pickle.dump(rfcl, file)

```

Lưu file để dễ dàng sử dụng.

3.5.2.3. Mạng noron nhân tạo

Đầu tiên ta tạo mô hình với các tham số có thể tinh chỉnh

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import ParameterGrid

def create_model(num_units1=12, num_units2=8, dropout_rate=0.2, learning_rate=0.001):
    model = Sequential()
    model.add(Dense(num_units1, input_dim=8, activation='relu'))
    model.add(Dense(num_units2, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

```

Định nghĩa hàm 'create_model':

- num_units1: Số lượng nơ-ron trong lớp Dense đầu tiên.
- num_units2: Số lượng nơ-ron trong lớp Dense thứ hai.
- dropout_rate: Tỷ lệ dropout để áp dụng cho lớp Dropout.
- learning_rate: Tốc độ học (learning rate) của trình tối ưu hóa Adam.
- model = Sequential(): Khởi tạo một mô hình tuần tự.
- model.add(Dense(num_units1, input_dim=8, activation='relu')): Thêm một lớp Dense với num_units1 nơ-ron, hàm kích hoạt ReLU và đầu vào có 8 đặc trưng.
- model.add(Dense(num_units2, activation='relu')): Thêm một lớp Dense thứ hai với num_units2 nơ-ron và hàm kích hoạt ReLU.
- model.add(Dropout(dropout_rate)): Thêm một lớp Dropout với tỷ lệ dropout được chỉ định.
- model.add(Dense(4, activation='relu')): Thêm một lớp Dense với 4 nơ-ron và hàm kích hoạt ReLU.
- model.add(Dense(1, activation='sigmoid')): Thêm lớp đầu ra với một nơ-ron và hàm kích hoạt sigmoid, phù hợp cho bài toán phân loại nhị phân.

Sử dụng tìm kiếm lưới (grid search) để tìm ra các tham số tốt nhất cho mô hình

mạng nơ-ron được định nghĩa trong hàm `create_model`:

Định nghĩa các tham số để thử nghiệm, tạo grid và khởi tạo các biến để theo dõi kết quả:

```
import numpy as np
# Định nghĩa các giá trị tham số để thử nghiệm
param_grid = {
    'num_units1': [8, 12, 16, 20, 24],
    'num_units2': [4, 8, 12, 16, 20, 24],
    'dropout_rate': [0.2, 0.3, 0.4, 0.5],
    'learning_rate': [0.001, 0.01, 0.1]
}
# Tạo grid
grid = ParameterGrid(param_grid)

best_score = -np.inf
best_params = None
best_model = None
```

- `grid = ParameterGrid(param_grid)`: Tạo ra tất cả các tổ hợp có thể của các tham số trong `param_grid`.
- `best_score`: Khởi tạo biến để lưu trữ điểm số tốt nhất tìm được. Ban đầu được đặt là âm vô cực (`-np.inf`) để đảm bảo bất kỳ điểm số nào cũng lớn hơn giá trị khởi tạo.
- `best_params`: Khởi tạo biến để lưu trữ tham số tốt nhất tìm được.
- `best_model`: Khởi tạo biến để lưu trữ mô hình tốt nhất tìm được.

Tiến hành chạy thử nghiệm các tham số để cho ra bộ tham số tốt nhất cho mô hình mạng nơ-ron.

```
# Thử nghiệm với các tham số
for params in grid:
    model = create_model(num_units1=params['num_units1'],
                        num_units2=params['num_units2'],
                        dropout_rate=params['dropout_rate'],
                        learning_rate=params['learning_rate'])

    history = model.fit(X_train, y_train, epochs=150, batch_size=15, validation_data=(X_test, y_test), verbose=0)
    score = model.evaluate(X_test, y_test, verbose=0)[1]

    if score > best_score:
        best_score = score
        best_params = params
        best_model = model

print(f'Best score: {best_score}')
print(f'Best parameters: {best_params}')
```

- for params in grid: Lặp qua từng tổ hợp tham số trong grid.
- model = create_model(...): Tạo mô hình với các tham số hiện tại từ lưới.
- history = model.fit(...): Huấn luyện mô hình với dữ liệu huấn luyện (X_train, y_train) trong 150 epochs và kích thước batch là 15. validation_data là dữ liệu kiểm tra - (X_test, y_test), và verbose=0 để không hiển thị thông tin chi tiết trong quá trình huấn luyện.
- score = model.evaluate(X_test, y_test, verbose=0)[1]: Đánh giá mô hình trên dữ liệu kiểm tra và lấy điểm số chính xác (accuracy) từ kết quả.
- if score > best_score: Nếu điểm số hiện tại tốt hơn điểm số tốt nhất đã tìm được, cập nhật best_score, best_params, và best_model.

KẾT QUẢ THU ĐƯỢC:


```
Best score: 0.747474730014801
Best parameters: {'dropout_rate': 0.2, 'learning_rate': 0.001, 'num_units1': 8, 'num_units2': 24}

best_model.summary()

Model: "sequential_6"
```

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 8)	72
dense_25 (Dense)	(None, 24)	216
dropout_6 (Dropout)	(None, 24)	0
dense_26 (Dense)	(None, 4)	100
dense_27 (Dense)	(None, 1)	5

```

Total params: 1,181 (4.62 KB)
Trainable params: 393 (1.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 788 (3.08 KB)

```

Với độ chính xác là 75%

Tiến hành lưu mô hình

```
best_model.save('model.h5')
```


3.5.3. Thử nghiệm

Mẫu 1: 3-88-58-24.8-0.267-22-0

Với hồi quy logistic:

Mang thai bao nhiêu lần	3
Mức glucose trong huyết tương (mg/dL)	88
Huyết áp tâm trương (mmHg)	58
Độ dày nếp gấp da (mm)	11
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	54
Chỉ số khối cơ thể (BMI)	24.8
Tiền sử mắc bệnh của gia đình (DPF)	0.267
Tuổi	22

Kết quả

 Không mắc bệnh tiểu đường

OK

Logistic


Neural Network

Random Forest

Với Random Forest:

Mang thai bao nhiêu lần	3
Mức glucose trong huyết tương (mg/dL)	88
Huyết áp tâm trương (mmHg)	58
Độ dày nếp gấp da (mm)	11
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	54
Chỉ số khối cơ thể (BMI)	24.8
Tiền sử mắc bệnh của gia đình (DPF)	0.267
Tuổi	22

Kết quả

 Không mắc bệnh tiểu đường.

OK

Logistic

Neural Network

Random Forest

Với Mạng nơ ron nhân tạo:

Mang thai bao nhiêu lần	3
Mức glucose trong huyết tương (mg/dL)	88
Huyết áp tâm trương (mmHg)	58
Độ dày nếp gấp da (mm)	11
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	54
Chỉ số khối cơ thể (BMI)	24.8
Tiền sử mắc bệnh của gia đình (DPF)	0.267
Tuổi	22

Kết quả dự đoán

i Xác suất mắc bệnh tiểu đường: 0.26
Dự đoán: Không mắc bệnh tiểu đường.

OK

Logistic

Neural Network

Random Forest

Mẫu 2: 4-111-72-47-207-37.1-1.39-56-1

Với Hồi quy Logistic:

Mang thai bao nhiêu lần	4
Mức glucose trong huyết tương (mg/dL)	111
Huyết áp tâm trương (mmHg)	72
Độ dày nếp gấp da (mm)	47
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	207
Chỉ số khối cơ thể (BMI)	37.1
Tiền sử mắc bệnh của gia đình (DPF)	1.39
Tuổi	56

Kết quả

i Mắc bệnh tiểu đường

OK

Logistic

Neural Network

Random Forest

Với Random forest:

Mang thai bao nhiêu lần	4
Mức glucose trong huyết tương (mg/dL)	111
Huyết áp tâm trương (mmHg)	72
Độ dày nếp gấp da (mm)	47
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	207
Chỉ số khối cơ thể (BMI)	37.1
Tiền sử mắc bệnh của gia đình (DPF)	1.39
Tuổi	56

Kết quả

i Mắc bệnh tiểu đường.

OK

Logistic

Neural Network

Random Forest

Với Mạng nơ ron nhân tạo:

Mang thai bao nhiêu lần	4
Mức glucose trong huyết tương (mg/dL)	111
Huyết áp tâm trương (mmHg)	72
Độ dày nếp gấp da (mm)	47
Mức insulin trong huyết thanh (2 giờ sau kiểm tra glucose)	207
Chỉ số khối cơ thể (BMI)	37.1
Tiền sử mắc bệnh của gia đình (DPF)	1.39
Tuổi	56

Kết quả dự đoán

Xác suất mắc bệnh tiểu đường: 0.53
Dự đoán: Mắc bệnh tiểu đường.

OK

Logistic

Neural Network

Random Forest

Chương 4. Tác động dự kiến

4.1 Thành tựu và lợi ích

Phát triển mô hình AI để chẩn đoán bệnh tiểu đường có thể mang lại nhiều thành

tự và lợi ích đáng kể cho cả lĩnh vực y tế và cộng đồng.

Thành tựu:

1. Tăng Cường Chính Xác Trong Chẩn Đoán:

AI có khả năng phân tích các dữ liệu y tế (như hình ảnh y khoa, xét nghiệm máu, và dữ liệu lâm sàng) với độ chính xác cao. Điều này giúp cải thiện khả năng phát hiện bệnh tiểu đường sớm và chính xác hơn.

2. Phát Hiện Sớm và Dự Đoán Rủi Ro:

Các mô hình AI có thể phân tích các yếu tố nguy cơ và xu hướng từ dữ liệu bệnh nhân để phát hiện sớm các dấu hiệu tiểu đường, từ đó dự đoán nguy cơ mắc bệnh cho những người có nguy cơ cao.

3. Tăng Cường Hiệu Quả Quản Lý Dữ Liệu:

AI giúp tổ chức và phân tích khối lượng dữ liệu lớn một cách nhanh chóng, giúp bác sĩ và nhà nghiên cứu có cái nhìn toàn diện hơn về tình trạng bệnh nhân và xu hướng dịch tễ học.

Lợi ích:

1. Cải Thiện Sức Khỏe Cộng Đồng:

Phát hiện và điều trị sớm bệnh tiểu đường giúp giảm tỷ lệ biến chứng và tử vong, từ đó nâng cao chất lượng cuộc sống và giảm gánh nặng bệnh tật cho cộng đồng.

2. Tiết Kiệm Chi Phí Y Tế:

Việc phát hiện bệnh sớm và điều trị hiệu quả có thể giảm chi phí điều trị dài hạn và các biến chứng nghiêm trọng, giúp tiết kiệm ngân sách y tế.

3. Nâng Cao Hiệu Quả Nghiên Cứu:

AI giúp phân tích dữ liệu nghiên cứu nhanh chóng, từ đó đẩy nhanh tiến trình phát triển các phương pháp điều trị mới và các loại thuốc mới cho bệnh tiểu đường.

4. Tăng Cường Giáo Dục và Nhận Thức:

Các hệ thống AI có thể cung cấp thông tin giáo dục và hướng dẫn về bệnh tiểu đường cho bệnh nhân và cộng đồng, nâng cao nhận thức và giúp phòng ngừa bệnh

tật.

Nhìn chung, sự phát triển và áp dụng các mô hình AI trong chẩn đoán bệnh tiểu đường không chỉ cải thiện hiệu quả chẩn đoán và điều trị mà còn có tiềm năng to lớn trong việc nâng cao sức khỏe cộng đồng và tối ưu hóa hệ thống y tế.

4.2 Những cải tiến trong tương lai

Sự phát triển mô hình AI trong chẩn đoán bệnh tiểu đường đang diễn ra tốt và có thể tiếp tục tiến bộ mạnh mẽ trong tương lai. Dưới đây là những cải tiến có thể trong tương lai:

1. Tăng Cường Tính Chính Xác và Độ Tin Cậy

- **Sử Dụng Dữ Liệu Đầu Vào Đa Dạng:** Tích hợp nhiều loại dữ liệu hơn như hình ảnh y khoa, xét nghiệm gen, và dữ liệu sinh học để nâng cao độ chính xác trong chẩn đoán.
- **Mô Hình Tinh Vi:** Phát triển các mô hình học sâu (deep learning) phức tạp hơn để cải thiện khả năng phân tích và nhận diện các mẫu dữ liệu tinh vi hơn.

2. Cải Tiến Về Dự Đoán và Phát Hiện Sớm

- **AI Tinh Chỉnh Để Phát Hiện Sớm:** Sử dụng AI để phân tích dữ liệu từ các giai đoạn tiền tiểu đường và các yếu tố nguy cơ, giúp phát hiện bệnh sớm hơn trước khi xuất hiện triệu chứng rõ ràng.
- **Dự Đoán Cá Nhân Hóa:** Phát triển các mô hình dự đoán cá nhân hóa dựa trên lịch sử y tế, lối sống, và dữ liệu gen của từng người để cung cấp dự đoán chính xác hơn về nguy cơ mắc bệnh.

3. Tăng Cường Tính Cá Nhân Hóa Trong Điều Trị

- **Kế Hoạch Điều Trị Tinh Vi:** AI có thể giúp tạo ra các kế hoạch điều trị và chế độ dinh dưỡng được cá nhân hóa dựa trên dữ liệu sức khỏe cụ thể của từng bệnh nhân.
- **Theo Dõi Và Điều Chỉnh Liên Tục:** Sử dụng thiết bị đeo và ứng dụng di động tích hợp AI để theo dõi liên tục các chỉ số sức khỏe và điều

chính kế hoạch điều trị theo thời gian thực.

4. Nâng Cao Giáo Dục và Nhận Thức

- **Chương Trình Đào Tạo AI Cho Y Bác Sĩ:** Cung cấp đào tạo cho các bác sĩ và nhân viên y tế về cách sử dụng các công cụ AI hiệu quả trong chẩn đoán và điều trị bệnh tiểu đường.
- **Tăng Cường Nhận Thức Cộng Đồng:** Sử dụng AI để phát triển các chương trình giáo dục sức khỏe cộng đồng, giúp nâng cao nhận thức về bệnh tiểu đường và các biện pháp phòng ngừa.

5. Tối Ưu Hóa Quản Lý Dữ Liệu

- **Phân Tích Dữ Liệu Lớn (Big Data):** Phát triển các công cụ AI mạnh mẽ để phân tích và khai thác dữ liệu lớn từ các nghiên cứu lâm sàng và hồ sơ bệnh nhân, cung cấp cái nhìn sâu hơn về bệnh tiểu đường.
- **Học Máy Hợp Tác:** Tăng cường hợp tác giữa các hệ thống học máy để cải thiện khả năng tổng hợp thông tin từ nhiều nguồn dữ liệu khác nhau.

Những cải tiến này không chỉ giúp nâng cao khả năng chẩn đoán và điều trị bệnh tiểu đường mà còn mở ra cơ hội mới trong việc quản lý bệnh và cải thiện chất lượng cuộc sống cho bệnh nhân.

Chương 5. Đánh giá và nhận xét của thành viên nhóm

Chương 6. Giảng viên nhận xét

TÀI LIỆU THAM KHẢO

- [1] <https://www.kaggle.com/datasets/mathchi/diabetes-data-set> lần cuối truy cập 22/08/2024
- [2] <https://www.sciencedirect.com/science/article/abs/pii/S0045790613001778?via%3Dihub> lần cuối truy cập 22/08/2024
- [3] <https://github.com/Prajwal10031999/Diabetes-Prediction-using-Logistic-Regression> lần cuối truy cập 22/08/2024
- [4] <https://github.com/fatimaAfzaal/Diabetes-Prediction-Project-Using-Random-Forest> lần cuối truy cập 22/08/2024
- [5] <https://github.com/fatimaAfzaal/Diabetes-Prediction-Project-Using-Random-Forest> lần cuối truy cập 22/08/2024
- [6] <https://github.com/akanshu11121/Diabetes-Detection-using-Neural-Network/blob/master/Models/Diabetes%20Detection%20using%20Neural%20Network.ipynb> lần cuối truy cập 22/08/2024
- [7] https://github.com/AaronCosmos/wdcnn_bearing_fault_diagnosis/blob/master/main.py lần cuối truy cập 22/08/2024
- [8] <https://timtailieu.vn/tai-lieu/chuong-1-tien-xu-ly-du-lieu-40231/> lần cuối truy cập 22/08/2024
- [9] <https://viblo.asia/p/tong-quan-ve-artificial-neural-network-1VgZvwYrlAw> lần cuối truy cập 22/08/2024
- [10] <https://machinelearningcoban.com/2017/06/15/pca/> lần cuối truy cập 22/08/2024

CÁC THUẬT NGỮ