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

Báo cáo bài tập lớn

Khai Phá Dữ Liệu

Chủ đề

Tìm hiểu và xây dựng hệ thống dự đoán trụy tim

Giảng viên hướng dẫn: Đỗ Thanh Thái

Danh sách thành viên

Họ và tên	**MSSV**
Hồ Thanh Nhã	2212345
Trần Thế Nhân	2212383
Đặng Minh Nhật	2212387

Tp. Hồ Chí Minh, Tháng 9/2025

Work division table

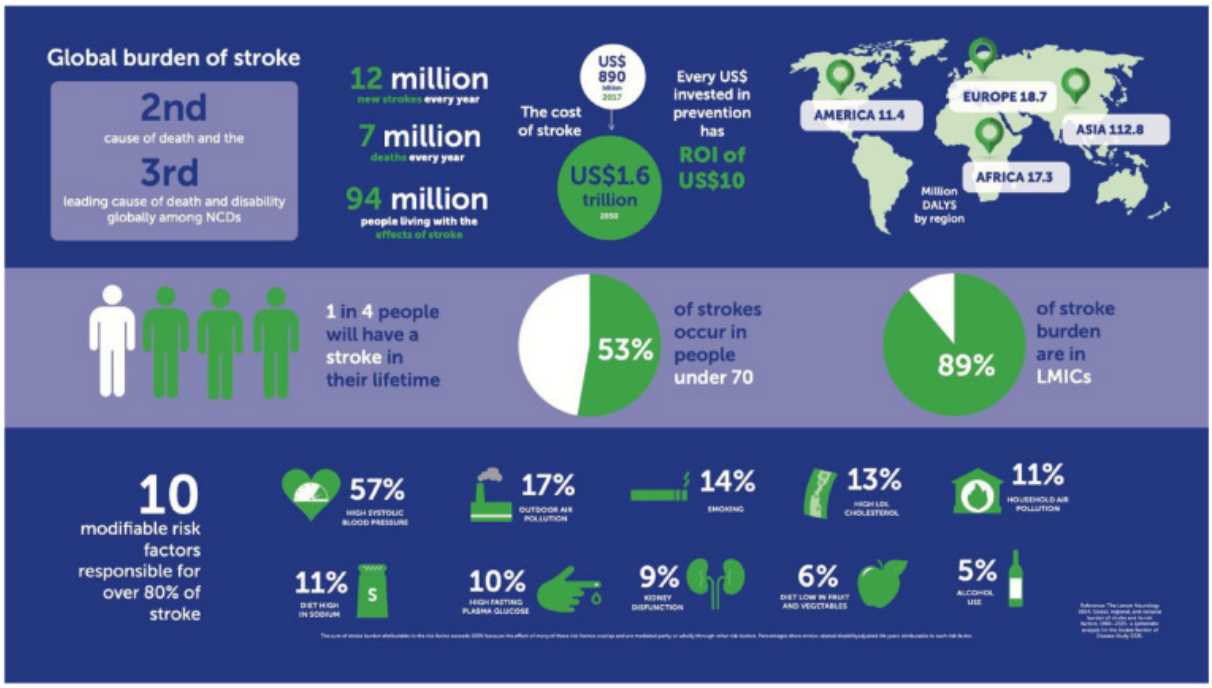
Full name	Student ID	Assigned works	Completion rate
Hồ Thanh Nhã	2212345		100%
Đặng Minh Nhật	2212387		100%
Trần Thế Nhân	2212383		100%

Mục lục

1 Giới thiệu

1.1 Bối cảnh và động lực phát triển

Đột quỵ là một trong những nguyên nhân hàng đầu gây tử vong và tàn tật trên toàn thế giới. Theo Tổ chức Y tế Thế giới (WHO), mỗi năm có khoảng 15 triệu người bị đột quỵ, trong đó 5 triệu người tử vong và 5 triệu người bị tàn tật vĩnh viễn [1]. Những con số này cho thấy mức độ nghiêm trọng của đột quỵ đối với sức khỏe cộng đồng. Thực tế, đột quỵ được xếp hạng là nguyên nhân gây tử vong đứng thứ hai trên thế giới và là nguyên nhân hàng đầu gây tàn tật kéo dài [2] .



Hình 1: *Dữ liệu từ World Stroke Organization (WSO) Global Stroke Fact Sheet 2025*

Tại Việt Nam, tỷ lệ mắc đột quỵ đang có xu hướng gia tăng, đặc biệt ở nhóm tuổi trung niên và cao tuổi. Theo số liệu từ Báo cáo Gánh nặng bệnh tật toàn cầu năm 2019, Việt Nam ghi nhận tới 135.999 ca tử vong do đột quỵ, đứng đầu trong nhóm các bệnh lý tim mạch. Tỷ lệ mắc mới ước tính khoảng 222 ca trên 100.000 dân mỗi năm, trong khi tỷ lệ hiện mắc lên đến 1.541 trên 100.000 dân, thuộc nhóm cao nhất khu vực Đông Nam Á. [3]

Phát hiện sớm nguy cơ đột quỵ đóng vai trò then chốt trong phòng ngừa và điều trị hiệu quả. Tuy nhiên, việc sàng lọc thủ công dựa trên kinh nghiệm lâm sàng thường tốn kém thời gian và nguồn lực y tế, đồng thời có thể bỏ sót các trường hợp nguy cơ cao. Hay việc nhận biết thông các dấu hiệu ban đầu dựa theo phương pháp **F.A.S.T** vẫn còn khá kém hiệu quả so với thực tế, những biểu hiện này thường bị nhầm lẫn hoặc bỏ qua, khiến cơ hội can thiệp sớm bị bỏ lỡ.

Trong bối cảnh đó, ứng dụng Machine Learning (ML) và Data Mining vào y tế đã mở ra hướng tiếp cận mới: xây dựng các mô hình dự đoán tự động dựa trên dữ liệu y tế và nhân khẩu học.

[1] Stroke, Cerebrovascular accident - https://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/
[2] World Stroke Organization: Global Stroke Fact Sheet 2025 - https://pmc.ncbi.nlm.nih.gov/articles/PMC11786524/
[3] Việt Nam thuộc nhóm nước có tỷ lệ đột quỵ cao nhất Đông Nam Á - https://moh.gov.vn/su-kien-y-te-noi-bat/-/asset_publisher//8EeXRtRENhb6/content/viet-nam-thuoc-nhom-nuoc-co-ty-le-ot-quy-cao-nhat-ong-nam-a

Với sự phát triển mạnh mẽ của công nghệ thông tin và khả năng thu thập dữ liệu sức khỏe điện tử (EHR - Electronic Health Records), chúng ta có cơ hội tiếp cận các tập dữ liệu lớn về lịch sử bệnh án, các chỉ số sinh học và yếu tố nguy cơ của bệnh nhân. Việc khai thác hiệu quả nguồn dữ liệu này thông qua các thuật toán ML có thể giúp xác định sớm những cá nhân có nguy cơ cao, từ đó hỗ trợ bác sĩ đưa ra quyết định can thiệp kịp thời.

1.2 Mục tiêu đề tài

Mục tiêu tổng quát của đề tài là xây dựng một hệ thống dự đoán nguy cơ đột quỵ chính xác và đáng tin cậy dựa trên các thuật toán Machine Learning, từ đó hỗ trợ công tác sàng lọc và phòng ngừa bệnh trong thực tiễn y tế.

Động lực chính của đề tài xuất phát từ:

- **Nhu cầu y tế cấp thiết**: Giảm gánh nặng bệnh tật và tử vong do đột quỵ thông qua phát hiện sớm
- **Cơ hội công nghệ**: Tận dụng sức mạnh của ML trong phân tích dữ liệu y tế quy mô lớn
- **Khả năng ứng dụng thực tế**: Xây dựng công cụ hỗ trợ quyết định lâm sàng có độ chính xác cao
- **Giá trị học thuật**: Khám phá các yếu tố nguy cơ quan trọng và mối quan hệ phi tuyến giữa chúng

1.2.1 Mục tiêu cụ thể

1. **Phân tích và làm sạch dữ liệu y tế**:
   - Thu thập và khảo sát bộ dữ liệu Healthcare Dataset Stroke Data từ Kaggle (5,110 bệnh nhân, 12 thuộc tính)
   - Xử lý giá trị thiếu (missing values), ngoại lai (outliers) và mất cân bằng dữ liệu (class imbalance)
   - Thực hiện phân tích khám phá dữ liệu (EDA) để hiểu rõ đặc tính và phân phối của các biến
2. **Xây dựng quy trình tiền xử lý tự động**:
   - Thiết kế pipeline preprocessing chuẩn hóa sử dụng sklearn
   - Áp dụng kỹ thuật SMOTE (Synthetic Minority Oversampling Technique) để cân bằng dữ liệu huấn luyện
3. **Lựa chọn đặc trưng quan trọng**:
   - Sử dụng 4 phương pháp độc lập: Correlation Analysis, Mutual Information, Random Forest Importance, Statistical Tests
   - Xác định top 8 đặc trưng có tác động mạnh nhất đến nguy cơ đột quỵ
   - So sánh và xác thực kết quả với kiến thức y học hiện đại
4. **Huấn luyện và so sánh các mô hình ML**:
   - Triển khai 4 thuật toán: Logistic Regression, Random Forest, SVM (RBF kernel), K-Nearest Neighbors
   - Đánh giá hiệu năng dựa trên F1-Score, Recall, ROC-AUC (thay vì Accuracy do dữ liệu mất cân bằng)
   - Lựa chọn mô hình tối ưu ưu tiên Recall cao (giảm thiểu False Negatives - bỏ sót ca bệnh)

5. **Đánh giá và giải thích kết quả**:
    - Phân tích confusion matrix, ROC curves và các metrics chi tiết
    - Giải thích ý nghĩa y học của các kết quả dự đoán (false positives vs false negatives)
    - Đề xuất chiến lược triển khai mô hình trong môi trường lâm sàng

1.2.2 Chỉ số thành công

Đề tài được coi là thành công khi đạt được:

- **Recall $\geq$ 0.75**: Phát hiện ít nhất 75% các ca đột quỵ (minimize False Negatives)
- **F1-Score $\geq$ 0.20**: Cân bằng hợp lý giữa Precision và Recall
- **ROC-AUC $\geq$ 0.80**: Khả năng phân biệt tốt giữa hai lớp (stroke vs no stroke)
- **Reproducibility**: Kết quả ổn định và có thể tái tạo với random_state cố định

1.3 Đối tượng áp dụng

Hệ thống dự đoán nguy cơ đột quỵ được thiết kế nhằm phục vụ các đối tượng sau:

1.3.1 Đối tượng trực tiếp

1. **Bác sĩ lâm sàng**:
    - Bác sĩ đa khoa tại phòng khám, trung tâm y tế
    - Bác sĩ chuyên khoa tim mạch, thần kinh
    - Y sĩ và điều dưỡng thực hiện khám sàng lọc ban đầu
2. **Cơ sở y tế**:
    - Bệnh viện, phòng khám đa khoa
    - Trung tâm y tế dự phòng
    - Trạm y tế xã/phường (chăm sóc sức khỏe ban đầu)
3. **Người dân**:
    - Người lớn tuổi (từ 40 tuổi trở lên) cần kiểm tra sức khỏe định kỳ
    - Người có yếu tố nguy cơ cao: tăng huyết áp, bệnh tim, tiểu đường
    - Người có tiền sử gia đình mắc đột quỵ
    - Cộng đồng quan tâm đến phòng ngừa bệnh tật

1.3.2 Đối tượng gián tiếp

- **Nhà nghiên cứu y sinh**: Sử dụng mô hình và phương pháp làm tài liệu tham khảo
- **Sinh viên y khoa/công nghệ thông tin**: Học tập về ứng dụng ML trong y tế
- **Nhà quản lý y tế**: Đưa ra chính sách phòng ngừa dựa trên phân tích dữ liệu
- **Nhà phát triển phần mềm y tế**: Tích hợp mô hình vào hệ thống EMR/HIS

1.4 Phạm vi và ứng dụng

1.4.1 Phạm vi nghiên cứu

Phạm vi dữ liệu: Bộ dữ liệu được sử dụng trong đề tài lấy từ **Stroke Prediction Dataset** trên nền tảng **Kaggle**[4] . Tập dữ liệu có các đặc điểm chính như sau:

- **Kích thước mẫu:** 5.110 bản ghi, mỗi bản ghi tương ứng với một bệnh nhân.
- **Các biến đầu vào:** Gồm 11 biến độc lập (loại bỏ 1 biến ID), được chia thành ba nhóm chính:
  - *Nhân khẩu học:* Tuổi, giới tính, tình trạng hôn nhân, loại công việc, nơi cư trú.
  - *Y tế:* Tăng huyết áp, bệnh tim, mức glucose trung bình, chỉ số khối cơ thể (BMI).
  - *Lối sống:* Tình trạng hút thuốc.
- **Biến mục tiêu:** `stroke` (0 = Không đột quỵ, 1 = Đột quỵ).

Phạm vi phương pháp: Trong phạm vi đề tài, nhóm nghiên cứu triển khai các bước xử lý và mô hình hóa dữ liệu như sau:

- *Tiền xử lý dữ liệu (Preprocessing):* Bổ sung giá trị khuyết (missing value imputation), xử lý ngoại lệ bằng IQR, chuẩn hóa (scaling) và mã hóa biến (encoding).
- *Cân bằng dữ liệu (Class balancing):* Áp dụng kỹ thuật **SMOTE** để tăng cường mẫu thuộc lớp thiểu số (chỉ áp dụng cho tập huấn luyện).
- *Lựa chọn đặc trưng (Feature selection):* Sử dụng bốn phương pháp: hệ số tương quan, Mutual Information, Random Forest Importance và các kiểm định thống kê.
- *Huấn luyện mô hình (Modeling):* Thực nghiệm với bốn thuật toán học có giám sát gồm: Logistic Regression, Random Forest, Support Vector Machine (SVM) và K-Nearest Neighbors (KNN).
- *Đánh giá mô hình (Evaluation):* Sử dụng các thước đo F1-Score, Recall, Precision, ROC-AUC và ma trận nhầm lẫn (Confusion Matrix).

Giới hạn nghiên cứu:

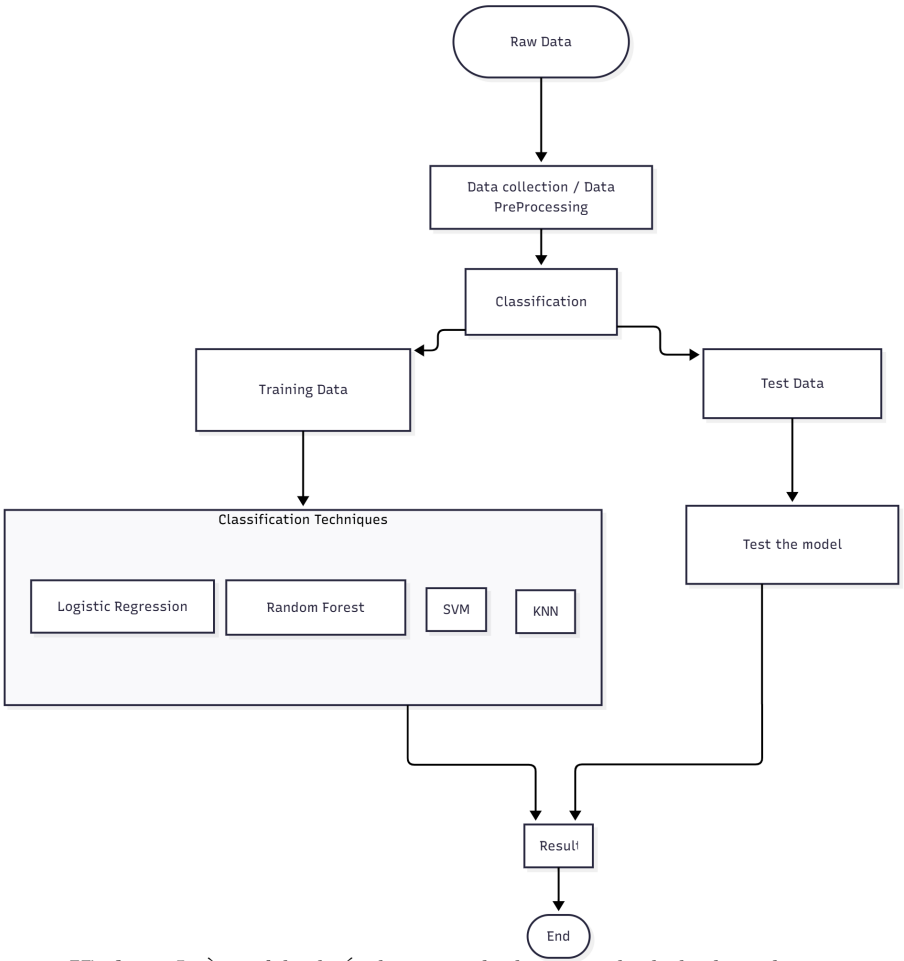
- Dữ liệu được thu thập từ một nguồn duy nhất (Kaggle), chưa phản ánh đa dạng dân số.
- Bộ dữ liệu thiếu yếu tố thời gian, do đó không thể theo dõi diễn tiến bệnh.
- Thiếu một số biến y học quan trọng như: nồng độ lipid máu, mức độ vận động, chế độ dinh dưỡng.
- Chưa tích hợp các dữ liệu y học hình ảnh (CT Scan, MRI), hạn chế khả năng phân tích chuyên sâu.

1.4.2 Ứng dụng thực tiễn

Đề tài hướng đến ba hướng ứng dụng chính trong thực tiễn y tế:

1. **Hệ thống sàng lọc tự động:**
   - Tích hợp vào các phần mềm quản lý bệnh viện (HIS – Hospital Information System).
   - Cảnh báo sớm khi bệnh nhân có nguy cơ đột quỵ cao (xác suất > 50

[4]Nguồn: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset



Hình 2: *Luồng xử lý, huấn luyện và đánh giá mô hình dự đoán đột quỵ*

- Hỗ trợ ưu tiên khám chuyên sâu cho nhóm bệnh nhân nguy cơ cao.

2. **Công cụ hỗ trợ quyết định lâm sàng (CDSS – Clinical Decision Support System):**
   - Cung cấp điểm số nguy cơ (risk score) và diễn giải các yếu tố đóng góp vào quyết định.
   - Gợi ý các xét nghiệm, kiểm tra bổ sung dựa trên hồ sơ bệnh nhân.
   - Đề xuất các biện pháp phòng ngừa phù hợp như điều chỉnh thuốc hoặc thay đổi lối sống.
3. **Nghiên cứu dịch tễ học:**
   - Phân tích các yếu tố nguy cơ trên quy mô lớn để nhận diện xu hướng bệnh lý.
   - Xác định nhóm dân số dễ tổn thương nhằm định hướng chính sách y tế công cộng.
   - Đánh giá hiệu quả của các chương trình phòng chống và can thiệp cộng đồng.

2 Cơ sở lý thuyết

2.1 Tổng quan về bài toán phân loại nhị phân

2.1.1 Định nghĩa bài toán

Bài toán phân loại nhị phân (Binary Classification) là một trong những bài toán cơ bản nhất trong học máy có giám sát, trong đó mục tiêu là phân loại các mẫu dữ liệu vào một trong hai lớp rời rạc. Trong dự án này, hai lớp được định nghĩa là:

- **Lớp 0 (Negative)**: Bệnh nhân không bị đột quỵ (No Stroke)
- **Lớp 1 (Positive)**: Bệnh nhân bị đột quỵ (Stroke)

Với dataset **Healthcare Stroke Data** gồm 5,110 bệnh nhân, bài toán có đặc điểm:

$$\text{Tỷ lệ lớp} = \frac{N_{\text{negative}}}{N_{\text{positive}}} = \frac{4,861}{249} \approx 19.5 : 1 \tag{1}$$

Đây là một bài toán **class imbalance nghiêm trọng**, với 95.13% mẫu thuộc lớp âm và chỉ 4.87% mẫu thuộc lớp dương.

2.1.2 Thách thức của class imbalance

Sự mất cân bằng lớp dữ liệu tạo ra các thách thức đặc biệt:

1. **Bias towards majority class**: Mô hình có xu hướng dự đoán tất cả mẫu thuộc lớp đa số để tối đa hóa accuracy.
2. **Poor minority class detection**: Khả năng phát hiện lớp thiểu số (stroke cases) bị hạn chế nghiêm trọng.
3. **Misleading accuracy**: Accuracy cao không đảm bảo hiệu suất tốt. Ví dụ: mô hình dự đoán tất cả là "No Stroke" vẫn đạt 95% accuracy nhưng hoàn toàn vô dụng trong thực tế y tế.

2.1.3 Metrics đánh giá phù hợp

Với bài toán imbalanced, các metrics quan trọng là:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall (Sensitivity)} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

$$\text{ROC-AUC} = \int_0^1 \text{TPR}(FPR^{-1}(x))\, dx \tag{5}$$

Trong đó:

- **TP** (True Positive): Số ca stroke được phát hiện đúng

- **FP** (False Positive): Số ca không stroke bị dự đoán nhầm là stroke
- **FN** (False Negative): Số ca stroke bị bỏ sót (nguy hiểm!)
- **TN** (True Negative): Số ca không stroke được phát hiện đúng

Trong ngữ cảnh y tế, **False Negative** (bỏ sót ca stroke) nguy hiểm hơn **False Positive** (cảnh báo nhầm), do đó **Recall** được ưu tiên cao hơn **Precision**.

2.2 Các kỹ thuật tiền xử lý dữ liệu

2.2.1 Xử lý giá trị thiếu (Missing Values)

Phương pháp Imputation

Trong dataset, cột `bmi` có 201 giá trị thiếu (3.93%). Các phương pháp imputation được sử dụng:

1. **Median Imputation** (cho biến số):

$$x_{\text{missing}} = \text{median}(\{x_i | x_i \text{ không thiếu}\}) \tag{6}$$

   Ưu điểm: Robust với outliers, không bị ảnh hưởng bởi giá trị cực trị.

2. **Mode Imputation** (cho biến phân loại):

$$x_{\text{missing}} = \text{mode}(\{x_i | x_i \text{ không thiếu}\}) \tag{7}$$

   Ưu điểm: Bảo toàn phân phối của biến categorical.

2.2.2 Xử lý outliers

Phương pháp IQR (Interquartile Range)

Outliers được xử lý bằng IQR-based capping:

$$Q_1 = 25\text{th percentile}, \quad Q_3 = 75\text{th percentile} \tag{8}$$

$$IQR = Q_3 - Q_1 \tag{9}$$

$$\text{Lower bound} = Q_1 - 1.5 \times IQR \tag{10}$$

$$\text{Upper bound} = Q_3 + 1.5 \times IQR \tag{11}$$

Giá trị ngoài khoảng [Lower, Upper] được cắt (clipping):

$$x_{\text{capped}} = \begin{cases} \text{Lower} & \text{if } x < \text{Lower} \\ \text{Upper} & \text{if } x > \text{Upper} \\ x & \text{otherwise} \end{cases} \tag{12}$$

Áp dụng cho: `bmi` (max 97.6 → outlier) và `avg_glucose_level`.

2.2.3 Feature Scaling

StandardScaler (Z-score Normalization)

Chuyển đổi features về phân phối chuẩn với mean = 0 và std = 1:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma} \tag{13}$$

trong đó:

- $\mu = \text{mean}(x)$
- $\sigma = \text{std}(x)$

Ưu điểm:

- Phù hợp với Logistic Regression, SVM
- Bảo toàn thông tin về outliers (sau khi capping)
- Robust với các thuật toán gradient-based

MinMaxScaler

Chuyển đổi features về khoảng [0, 1]:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{14}$$

Ưu điểm:

- Bảo toàn zero entries trong sparse data
- Phù hợp với neural networks

2.2.4 Feature Encoding

One-Hot Encoding

Chuyển đổi biến phân loại thành binary vectors:

$$x \in \{\text{cat}_1, \text{cat}_2, \ldots, \text{cat}_n\} \rightarrow \mathbf{v} \in \{0,1\}^n \tag{15}$$

Ví dụ: `gender` $\in$ {Male, Female, Other} $\rightarrow$ 3 binary columns:

Original	gender_Male	gender_Female	gender_Other
Male	1	0	0
Female	0	1	0
Other	0	0	1

Bảng 2: *One-Hot Encoding cho biến gender*

2.3 Kỹ thuật cân bằng dữ liệu

2.3.1 SMOTE (Synthetic Minority Oversampling Technique)

Nguyên lý hoạt động

Thuật toán **SMOTE (Synthetic Minority Over-sampling Technique)** tạo ra các mẫu tổng hợp (*synthetic samples*) cho lớp thiểu số bằng cách nội suy giữa các mẫu hiện có. Ý tưởng chính là mở rộng không gian đặc trưng của lớp thiểu số thay vì chỉ sao chép lại các mẫu cũ.

Algorithm 1 Thuật toán SMOTE

1: **for** mỗi mẫu thiểu số x_i **do**
2: Tìm k láng giềng gần nhất (nearest neighbors) trong lớp thiểu số
3: Chọn ngẫu nhiên một láng giềng x_{nn}
4: Sinh mẫu mới:

$$x_{\text{new}} = x_i + \lambda \cdot (x_{\text{nn}} - x_i) \tag{16}$$

5: với $\lambda \sim \text{Uniform}(0, 1)$
6: **end for**

Lưu ý quan trọng:

- SMOTE chỉ áp dụng trên **training set**
- Test set giữ nguyên phân phối gốc (realistic evaluation)
- Fit SMOTE sau khi split train/test để tránh data leakage

2.3.2 Stratified Sampling

Đảm bảo tỷ lệ lớp được bảo toàn khi split train/test:

$$\frac{N_{\text{positive}}^{\text{train}}}{N_{\text{total}}^{\text{train}}} = \frac{N_{\text{positive}}^{\text{test}}}{N_{\text{total}}^{\text{test}}} = \frac{N_{\text{positive}}}{N_{\text{total}}} \tag{17}$$

2.4 Các thuật toán Áp dụng

2.4.1 Logistic Regression

Khái niệm:
Logistic Regression (hồi quy Logistic) là một phương pháp thống kê và học máy phổ biến, được sử dụng chủ yếu trong các bài toán *phân loại nhị phân* (binary classification). Mục tiêu của mô hình là ước lượng xác suất một quan sát thuộc về một trong hai lớp dựa trên các biến độc lập.
Khác với hồi quy tuyến tính (Linear Regression), Logistic Regression không dự đoán giá trị thực mà ánh xạ đầu ra thông qua **hàm sigmoid (logistic function)** để đảm bảo giá trị đầu ra nằm trong khoảng $(0, 1)$, tương ứng với xác suất.

Nguyên lý toán học:
Hàm sigmoid là một hàm phi tuyến, có dạng:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{18}$$

trong đó:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p = \boldsymbol{\beta}^T \boldsymbol{x} \tag{19}$$

là tổ hợp tuyến tính của các biến độc lập.

Đặc điểm của hàm sigmoid:

- Đầu vào $z \in (-\infty, +\infty)$.
- Đầu ra $\sigma(z) \in (0, 1)$.
- Có thể diễn giải trực tiếp như một xác suất — giá trị $\sigma(z)$ càng gần 1 thì khả năng mẫu thuộc lớp dương càng cao.

Mô hình xác suất

Xác suất để biến phụ thuộc $Y \in \{0, 1\}$ nhận giá trị 1 được xác định bởi:

$$P(Y = 1|\boldsymbol{x}) = \pi(\boldsymbol{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}} \tag{20}$$

và tương ứng:

$$P(Y = 0|\boldsymbol{x}) = 1 - \pi(\boldsymbol{x}) \tag{21}$$

Để thuận tiện phân tích, Logistic Regression sử dụng phép biến đổi **logit** (log-odds):

$$\text{logit}(\pi(\boldsymbol{x})) = \ln\left(\frac{\pi(\boldsymbol{x})}{1 - \pi(\boldsymbol{x})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{22}$$

Nhờ phép biến đổi này, mối quan hệ giữa các biến độc lập và log-odds trở nên tuyến tính, giúp việc ước lượng tham số $\boldsymbol{\beta}$ có thể thực hiện bằng các phương pháp tối ưu như *Maximum Likelihood Estimation (MLE)*.

Quy trình huấn luyện và dự đoán:

Với một mẫu đầu vào $\boldsymbol{x}$, xác suất dự đoán thuộc lớp dương (ví dụ: *bị đột quỵ*) là:

$$P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{\beta}^T \boldsymbol{x}) \tag{23}$$

Dựa vào ngưỡng mặc định 0.5, mô hình đưa ra quyết định phân loại:

$$\hat{y} = \begin{cases} 1, & \text{nếu } P(y = 1|\boldsymbol{x}) \geq 0.5 \\ 0, & \text{nếu } P(y = 1|\boldsymbol{x}) < 0.5 \end{cases} \tag{24}$$

Hàm mất mát: Binary Cross-Entropy

Mô hình Logistic Regression được huấn luyện bằng cách tối thiểu hóa hàm mất mát dạng *Binary Cross-Entropy (Log Loss)*:

$$L(\boldsymbol{\beta}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{25}$$

trong đó:

- N: số mẫu huấn luyện
- y_i: nhãn thực tế của mẫu thứ i

- $\hat{y}_i = P(y_i = 1|\boldsymbol{x}_i)$: xác suất dự đoán

Tối ưu hóa: Gradient Descent
Các tham số $\boldsymbol{\beta}$ được cập nhật thông qua quy tắc Gradient Descent:

$$\beta_j := \beta_j - \alpha \frac{\partial L}{\partial \beta_j} \tag{26}$$

với hệ số học α (learning rate) và gradient:

$$\frac{\partial L}{\partial \beta_j} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_{ij} \tag{27}$$

Regularization:
Để tránh overfitting, Logistic Regression thường bổ sung thành phần *Regularization* vào hàm mất mát:

- **L2 Regularization (Ridge):**

$$L_{\text{ridge}}(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{28}$$

- **L1 Regularization (Lasso):**

$$L_{\text{lasso}}(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j| \tag{29}$$

trong đó λ là hệ số điều chỉnh mức phạt (với $C = 1/\lambda$ trong `scikit-learn`).

Ưu điểm:

- Dễ hiểu, dễ triển khai, và kết quả có thể diễn giải rõ ràng
- Không yêu cầu giả định phân phối chuẩn của biến độc lập
- Tính toán nhanh, phù hợp với dữ liệu vừa và lớn
- Có thể diễn giải trực tiếp trọng số β_j dưới dạng tầm quan trọng của đặc trưng
- Cho phép mở rộng sang bài toán đa lớp (One-vs-Rest, Multinomial)

Hạn chế:

- Giả định mối quan hệ tuyến tính giữa log-odds và các biến độc lập
- Nhạy cảm với dữ liệu mất cân bằng và hiện tượng đa cộng tuyến
- Hiệu suất kém với dữ liệu phi tuyến hoặc phức tạp
- Yêu cầu chuẩn hóa đặc trưng để mô hình hội tụ tốt

Ứng dụng trong khai phá dữ liệu:

- Phân loại nhị phân: dự đoán nhãn cho các đối tượng dựa trên đặc trưng.
- Phân tích rủi ro: ước lượng xác suất xảy ra sự kiện (ví dụ khách hàng vỡ nợ).
- Là mô hình *baseline* trong nhiều nghiên cứu, dùng để so sánh hiệu quả với các mô hình phi tuyến (Random Forest, SVM, Neural Network)

2.4.2 Random Forest

Khái niệm:

Random Forest là một thuật toán học máy thuộc nhóm *ensemble learning*, trong đó nhiều mô hình yếu (*weak learners*) — cụ thể là các cây quyết định (*Decision Trees*) — được kết hợp lại để tạo thành một mô hình mạnh (*strong learner*). Ý tưởng chính là việc tổng hợp nhiều cây quyết định huấn luyện trên các tập dữ liệu ngẫu nhiên và tập con đặc trưng khác nhau sẽ giúp giảm phương sai (variance) của mô hình, hạn chế hiện tượng overfitting, đồng thời cải thiện hiệu năng dự đoán.

Random Forest được sử dụng rộng rãi trong nhiều lĩnh vực như:

- **Y tế:** dự đoán bệnh lý, phân loại hình ảnh y khoa.
- **Tài chính:** đánh giá rủi ro tín dụng, phát hiện gian lận.
- **Marketing:** phân khúc khách hàng, dự đoán hành vi mua hàng.
- **Dữ liệu lớn:** huấn luyện song song, ổn định cao khi xử lý tập dữ liệu khổng lồ.

Nguyên lý hoạt động:

a. Cây quyết định (Decision Tree) làm nền tảng

- Với bài toán phân loại: cây quyết định dự đoán nhãn bằng cách đi theo các nút phân nhánh dựa trên thuộc tính, kết quả cuối cùng tại lá (leaf node) là một lớp (class)
- Với bài toán hồi quy: kết quả tại lá là một giá trị trung bình của tập dữ liệu thuộc lá đó.

Mỗi node được chia theo tiêu chí chọn feature tối ưu, phổ biến nhất là:

Tiêu chí phân chia node – Gini Impurity:

Gini Impurity đo lường độ "không thuần khiết" (*impurity*) của một node trong cây quyết định, được xác định như sau:

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{30}$$

trong đó p_k là tỷ lệ mẫu thuộc lớp k tại node hiện tại. Giá trị Gini càng nhỏ thì node càng "thuần khiết" — tức là các mẫu trong node chủ yếu thuộc về cùng một lớp.

Mức giảm độ không thuần khiết khi phân chia node (Gini Gain) được tính bằng:

$$\text{Gain}_{\text{Gini}} = \text{Gini}(D_{\text{parent}}) - \left(\frac{|D_{\text{left}}|}{|D|} \text{Gini}(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D|} \text{Gini}(D_{\text{right}}) \right) \tag{31}$$

Trong đó:

- D_{parent}: tập dữ liệu tại node cha.
- $D_{\text{left}}, D_{\text{right}}$: các tập dữ liệu con sau khi phân chia.
- $|D|$: số lượng mẫu trong node hiện tại.

Node được chọn để tách sao cho giá trị $\text{Gain}_{\text{Gini}}$ là lớn nhất, giúp tăng độ thuần khiết của cây và cải thiện khả năng phân loại của mô hình.

b. Ngẫu nhiên hóa: Bootstrap Aggregation (Bagging) và Random Feature Selection

Random Forest được xây dựng dựa trên hai kỹ thuật ngẫu nhiên hóa chính:

- **Bootstrap Aggregation (Bagging):** Từ tập dữ liệu gốc D, tạo ra nhiều tập con $D_1, D_2, \ldots, D_B$ bằng cách lấy mẫu có hoàn lại (*sampling with replacement*). Mỗi tập D_b sẽ được sử dụng để huấn luyện một cây quyết định độc lập. Cách tiếp cận này giúp giảm phương sai và tăng tính ổn định của mô hình tổng thể.

  1. Bootstrap Sampling: Tạo B bootstrap samples từ tập huấn luyện gốc:

$$D_b = \text{sample with replacement}(D_{\text{train}}), \quad b = 1, 2, \ldots, B \tag{32}$$

- **Random Feature Selection:** Tại mỗi nút (*node*) trong quá trình xây dựng cây, chỉ xét một tập con ngẫu nhiên gồm m thuộc tính trong tổng số n_{features} để chọn ra đặc trưng tốt nhất cho việc phân chia. Điều này giúp giảm tương quan giữa các cây, tăng tính đa dạng và cải thiện độ tổng quát của mô hình.

  2. Feature Randomness: Khi xây dựng mỗi node trong cây, chỉ xét một tập con ngẫu nhiên gồm m features:

$$m = \lfloor \sqrt{n_{\text{features}}} \rfloor \quad \text{(cho phân loại)} \tag{33}$$

c. Quy trình huấn luyện và dự đoán:

1. **Bước 1:** Xây dựng B cây quyết định độc lập, mỗi cây được huấn luyện trên một tập bootstrap D_b.
2. **Bước 2:** Mỗi cây được phát triển đầy đủ (không *pruning*).
3. **Bước 3:** Kết hợp kết quả dự đoán từ tất cả các cây.

Phân loại (Classification):

$$\hat{y} = \text{mode}\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_B(\mathbf{x})\} \tag{34}$$

Xác suất dự đoán:

$$P(y = 1|\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \mathbb{1}[h_b(\mathbf{x}) = 1] \tag{35}$$

Hồi quy (Regression):

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} h_b(\mathbf{x}) \tag{36}$$

d. Đánh giá mô hình — Out-of-Bag (OOB) Error:
Do sử dụng bootstrap sampling, khoảng 37% mẫu không được dùng để huấn luyện mỗi cây (*out-of-bag samples*). Các mẫu này có thể dùng để ước lượng lỗi của mô hình mà không cần tập kiểm thử riêng biệt:

$$\text{OOB Error} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left[\hat{y}_i^{\text{OOB}} \neq y_i\right] \tag{37}$$

e. Độ quan trọng của đặc trưng (Feature Importance):
Mức độ quan trọng của từng đặc trưng được đo dựa trên mức giảm trung bình của chỉ số Gini:

$$\text{Importance}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in \text{Tree}_b} \Delta\text{Gini}_t \cdot \mathbb{1}[X_j \text{ được chọn tại node } t] \tag{38}$$

Giá trị càng lớn chứng tỏ đặc trưng X_j có ảnh hưởng lớn đến mô hình.

f. Các siêu tham số (Hyperparameters) quan trọng:

Hiệu năng của mô hình Random Forest phụ thuộc đáng kể vào việc lựa chọn các siêu tham số (hyperparameters). Một số tham số quan trọng bao gồm:

- **n_estimators (B):** Số lượng cây quyết định trong rừng. Giá trị càng lớn, mô hình càng ổn định nhưng chi phí tính toán tăng. Thông thường B nằm trong khoảng 100–500.
- **max_depth:** Độ sâu tối đa của mỗi cây. Nếu để mặc định (None), các cây có thể phát triển đến khi lá hoàn toàn thuần khiết, dễ dẫn đến overfitting.
- **min_samples_split:** Số lượng mẫu tối thiểu cần có để tiếp tục phân chia một node. Giá trị cao giúp giảm overfitting nhưng có thể làm giảm độ chính xác.
- **min_samples_leaf:** Số lượng mẫu tối thiểu tại mỗi lá. Thường được chọn từ 1 đến 5, giúp kiểm soát độ sâu của cây.
- **max_features:** Số lượng đặc trưng được xem xét tại mỗi lần chia node. Với bài toán phân loại thường lấy $m = \sqrt{n_{\text{features}}}$, còn với hồi quy là $m = \frac{n_{\text{features}}}{3}$.
- **bootstrap:** Quy định việc sử dụng lấy mẫu có hoàn lại (bootstrap sampling). Thường đặt `True` để tận dụng lợi thế của phương pháp bagging.
- **class_weight:** Cân bằng trọng số giữa các lớp khi dữ liệu mất cân bằng (*imbalanced dataset*). Có thể đặt là `balanced` để tự động điều chỉnh theo tần suất lớp.

2.3. Ưu điểm và hạn chế

Ưu điểm:

- Hiệu năng cao trong nhiều bài toán thực tế, đặc biệt khi dữ liệu có nhiều đặc trưng và phi tuyến.
- Giảm hiện tượng overfitting so với Decision Tree đơn lẻ nhờ cơ chế bagging và random feature selection.
- Có khả năng đánh giá tầm quan trọng của đặc trưng (feature importance).
- Hoạt động tốt trên dữ liệu có nhiễu hoặc thiếu giá trị.
- Có thể song song hóa quá trình huấn luyện, phù hợp với dữ liệu lớn.

Hạn chế:

- Kích thước mô hình lớn, tiêu tốn nhiều bộ nhớ và tài nguyên tính toán.
- Khó diễn giải trực quan hơn Logistic Regression hoặc một cây quyết định đơn lẻ.
- Thời gian huấn luyện và dự đoán có thể chậm khi số lượng cây lớn.

2.4. Ứng dụng trong Data Mining

Trong lĩnh vực khai phá dữ liệu (*Data Mining*), Random Forest được xem là một trong những thuật toán mạnh mẽ và linh hoạt nhất nhờ khả năng tổng quát hóa cao và hiệu năng ổn định. Một số ứng dụng tiêu biểu bao gồm:

- **Phân loại (Classification):** Dự đoán nhãn đầu ra, ví dụ như dự đoán khách hàng rời bỏ dịch vụ (*customer churn*) hoặc xác định bệnh nhân có nguy cơ đột quỵ.
- **Hồi quy (Regression):** Dự đoán các giá trị liên tục như giá nhà, doanh thu, hoặc tỷ lệ tăng trưởng.
- **Feature Selection:** Xác định và đánh giá độ quan trọng của các đặc trưng, từ đó rút gọn chiều dữ liệu.
- **Phát hiện bất thường (Anomaly Detection):** Phát hiện các điểm dữ liệu hiếm hoặc bất thường trong các hệ thống giám sát hoặc gian lận tài chính.

Nhờ sự cân bằng giữa độ chính xác, khả năng mở rộng và tính ổn định, Random Forest thường được xem là một **mô hình baseline mạnh mẽ** để so sánh hiệu quả với các mô hình tiên tiến hơn như Gradient Boosting hoặc Neural Network.

2.4.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) là một mô hình học máy có giám sát mạnh mẽ, chủ yếu được sử dụng cho các bài toán phân loại, nhưng cũng có thể mở rộng cho bài toán hồi quy. Mục tiêu chính của SVM là tìm ra một *ranh giới quyết định* (*decision boundary*) tối ưu để phân tách các lớp dữ liệu khác nhau.

Nguyên lý hoạt động: Tìm siêu phẳng phân tách tối ưu

- **Siêu phẳng (Hyperplane):** Trong không gian n chiều, siêu phẳng là một mặt phẳng có số chiều $n-1$.
  - Trong không gian 2D, siêu phẳng là một đường thẳng (1 chiều).
  - Trong không gian 3D, siêu phẳng là một mặt phẳng (2 chiều).
- **Mục tiêu:** SVM không chỉ tìm được siêu phẳng phân tách các lớp dữ liệu, mà còn *tối đa hóa khoảng cách* (*margin*) giữa siêu phẳng và các điểm dữ liệu gần nhất của từng lớp.
- **Margin và Support Vectors:** *Margin* là khoảng cách từ siêu phẳng đến các điểm dữ liệu gần nhất. Các điểm nằm trên biên của margin được gọi là *Support Vectors* — đây là các điểm dữ liệu "then chốt" xác định vị trí của siêu phẳng tối ưu. Khi các Support Vectors thay đổi, siêu phẳng cũng sẽ thay đổi tương ứng.

Ưu điểm:

- Hiệu quả trong không gian nhiều chiều, đặc biệt khi số lượng đặc trưng lớn hơn số mẫu.
- Khả năng tổng quát hóa cao nhờ nguyên lý "tối đa hóa margin", giúp mô hình hoạt động tốt ngay cả với dữ liệu huấn luyện nhỏ.
- Ổn định (Robust) với nhiễu do chỉ phụ thuộc vào một tập nhỏ các Support Vectors.

Hạn chế:

- Khó lựa chọn hàm kernel và tinh chỉnh các tham số (C, γ), đòi hỏi kiến thức chuyên sâu và thử nghiệm nhiều lần.
- Chi phí tính toán cao đối với bộ dữ liệu lớn, làm giảm tính khả thi khi triển khai thực tế.
- Khó diễn giải, đặc biệt với các kernel phi tuyến — mô hình hoạt động như một "hộp đen" khó hiểu.
- Hiệu suất giảm khi dữ liệu nhiễu hoặc mất cân bằng giữa các lớp.

2.4.4 K-Nearest Neighbors (KNN)

Cơ sở lý thuyết: K-Nearest Neighbors (KNN) là một thuật toán học máy giám sát (*supervised learning*) đơn giản nhưng hiệu quả, có thể được sử dụng cho cả bài toán phân loại (*classification*) và hồi quy (*regression*). Nguyên lý của KNN dựa trên giả định rằng các điểm dữ liệu có đặc trưng tương tự nhau sẽ có nhãn hoặc giá trị đầu ra gần nhau.

Nguyên lý hoạt động:

- Tính khoảng cách giữa mẫu cần dự đoán và toàn bộ các điểm dữ liệu trong tập huấn luyện (thường dùng khoảng cách Euclidean, Manhattan hoặc Minkowski).
- Chọn ra K điểm dữ liệu gần nhất — gọi là *neighbors*.
- Dự đoán:
  - **Phân loại:** Mẫu mới được gán vào lớp chiếm đa số trong K láng giềng gần nhất.
  - **Hồi quy:** Giá trị dự đoán là trung bình hoặc trung vị của giá trị đầu ra của K láng giềng gần nhất.

Công thức khoảng cách Euclidean phổ biến được sử dụng:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Trong đó, x và y là hai điểm dữ liệu trong không gian đặc trưng n chiều.

Lựa chọn tham số K:

- K nhỏ (ví dụ $K = 1$ hoặc $K = 3$): Mô hình nhạy cảm với nhiễu, dễ bị *overfitting*.
- K lớn (ví dụ $K = 15$ hoặc $K = 30$): Mô hình mượt hơn, ít nhiễu hơn nhưng có thể bị *underfitting*.
- Giá trị K tối ưu thường được xác định thông qua phương pháp *cross-validation*.

Ưu điểm:

- Đơn giản, dễ hiểu, không cần giai đoạn huấn luyện phức tạp.
- Linh hoạt: Áp dụng được cho cả phân loại và hồi quy.
- Không yêu cầu giả định về phân bố dữ liệu.

Hạn chế:

- Chi phí tính toán cao: Mỗi lần dự đoán phải tính khoảng cách với toàn bộ dữ liệu huấn luyện.
- Nhạy cảm với dữ liệu nhiễu và thang đo — cần chuẩn hóa dữ liệu trước khi áp dụng.
- Hiệu suất giảm khi dữ liệu có nhiều chiều (*curse of dimensionality*).

Ứng dụng:

- Nhận dạng chữ viết tay, nhận dạng khuôn mặt.
- Phân loại văn bản, hệ thống gợi ý sản phẩm.
- Dự đoán giá nhà hoặc nhiệt độ trong bài toán hồi quy.

3 Khảo sát và phân tích dữ liệu - EDA

3.1 Tổng quan về dataset

3.1.1 Đặc điểm cơ bản

Dataset **Healthcare Stroke Data** được sử dụng trong dự án bao gồm thông tin y tế và nhân khẩu học của 5,110 bệnh nhân. Mục tiêu là dự đoán nguy cơ đột quỵ dựa trên 11 thuộc tính đầu vào.

3.1.2 Cấu trúc thuộc tính

Dataset bao gồm các loại thuộc tính sau:
1. Biến số liên tục (Numeric):

- `age`: Tuổi (0.08 - 82)
- `avg_glucose_level`: Mức glucose trung bình (55.12 - 271.74 mg/dL)
- `bmi`: Chỉ số khối cơ thể (10.3 - 97.6)

2. Biến nhị phân (Binary):

- `hypertension`: Tăng huyết áp (0: Không, 1: Có)
- `heart_disease`: Bệnh tim (0: Không, 1: Có)

3. Biến phân loại (Categorical):

- `gender`: Giới tính (Male, Female, Other)
- `ever_married`: Tình trạng hôn nhân (Yes, No)
- `work_type`: Loại công việc (Private, Self-employed, Govt_job, children, Never_worked)
- `Residence_type`: Nơi cư trú (Urban, Rural)
- `smoking_status`: Tình trạng hút thuốc (formerly smoked, never smoked, smokes, Unknown)

4. Biến mục tiêu (Target):

- `stroke`: Đột quỵ (0: Không, 1: Có)

3.2 Phân tích biến mục tiêu (Target Analysis)

3.2.1 Phân phối class

Một trong những vấn đề quan trọng nhất của dataset là **class imbalance nghiêm trọng**. Phân tích cho thấy:

$$P(\text{stroke} = 1) = \frac{249}{5110} = 0.0487 \approx 4.87\% \tag{39}$$

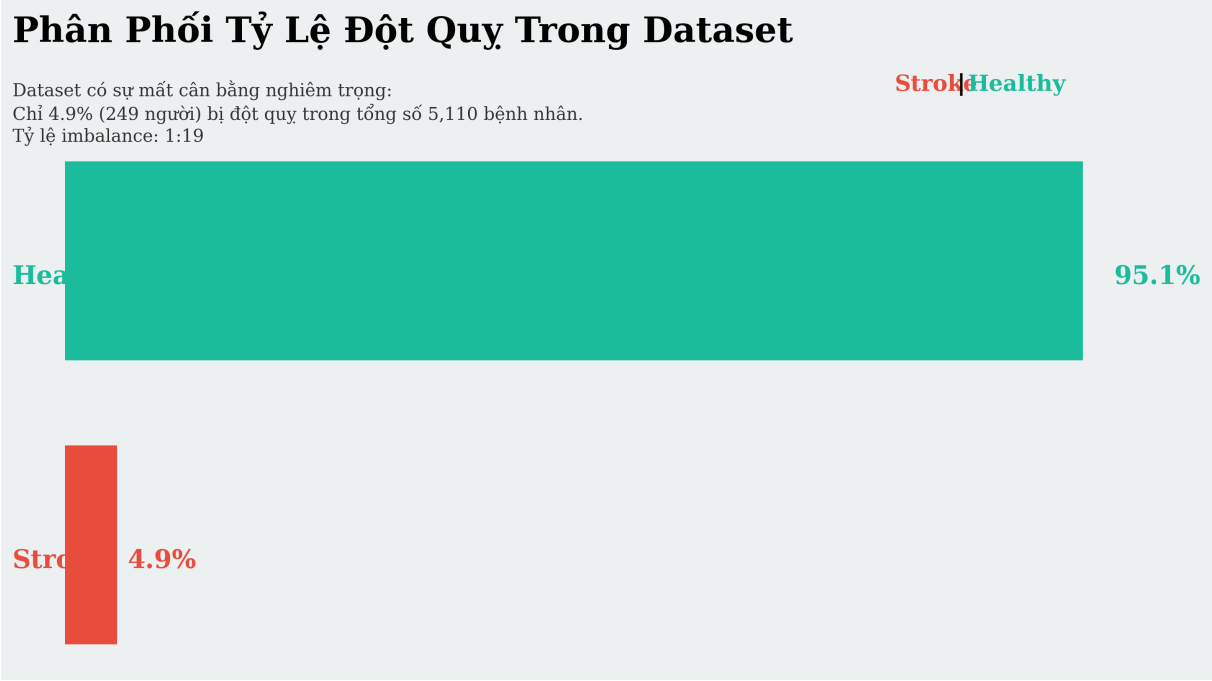
$$P(\text{stroke} = 0) = \frac{4861}{5110} = 0.9513 \approx 95.13\% \tag{40}$$

$$\text{Imbalance Ratio} = \frac{N_{\text{majority}}}{N_{\text{minority}}} = \frac{4861}{249} \approx 19.5 \tag{41}$$

Sự mất cân bằng này có ý nghĩa quan trọng:

- Models có xu hướng bias về lớp đa số (No Stroke)
- Accuracy không phải là metric đánh giá phù hợp
- Cần áp dụng techniques như SMOTE để cân bằng dữ liệu training
- Metrics như F1-Score, Recall, ROC-AUC quan trọng hơn

Visualization: Hình 3 minh họa phân phối không cân bằng của biến target.



Hình 3: *Phân phối biến target - Class Imbalance nghiêm trọng (95.13% vs 4.87%)*

3.3 Phân tích biến số (Numeric Analysis)

3.3.1 Thống kê mô tả

Ba biến số chính trong dataset được phân tích với các thống kê mô tả:

Bảng 3: *Thống kê mô tả các biến số*

Metric	age	avg_glucose_level	bmi
Count	5,110	5,110	4,909
Mean	43.23	106.15	28.89
Std Dev	22.61	45.28	7.85
Min	0.08	55.12	10.30
25%	25.00	77.24	23.50
Median	45.00	91.88	28.10
75%	61.00	114.09	33.10
Max	82.00	271.74	97.60

3.3.2 Phát hiện outliers

Phân tích box plots và histograms cho thấy:
1. Age (Tuổi):

- Phân phối tương đối đều, có một số trường hợp trẻ em (< 1 tuổi)
- Không có outliers nghiêm trọng
- Xu hướng: Nguy cơ đột quỵ tăng theo tuổi

2. Average Glucose Level:

- Phân phối lệch phải (right-skewed)
- Có outliers với giá trị cao (> 200 mg/dL)
- Cần áp dụng IQR capping trong preprocessing

3. BMI:

- Phân phối gần chuẩn (near-normal)
- Có outliers ở cả hai đầu (< 15 hoặc > 50)
- Missing values: 201 mẫu (3.93%) - cần imputation

Hình 4: *Phân tích biến số: Histograms và Box plots theo stroke status*

3.3.3 Correlation với target

Phân tích correlation Pearson với biến target cho thấy:

Bảng 4: *Correlation của biến số với stroke*

Feature	Pearson Correlation (absolute)
age	0.2453
avg_glucose_level	0.1319
bmi	0.0361

Nhận xét:

- **age** có correlation mạnh nhất với stroke (0.245)
- **avg_glucose_level** có correlation trung bình (0.132)
- **bmi** có correlation yếu (0.036)

3.4 Phân tích biến phân loại (Categorical Analysis)

3.4.1 Phân phối và stroke rate theo từng biến

1. Gender (Giới tính):

- Female: 2,994 mẫu (58.6%)
- Male: 2,115 mẫu (41.4%)

- Other: 1 mẫu (0.02%)
- Stroke rate: Female (4.71%), Male (5.11%), Other (0%)

2. Ever Married (Tình trạng hôn nhân):

- Yes: 3,353 mẫu (65.6%)
- No: 1,757 mẫu (34.4%)
- **Quan sát quan trọng**: Người đã kết hôn có stroke rate cao hơn (6.5% vs 1.3%)
- Có thể do correlation với tuổi (người đã kết hôn thường lớn tuổi hơn)

3. Work Type (Loại công việc):

- Private: 2,925 mẫu (57.2%)
- Self-employed: 819 mẫu (16.0%)
- children: 687 mẫu (13.4%)
- Govt_job: 657 mẫu (12.9%)
- Never_worked: 22 mẫu (0.4%)
- Stroke rate cao nhất: Self-employed (7.94%), thấp nhất: children (0.29%)

4. Residence Type (Nơi cư trú):

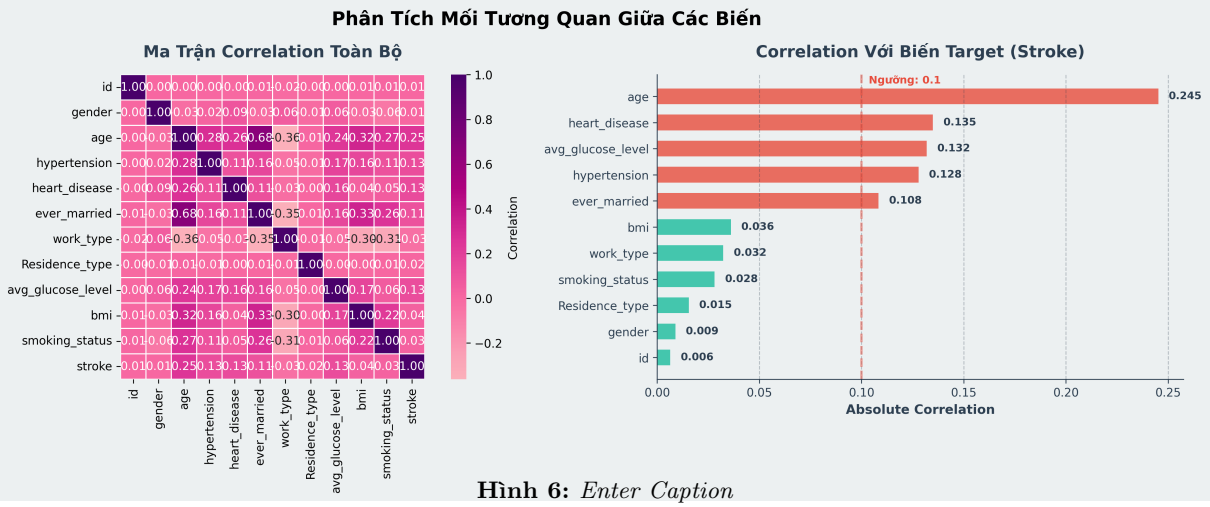
- Urban: 2,596 mẫu (50.8%)
- Rural: 2,514 mẫu (49.2%)
- Stroke rate: Urban (5.20%) vs Rural (4.53%) - Sự khác biệt không lớn

5. Smoking Status (Tình trạng hút thuốc):

- never smoked: 1,892 mẫu (37.0%)
- Unknown: 1,544 mẫu (30.2%)
- formerly smoked: 885 mẫu (17.3%)
- smokes: 789 mẫu (15.4%)
- Stroke rate cao nhất: formerly smoked (7.91%), Unknown có rate thấp nhất (3.04%)

Hình 5: *Phân tích biến phân loại: Count plots phân biệt theo stroke status*

3.5 Phân tích tương quan (Correlation Analysis)



Hình 6: *Enter Caption*

3.5.1 Ma trận correlation

Để phân tích correlation giữa các biến, các biến categorical được encode thành numeric codes. Ma trận correlation hoàn chỉnh cho thấy:

Bảng 5: *Top correlations với biến target stroke*

Feature	Correlation (absolute)
age	0.2453
heart_disease	0.1349
avg_glucose_level	0.1319
hypertension	0.1279
ever_married	0.1083
bmi	0.0361
work_type	0.0323
smoking_status	0.0281
Residence_type	0.0155
gender	0.0089

Insights quan trọng:

- **Age** là predictor mạnh nhất (r = 0.245)
- **Heart disease** và **Hypertension** có correlation trung bình (0.13-0.14)
- **Residence_type** và **Gender** có correlation rất yếu (< 0.02)
- Không có multicollinearity nghiêm trọng giữa các features

Hình 7: *Ma trận correlation heatmap - Phân tích mối quan hệ giữa các biến*

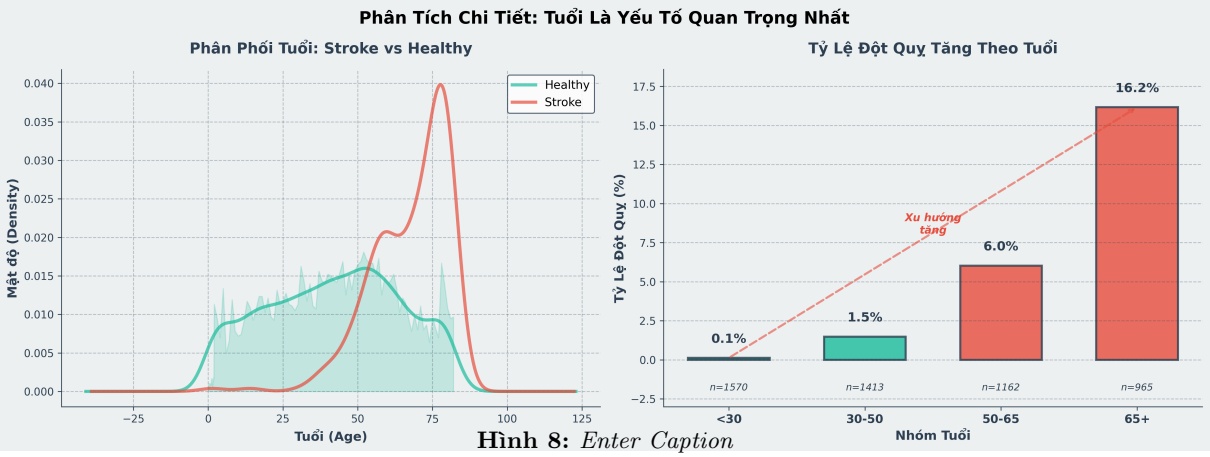
3.6 Phân tích chi tiết biến Age

Do **age** là biến có correlation cao nhất với stroke, một phân tích chuyên sâu được thực hiện:

3.6.1 Phân chia nhóm tuổi

Dữ liệu được chia thành 4 nhóm tuổi:

- $<$ **30**: Trẻ và người trẻ tuổi
- **30-50**: Trung niên
- **50-65**: Người lớn tuổi
- **65+**: Người cao tuổi



Hình 8: *Enter Caption*

3.6.2 Phát hiện chính

1. **Xu hướng tăng mạnh**: Tỷ lệ stroke tăng gấp 124 lần từ nhóm $<$ 30 (0.13%) đến nhóm 65+ (16.17%)
2. **Ngưỡng quan trọng**: Nguy cơ tăng đáng kể sau 50 tuổi

$$\text{Risk Ratio}_{65+/<30} = \frac{16.17\%}{0.13\%} \approx 124 \tag{42}$$

3. **Phân phối không đồng đều**: Nhóm $<$ 30 chiếm 30.7% mẫu nhưng chỉ có 0.8% ca stroke
4. **Nhóm cao rủi ro**: Người cao tuổi (65+) chiếm 18.9% mẫu nhưng có 62.7% tổng số ca stroke

Hình 9: *Phân tích chi tiết tuổi: Phân phối và tỷ lệ stroke theo nhóm tuổi*

3.7 Phát hiện và kết luận từ EDA

3.7.1 Phát hiện chính

1. **Class Imbalance nghiêm trọng**:
   - Tỷ lệ 19.5:1 (4,861 vs 249)
   - Cần SMOTE hoặc class_weight trong training

- Metrics: Ưu tiên F1-Score, Recall, ROC-AUC thay vì Accuracy

2. **Biến quan trọng nhất - Age**:
   - Correlation cao nhất (0.245)
   - Tỷ lệ stroke tăng 124 lần từ < 30 đến 65+
   - Có thể áp dụng feature engineering (age groups, polynomial features)
3. **Missing Values**:
   - Chỉ BMI có 201 missing values (3.93%)
   - Cần imputation strategy: median/mean/KNN imputation
4. **Outliers**:
   - avg_glucose_level: Nhiều giá trị cao > 200
   - BMI: Một số giá trị cực đoan (< 15 hoặc > 50)
   - Áp dụng IQR capping trong preprocessing
5. **Categorical Variables**:
   - ever_married có correlation với age và stroke (Yes: 6.56% vs No: 1.65%)
   - smoking_status: "formerly smoked" có stroke rate cao (7.91%)
   - Residence_type: Urban (5.20%) vs Rural (4.53%) - khác biệt nhỏ
   - Gender: Female (4.71%) vs Male (5.11%) - tương đương nhau
6. **Feature Engineering Opportunities**:
   - Age groups (binning)
   - Interaction features: age $\times$ hypertension, age $\times$ heart_disease
   - Polynomial features cho age

3.7.2 Hướng tiếp cận preprocessing

Dựa trên EDA, pipeline preprocessing cần bao gồm:

1. **Missing Value Handling**:
   - BMI: SimpleImputer với strategy='median'
2. **Outlier Treatment**:
   - IQR capping cho avg_glucose_level và BMI
   - Công thức: $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$
3. **Scaling**:
   - StandardScaler cho numeric features (age, avg_glucose_level, BMI)
   - Quan trọng cho SVM, KNN, Logistic Regression
4. **Encoding**:

- OneHotEncoder cho categorical variables
- handle_unknown='ignore' để xử lý unseen categories

5. **Balancing**:
   - SMOTE oversampling cho training set
   - Giữ nguyên test set distribution để đánh giá realistic

3.7.3 Kết luận

EDA cho thấy dataset có những đặc điểm:

- **Strengths**: Dữ liệu sạch với ít missing values, các features có ý nghĩa y tế rõ ràng
- **Challenges**: Class imbalance nghiêm trọng, outliers trong biến số, correlation yếu giữa hầu hết features với target
- **Key Insight**: Age là predictor mạnh nhất, cần đặc biệt chú ý trong modeling
- **Next Steps**: Áp dụng preprocessing pipeline toàn diện với SMOTE, scaling, encoding

Các visualizations từ EDA được sử dụng để hiểu sâu về dữ liệu và định hướng các bước tiền xử lý, feature selection, và model training tiếp theo.

4 Tiền xử lý dữ liệu

4.1 Tổng quan quy trình tiền xử lý

Tiền xử lý dữ liệu là bước quan trọng nhằm chuẩn bị dữ liệu thô thành dạng phù hợp cho các thuật toán Machine Learning. Dựa trên kết quả EDA ở Chương 4, quy trình tiền xử lý được thiết kế để giải quyết các vấn đề:

- **Missing values**: BMI có 201 giá trị thiếu (3.93%)
- **Outliers**: avg_glucose_level và BMI có outliers nghiêm trọng
- **Class imbalance**: Tỷ lệ 19.5:1 giữa lớp âm và dương
- **Mixed data types**: Numeric, categorical, và binary features
- **Feature scaling**: Các biến số có range khác nhau

Pipeline tiền xử lý được implement trong file `prepare-stroke.py` với các bước tuần tự: Data Cleaning → Train-Test Split → Preprocessing Pipeline → SMOTE Balancing.

4.2 Xử lý giá trị thiếu (Missing Values)

4.2.1 Phân tích missing values

Từ EDA, chỉ có cột `bmi` có giá trị thiếu (201 mẫu - 3.93%). Sử dụng **SimpleImputer** từ scikit-learn với hai strategy:

1. Cho biến số (Numeric):

- Strategy: `median` - Robust với outliers
- Áp dụng: `age`, `avg_glucose_level`, `bmi`

$$\text{value}_{\text{imputed}} = \begin{cases} \text{value}_{\text{original}} & \text{if not missing} \\ \text{median(column)} & \text{if missing} \end{cases} \tag{43}$$

2. Cho biến phân loại (Categorical):

- Strategy: `most_frequent`
- Áp dụng: `gender`, `ever_married`, `work_type`, `Residence_type`, `smoking_status`

4.3 Xử lý outliers

Sử dụng phương pháp **IQR (Interquartile Range)** với $k = 1.5$ (Tukey's method):

$$\text{IQR} = Q_3 - Q_1 \tag{44}$$

$$\text{Outlier bounds} = [Q_1 - 1.5 \times \text{IQR}, Q_3 + 1.5 \times \text{IQR}] \tag{45}$$

Thay vì loại bỏ, sử dụng **capping** - giới hạn giá trị vào khoảng IQR:

$$\text{value}_{\text{capped}} = \begin{cases} Q_1 - 1.5 \times \text{IQR} & \text{if value} < Q_1 - 1.5 \times \text{IQR} \\ Q_3 + 1.5 \times \text{IQR} & \text{if value} > Q_3 + 1.5 \times \text{IQR} \\ \text{value} & \text{otherwise} \end{cases} \tag{46}$$

Áp dụng cho `bmi` và `avg_glucose_level`.

4.4 Chia tập dữ liệu (Train-Test Split)

Sử dụng `train_test_split` với stratified sampling:

- **test_size**: 0.2 (20% cho test set)
- **random_state**: 42 (reproducibility)
- **stratify**: `y` (giữ tỷ lệ class)

Bảng 6: *Phân phối stroke sau train-test split*

Dataset	Total	Stroke	Stroke %
Original	5,110	249	4.87%
Train	4,088	199	4.87%
Test	1,022	50	4.89%

4.5 Feature Engineering và Encoding

4.5.1 Chuẩn hóa biến số (StandardScaler)

Đưa các biến số về cùng scale bằng z-score normalization:

$$z = \frac{x - \mu}{\sigma} \tag{47}$$

Áp dụng cho: `age`, `avg_glucose_level`, `bmi`

4.5.2 One-Hot Encoding

Chuyển categorical variables thành binary vectors. Ví dụ với `gender`:

$$\text{OneHot(gender)} = \begin{cases} [1,0,0] & \text{if Female} \\ [0,1,0] & \text{if Male} \\ [0,0,1] & \text{if Other} \end{cases} \tag{48}$$

5 biến categorical tạo ra 16 features sau encoding.

4.5.3 Binary Features

`hypertension` và `heart_disease` giữ nguyên (0/1).

4.6 ColumnTransformer Pipeline

Xử lý đồng thời các loại features khác nhau:

Listing 1: Preprocessing pipeline

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Numeric pipeline
numeric_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# Categorical pipeline
categorical_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

# Combine transformers
preprocessor = ColumnTransformer([
    ("num", numeric_pipeline, numeric_cols),
    ("cat", categorical_pipeline, categorical_cols),
    ("bin", "passthrough", binary_cols)
])

# Fit on train only (avoid data leakage!)
preprocessor.fit(X_train)
X_train_transformed = preprocessor.transform(X_train)
X_test_transformed = preprocessor.transform(X_test)
```

Quan trọng: Pipeline chỉ `fit()` trên training set để tránh data leakage.

4.7 Xử lý Class Imbalance với SMOTE

4.7.1 SMOTE Algorithm

SMOTE (Synthetic Minority Over-sampling Technique) tạo synthetic samples cho lớp minority:

$$x_{\text{synthetic}} = x_i + \lambda \cdot (x_{\text{neighbor}} - x_i), \quad \lambda \sim U(0,1) \tag{49}$$

Quan trọng: SMOTE chỉ áp dụng cho **training set**!

Bảng 7: *Kết quả sau SMOTE*

Dataset	Total	Stroke	Stroke %
Train (Before SMOTE)	4,088	199	4.87%
Train (After SMOTE)	**7,778**	**3,889**	**50.00%**
Test (Unchanged)	1,022	50	4.89%

Lý do không SMOTE test set:

- Test set phải phản ánh real-world distribution

- SMOTE trên test → metrics không đáng tin cậy
- Model phải học detect stroke trong điều kiện imbalanced

4.8 Feature Space sau tiền xử lý

Dataset transform từ 12 cột gốc thành 21 features:

Bảng 8: *Feature composition*

Feature Type	Original	Final
Numeric (scaled)	3	3
Categorical (one-hot)	5	16
Binary (passthrough)	2	2
Total	**10**	**21**

Danh sách 21 features: age, avg_glucose_level, bmi, gender_Female, gender_Male, gender_Other, ever_married_No, ever_married_Yes, work_type_Govt_job, work_type_Never_worked, work_type_Private, work_type_Self-employed, work_type_children, Residence_type_Rural, Residence_type_Urban, smoking_status_Unknown, smoking_status_formerly smoked, smoking_status_never smoked, smoking_status_smokes, hypertension, heart_disease.

4.9 Tổng kết

Quy trình tiền xử lý hoàn chỉnh:

1. **Data Cleaning**: Impute BMI, cap outliers (IQR method)
2. **Train-Test Split**: 80-20, stratified sampling
3. **Preprocessing Pipeline**: Numeric (median + StandardScaler), Categorical (most_frequent + OneHotEncoder), Binary (passthrough)
4. **SMOTE**: Chỉ train set, 4,088 → 7,778 samples (50% balanced)

Bảng 9: *Kết quả preprocessing*

Metric	Train Set	Test Set
Số samples	7,778	1,022
Stroke ratio	50.00%	4.89%
Số features	21	21
Missing values	0	0

5 Xây dựng mô hình

5.1 Quy trình xây dựng mô hình

Các bước chính:

1. Tiền xử lý dữ liệu (chuẩn hóa, chia train/test, cân bằng bằng SMOTE).
2. Lựa chọn mô hình: Logistic Regression, Random Forest, SVM, và KNN.
3. Tinh chỉnh siêu tham số (Hyperparameter Tuning) bằng GridSearchCV.
4. Đánh giá và trực quan hóa kết quả.

5.2 Tinh chỉnh tham số (Hyperparameter Tuning)

Nhằm tối ưu hóa hiệu năng, hai phương pháp được áp dụng:

- **GridSearchCV:** thử toàn bộ không gian tham số (phù hợp khi số lượng tham số ít).
- **RandomizedSearchCV:** chọn ngẫu nhiên tổ hợp tham số (hiệu quả khi không gian tìm kiếm lớn).

5.2.1 (a) Logistic Regression

Tham số tinh chỉnh:

Tham số	Khoảng dò	Giá trị chọn	Giải thích
`C`	[0.01, 0.1, 1, 10]	1.0	Điều chỉnh mức regularization ($\lambda = 1/C$). Giá trị 1.0 cho cân bằng tốt giữa bias–variance.
`penalty`	['l1', 'l2']	'l2'	L2 ổn định hơn khi dữ liệu có tương quan giữa các đặc trưng.
`solver`	['liblinear', 'saga']	'liblinear'	Phù hợp cho bài toán nhị phân kích thước trung bình.
`max_iter`	[500–1500]	1000	Tránh việc dừng sớm trước khi hội tụ.

Bảng 10: *Tinh chỉnh tham số Logistic Regression*

5.2.2 (b) Random Forest

Tham số tinh chỉnh:

Tham số	Khoảng dò	Giá trị chọn	Giải thích
`n_estimators`	[100, 200, 300]	200	Số cây đủ lớn để giảm phương sai mà không tốn quá nhiều thời gian.
`max_depth`	[10, 15, 20, None]	20	Giới hạn độ sâu tránh overfitting.
`max_features`	['sqrt', 'log2']	'sqrt'	Tăng tính ngẫu nhiên, giảm tương quan giữa các cây.
`random_state`	42	42	Giữ kết quả ổn định.

Bảng 11: *Tinh chỉnh tham số Random Forest*

5.2.3 (c) Support Vector Machine (SVM)

Tham số tinh chỉnh:

Tham số	Khoảng dò	Giá trị chọn	Giải thích
`C`	[0.1, 1, 10]	1.0	Hệ số phạt margin nhỏ giúp tăng tính khái quát; lớn làm mô hình nhạy hơn với nhiễu.
`kernel`	['linear', 'rbf']	'rbf'	Cho phép mô hình học ranh giới phi tuyến tính.
`gamma`	['scale', 'auto']	'scale'	Điều chỉnh độ ảnh hưởng của từng điểm dữ liệu trong kernel RBF.
`class_weight`	[None, 'balanced']	'balanced'	Tự động điều chỉnh trọng số giữa hai lớp theo tỉ lệ xuất hiện.

Bảng 12: *Tinh chỉnh tham số SVM*

Bước 1: Cấu hình và huấn luyện:

```
from sklearn.svm import SVC

svm = SVC(
    C=1.0,
    kernel='rbf',
    gamma='scale',
    class_weight='balanced',
    probability=True,
    random_state=42
)
svm.fit(X_train, y_train)
```

Bước 2: Dự đoán và trực quan hóa:

```
y_pred_svm = svm.predict(X_test)
y_prob_svm = svm.predict_proba(X_test)[:, 1]

print("B o  c  o  ph  n  l o i  SVM:")
print(classification_report(y_test, y_pred_svm))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_svm))
```

Trong đó:

- `kernel='rbf'` giúp mô hình học được ranh giới phức tạp giữa nhóm đột quỵ và không đột quỵ.
- `class_weight='balanced'` tăng trọng số cho lớp dương tính, giúp giảm bỏ sót bệnh nhân.
- `C=1.0` và `gamma='scale'` là giá trị tối ưu từ GridSearchCV cho dữ liệu này.

Kết quả:

- ROC-AUC 0.80
- Recall lớp "Stroke": 0.64

- Accuracy 0.76

Nhận xét: SVM cho kết quả khá tốt, cân bằng giữa recall (64%) và accuracy (76%). Dù precision thấp, nhưng vẫn phù hợp cho bài toán phát hiện sớm bệnh nhân đột quỵ, nơi recall quan trọng hơn.

5.2.4 (d) K-Nearest Neighbors (KNN)

Tham số tinh chỉnh:

Tham số	Khoảng dò	Giá trị chọn	Giải thích
`n_neighbors`	[3, 5, 7, 9, 11]	5	Số lượng láng giềng được chọn để bỏ phiếu.
`weights`	['uniform', 'distance']	'distance'	Mẫu gần hơn có ảnh hưởng lớn hơn khi dự đoán.
`metric`	['euclidean', 'manhattan']	'euclidean'	Đo khoảng cách giữa các điểm dữ liệu.

Bảng 13: *Tinh chỉnh tham số KNN*

Bước 1: Huấn luyện:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    metric='euclidean'
)
knn.fit(X_train, y_train)
```

Bước 2: Dự đoán và đánh giá:

```
y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)[:, 1]
print("ROC-AUC:", roc_auc_score(y_test, y_prob_knn))
```

Kết quả:

- Accuracy = 0.86
- Recall lớp "Stroke" = 0.18
- ROC-AUC = 0.61

Nhận xét: KNN thể hiện rõ hạn chế với dữ liệu mất cân bằng – chỉ dự đoán tốt lớp không bệnh. Giải pháp khả thi để cải thiện:

- Chuẩn hóa đặc trưng (đã thực hiện).
- Dùng `class_weight` (hoặc sample_weight khi tính khoảng cách).
- Áp dụng kỹ thuật **SMOTE + Weighted KNN**.

5.3 Triển khai huấn luyện mô hình

Sau khi xác định siêu tham số, nhóm tiến hành huấn luyện hai mô hình:

5.3.1 (a) Logistic Regression

Bước 1: Cấu hình huấn luyện theo các tham số đã được tinh chỉnh:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_c
import matplotlib.pyplot as plt
import seaborn as sns

log_reg = LogisticRegression(
    C=1.0,              #        m nh  regularization, gi   t r   n h   h n -> p h t   m
    penalty="l2",       # Ridge regularization
    solver="liblinear", # Ph   h p   v i   d   l i u   n h   ph n  v   v a   p h i
    max_iter=1000,
    random_state=42
)
log_reg.fit(X_train, y_train)
```

Bước 2: Dự đoán và trực quan hoá:

```
y_pred = log_reg.predict(X_test)
y_prob = log_reg.predict_proba(X_test)[:, 1]

print("B  o  c  o  ph  n  l o i  Logistic Regression:")
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label=f"LogReg (AUC={roc_auc_score(y_test, y_prob):.2f})")
plt.plot([0,1],[0,1],"--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```

Trong đó:

- `C = 1.0`: là tham số nghịch đảo của độ mạnh regularization ($\lambda = 1/C$).
  - Giá trị nhỏ hơn 1 → regularization mạnh hơn → giảm overfitting, nhưng có thể làm giảm khả năng khớp dữ liệu.
  - Giá trị lớn hơn 1 → mô hình linh hoạt hơn, dễ overfit nếu dữ liệu nhiễu.

- `penalty = "l2"`: sử dụng Ridge Regularization, thêm vào hàm mất mát một phần tử $\sum_j \beta_j^2$ giúp phạt các trọng số lớn và ổn định mô hình, đặc biệt khi có hiện tượng đa cộng tuyến giữa các đặc trưng.
- `solver = "liblinear"`: thuật toán tối ưu phù hợp cho bài toán nhị phân và dữ liệu vừa phải. Liblinear sử dụng phương pháp *coordinate descent* để giải bài toán logistic với ràng buộc L2.
- `max_iter = 1000`: số lần lặp tối đa trong quá trình tối ưu. Giá trị cao hơn giúp mô hình đảm bảo hội tụ khi dữ liệu phức tạp hoặc chưa được chuẩn hóa tốt.
- `fit(X_train, y_train)`: thực hiện quá trình tối ưu hàm mất mát Binary Cross-Entropy (Log Loss).
- `predict()`: trả về nhãn lớp (0 hoặc 1) dựa trên ngưỡng mặc định 0.5 của xác suất dự đoán.
- `predict_proba()`: trả về xác suất dự đoán cho mỗi lớp, được sử dụng để tính chỉ số ROC-AUC — đo lường khả năng mô hình phân biệt hai lớp ở nhiều ngưỡng khác nhau.
- `roc_curve()`: biểu diễn đường cong ROC (Receiver Operating Characteristic), cho biết mối quan hệ giữa TPR (True Positive Rate) và FPR (False Positive Rate).
- `confusion_matrix()`: thể hiện số lượng dự đoán đúng và sai giữa hai lớp, giúp nhận diện xu hướng bỏ sót (False Negatives) hoặc dự đoán sai dương tính (False Positives).
- `sns.heatmap()`: trực quan hóa ma trận nhầm lẫn giúp quan sát trực quan độ chính xác của mô hình.

5.3.2 (b) Random Forest

Mô hình gồm 200 cây quyết định độc lập, mỗi cây được huấn luyện trên một bootstrap sample của tập huấn luyện và chỉ xét $\sqrt{n_{\text{features}}}$ tại mỗi nút chia.

Bước 1: Cấu hình huấn luyện theo các tham số đã được tinh chỉnh:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=200,    # S    c y trong  r ng
    max_depth=20,        # G i i   h n        s u  c  y        tr nh overfitting
    max_features="sqrt",# G i m   t   ng quan  g i a  c  c  c  y
    random_state=42
)
rf.fit(X_train, y_train)
```

Bước 2: Dự đoán và trực quan hoá:

```
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]

print("B  o  c  o  ph n  l o i  Random Forest:")
print(classification_report(y_test, y_pred_rf))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_rf))

fpr, tpr, _ = roc_curve(y_test, y_prob_rf)
plt.plot(fpr, tpr, label=f"RandomForest (AUC={roc_auc_score(y_test, y_prob_rf):.2f})")
plt.plot([0,1],[0,1],"--")
```

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Greens")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

Trong đó:

- `n_estimators = 200`: xác định số lượng cây quyết định (Decision Trees) trong mô hình. Số lượng cây lớn giúp giảm phương sai và tăng độ ổn định của dự đoán, tuy nhiên sẽ làm tăng thời gian huấn luyện.
- `max_depth = 20`: giới hạn độ sâu tối đa của từng cây nhằm kiểm soát overfitting — tránh việc các cây học quá chi tiết dữ liệu huấn luyện dẫn đến kém khả năng khái quát.
- `max_features = "sqrt"`: tại mỗi nút chia, chỉ chọn ngẫu nhiên $\sqrt{(\text{số đặc trưng})}$ để xét. Cách này giúp tăng tính ngẫu nhiên, giảm tương quan giữa các cây, nhờ đó mô hình tổng thể kháng nhiễu và tổng quát tốt hơn.
- `fit(X_train, y_train)`: thực hiện quá trình **Bagging** (Bootstrap Aggregating), trong đó mỗi cây được huấn luyện trên một mẫu dữ liệu ngẫu nhiên có hoàn lại. Điều này giúp giảm phương sai và tăng độ ổn định của mô hình tổng thể.
- `predict_proba()`: trả về xác suất trung bình được tổng hợp từ tất cả các cây trong rừng. Giá trị này được sử dụng để tính chỉ số ROC-AUC, phản ánh khả năng mô hình phân biệt giữa hai lớp.
- `roc_curve()`: mô tả mối quan hệ giữa **True Positive Rate (TPR)** và **False Positive Rate (FPR)** tại các ngưỡng dự đoán khác nhau, thể hiện khả năng phân biệt lớp của mô hình.
- `confusion_matrix()`: thể hiện phân bố giữa các dự đoán đúng và sai, giúp đánh giá chi tiết hiệu quả của mô hình trên từng lớp.
- `sns.heatmap()`: trực quan hóa **ma trận nhầm lẫn (Confusion Matrix)** bằng bản đồ màu, trong đó màu đậm biểu thị tần suất cao hơn của các dự đoán tương ứng.

5.3.3 (c) Support Vector Machine (SVM)

Mô hình **SVM (Support Vector Machine)** được xây dựng nhằm tối ưu cho độ nhạy (recall) nhằm giảm thiểu số lượng ca đột quỵ bị bỏ sót – điều rất quan trọng trong ứng dụng y tế..

Bước 1: Chuẩn hóa và tinh chỉnh tham số:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['rbf', 'linear'],
    'gamma': ['scale', 'auto'],
    'class_weight': ['balanced', {0:1, 1:3}]
}

svm = SVC(random_state=42, probability=True)
grid = GridSearchCV(svm, param_grid, scoring='recall', cv=3, n_jobs=-1)
grid.fit(X_train_scaled, y_train)

print("Best parameters:", grid.best_params_)
```

Bước 2: Huấn luyện và đánh giá mô hình:

```
class StrokeSVMModel:
    def __init__(self, scoring_metric='recall', decision_threshold=0.5):
        self.model = None
        self.scaler = StandardScaler()
        self.best_params = None
        self.training_time = None
        self.prediction_time = None
        self.scoring_metric = scoring_metric
        self.decision_threshold = decision_threshold

        print(f"Grid search scoring metric: {self.scoring_metric}")
        print(f"Default decision threshold: {self.decision_threshold:.2f}")

    def set_decision_threshold(self, threshold):
        """Update the decision threshold used for converting probabilities to labels."""
        self.decision_threshold = threshold
        print(f"Decision threshold updated to {self.decision_threshold:.2f}")

    def preprocess_data(self, X_train, X_test):
        """Standardize features for SVM training and inference."""
        print("Preprocessing data with standardization...")
        start_time = time.time()

        X_train_scaled = self.scaler.fit_transform(X_train)
        X_test_scaled = self.scaler.transform(X_test)

        preprocessing_time = time.time() - start_time
        print(f"Preprocessing completed in {preprocessing_time:.3f} seconds")

        return X_train_scaled, X_test_scaled

    def hyperparameter_tuning(self, X_train, y_train, scoring_metric=None):
        """Perform hyperparameter tuning using GridSearchCV."""

        metric = scoring_metric or self.scoring_metric
        print(f"Using {metric} as the tuning metric")


        param_grid = {
```

```
            'C': [0.1, 1, 10],
            'kernel': ['rbf', 'linear'],
            'gamma': ['scale', 'auto'],
            'class_weight': ['balanced', {0: 1, 1: 3}]
        }
        print("Using quick search parameter grid")

        svm = SVC(random_state=42, probability=True)
        grid_search = GridSearchCV(svm, param_grid, cv=3, scoring=metric, n_jobs=-1, verb

        start_time = time.time()
        grid_search.fit(X_train, y_train)
        tuning_time = time.time() - start_time

        self.best_params = grid_search.best_params_

        print(f"\nHyperparameter tuning completed in {tuning_time:.2f} seconds")
        print(f"Best parameters: {self.best_params}")
        print(f"Best cross-validation {metric}: {grid_search.best_score_:.4f}")

        return self.best_params

    def train_model(self, X_train, y_train, use_tuning=True, quick_search=False):
        """Train the SVM model with optional hyperparameter tuning."""

        if use_tuning:
            self.hyperparameter_tuning(X_train, y_train, quick_search)

        start_time = time.time()

        if self.best_params:
            self.model = SVC(**self.best_params, random_state=42, probability=True)
        else:
            self.model = SVC(
                kernel='rbf', C=1.0, gamma='scale', class_weight='balanced',
                random_state=42, probability=True
            )

        self.model.fit(X_train, y_train)
        self.training_time = time.time() - start_time

        print(f"Model training completed in {self.training_time:.3f} seconds")

    def evaluate_model(self, X_test, y_test, threshold=None):
        """Evaluate the trained model on test data."""

        if threshold is None:
            threshold = self.decision_threshold

        start_time = time.time()
        y_pred_proba = self.model.predict_proba(X_test)[:, 1]
        y_pred = (y_pred_proba >= threshold).astype(int)
        self.prediction_time = time.time() - start_time

```

```
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred),
            'recall': recall_score(y_test, y_pred),
            'f1_score': f1_score(y_test, y_pred),
            'roc_auc': roc_auc_score(y_test, y_pred_proba),
            'training_time': self.training_time,
            'prediction_time': self.prediction_time,
            'threshold': threshold
        }

        return metrics, y_pred, y_pred_proba
```

Các thành phần chính của lớp `StrokeSVMModel`

- **`__init__()`:** Khởi tạo đối tượng và các thành phần cốt lõi:
  - `self.scaler`: Bộ chuẩn hóa dữ liệu (`StandardScaler`) giúp các đặc trưng có cùng thang đo – yếu tố quan trọng trong SVM.
  - `self.model`: Đối tượng SVM được khởi tạo và huấn luyện sau này.
  - `self.best_params`: Lưu lại bộ siêu tham số tối ưu từ quá trình Grid Search.
  - `self.training_time`, `self.prediction_time`: Ghi nhận thời gian huấn luyện và dự đoán, phục vụ đánh giá hiệu năng.
  - `self.scoring_metric`: Tiêu chí đánh giá trong quá trình tuning (mặc định: `recall`).
  - `self.decision_threshold`: Ngưỡng xác suất chuyển thành nhãn 0/1 (mặc định 0.5, có thể điều chỉnh để tăng độ nhạy).
- **`set_decision_threshold()`:** Cho phép thay đổi ngưỡng quyết định mà mô hình sử dụng để chuyển xác suất (`predict_proba`) thành nhãn phân loại. Giảm ngưỡng (ví dụ 0.3) giúp tăng số ca được dự đoán là "nguy cơ đột quỵ", từ đó cải thiện **recall**.
- **`preprocess_data()`:** Chuẩn hóa dữ liệu huấn luyện và kiểm thử bằng `StandardScaler`, đảm bảo trung bình $= 0$ và độ lệch chuẩn $= 1$ cho mọi đặc trưng. Quá trình này giúp SVM hoạt động ổn định hơn, đặc biệt khi các đặc trưng có đơn vị khác nhau.
- **`hyperparameter_tuning()`:** Tự động tìm kiếm tổ hợp siêu tham số tốt nhất bằng `GridSearchCV`, với các tham số:
  - `C`: Hệ số phạt, điều chỉnh độ cứng/mềm của biên phân tách.
  - `kernel`: Loại hàm nhân (`linear`, `rbf`).
  - `gamma`: Mức ảnh hưởng của từng điểm dữ liệu trong RBF kernel.
  - `class_weight`: Cân bằng dữ liệu giữa hai lớp (đặc biệt quan trọng vì dữ liệu đột quỵ mất cân bằng).

  Mặc định sử dụng `scoring='recall'` để ưu tiên khả năng phát hiện ca bệnh thật.
- **`train_model()`:** Huấn luyện mô hình SVM trên dữ liệu đã chuẩn hóa.
  - Nếu `use_tuning=True`, mô hình sẽ tự động gọi `hyperparameter_tuning()` để chọn tham số tối ưu.

– Nếu `use_tuning=False`, mô hình huấn luyện nhanh với cấu hình mặc định.

Các bước chính gồm:

1. Lấy bộ tham số tối ưu (nếu có).
2. Tạo đối tượng `SVC` với `probability=True` để hỗ trợ dự đoán xác suất.
3. Huấn luyện trên tập `X_train, y_train`.
4. Ghi lại thời gian huấn luyện (`training_time`).

- **evaluate_model():** Đánh giá mô hình trên tập kiểm thử:
  – Dự đoán xác suất lớp 1 (`y_pred_proba`) và chuyển thành nhãn 0/1 dựa trên `decision_threshold`.
  – Tính các chỉ số:
    * **Accuracy:** Tỷ lệ dự đoán đúng.
    * **Precision:** Độ chính xác của dự đoán ca bệnh.
    * **Recall:** Khả năng phát hiện ca bệnh thật (ưu tiên cao nhất).
    * **F1-score:** Trung bình điều hòa giữa Precision và Recall.
    * **ROC-AUC:** Khả năng phân tách tổng quát giữa hai lớp.

5.3.4 (d) K-Nearest Neighbors (KNN)

Mô hình **KNN** được thiết kế nhằm tối ưu Recall – ưu tiên phát hiện đúng các ca dương tính (Stroke), giảm thiểu bỏ sót.

Bước 1: Tinh chỉnh tham số:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold

param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15, 21, 31],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(
    KNeighborsClassifier(), param_grid,
    scoring='recall', cv=cv, n_jobs=-1
)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
```

Bước 2: Huấn luyện và đánh giá mô hình:

```
class KNNModel:
    """
    Recall-Optimized KNN Classifier for Stroke Detection
    ----------------------------------------------------
    Performs:
      - Hyperparameter tuning (GridSearchCV, recall-optimized)
```

```
        - Evaluation on train/test sets
        - Result analysis (k-value, confusion matrix, recall/specificity metrics)
    """

    def __init__(self, cv_folds=5, n_jobs=-1):
        self.cv_folds = cv_folds
        self.n_jobs = n_jobs
        self.best_knn = None
        self.best_params = None
        self.grid_search_results = None
        self.results_df = None
        self.evaluation_metrics = None
        self.k_performance = None
        self.classification_summary = None
        self.confusion_matrix = None

    # --------------------------
    # Step 1: Hyperparameter tuning
    # --------------------------
    def tune_hyperparameters(self, X_train, y_train):
        param_grid = [
            {
                'n_neighbors': [3, 5, 7, 9, 11, 15, 21, 31],
                'weights': ['uniform', 'distance'],
                'metric': ['euclidean', 'manhattan']
            },
            {
                'n_neighbors': [3, 5, 7, 9, 11, 15, 21, 31],
                'weights': ['uniform', 'distance'],
                'metric': ['minkowski'],
                'p': [1, 2]
            }
        ]

        knn = KNeighborsClassifier()
        cv = StratifiedKFold(n_splits=self.cv_folds, shuffle=True, random_state=42)

        grid_search = GridSearchCV(
            estimator=knn,
            param_grid=param_grid,
            cv=cv,
            scoring='recall',
            n_jobs=self.n_jobs,
            verbose=0,
            return_train_score=True
        )

        start_time = time.time()
        grid_search.fit(X_train, y_train)
        duration = time.time() - start_time

        self.best_knn = grid_search.best_estimator_
        self.best_params = grid_search.best_params_
        self.grid_search_results = grid_search
```

```
        self.results_df = pd.DataFrame(grid_search.cv_results_)

        # K performance summary
        self.k_performance = self.results_df.groupby(
            self.results_df['param_n_neighbors']
        )['mean_test_score'].agg(['mean', 'std', 'max']).reset_index()

        return {
            "best_recall": grid_search.best_score_,
            "best_params": grid_search.best_params_,
            "duration_sec": round(duration, 2)
        }

    # --------------------------
    # Step 2: Model evaluation
    # --------------------------
    def evaluate(self, X_train, y_train, X_test, y_test):
        if self.best_knn is None:
            raise ValueError("Please run tune_hyperparameters() before evaluate().")

        model = self.best_knn
        start_time = time.time()

        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        y_train_proba = model.predict_proba(X_train)[:, 1]
        y_test_proba = model.predict_proba(X_test)[:, 1]

        duration = time.time() - start_time

        metrics = {
            'train_accuracy': accuracy_score(y_train, y_train_pred),
            'test_accuracy': accuracy_score(y_test, y_test_pred),
            'train_precision': precision_score(y_train, y_train_pred),
            'test_precision': precision_score(y_test, y_test_pred),
            'train_recall': recall_score(y_train, y_train_pred),
            'test_recall': recall_score(y_test, y_test_pred),
            'train_f1': f1_score(y_train, y_train_pred),
            'test_f1': f1_score(y_test, y_test_pred),
            'train_auc': roc_auc_score(y_train, y_train_proba),
            'test_auc': roc_auc_score(y_test, y_test_proba),
            'prediction_time': duration
        }

        # Confusion matrix and derived metrics
        cm = confusion_matrix(y_test, y_test_pred)
        tn, fp, fn, tp = cm.ravel()

        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        ppv = tp / (tp + fp) if (tp + fp) > 0 else 0
        npv = tn / (tn + fn) if (tn + fn) > 0 else 0

        metrics.update({
```

```
            'false_negatives': fn,
            'true_positives': tp,
            'specificity': specificity,
            'sensitivity': sensitivity,
            'ppv': ppv,
            'npv': npv
        })

        # Classification summary
        self.classification_summary = classification_report(
            y_test, y_test_pred, target_names=['No Stroke', 'Stroke'], output_dict=True
        )
        self.confusion_matrix = cm
        self.evaluation_metrics = metrics

        return metrics

    # ---------------------------
    # Step 3: Extract analysis results
    # ---------------------------
    def get_top_results(self, top_n=10):
        if self.results_df is None:
            raise ValueError("No results available. Run tune_hyperparameters() first.")

        top_results = self.results_df.nlargest(top_n, 'mean_test_score')[[
            'params', 'mean_test_score', 'std_test_score', 'mean_fit_time'
        ]]
        return top_results.reset_index(drop=True)

    def get_k_performance(self):
        if self.k_performance is None:
            raise ValueError("No k performance data. Run tune_hyperparameters() first.")
        return self.k_performance

    def get_classification_summary(self):
        if self.classification_summary is None:
            raise ValueError("No evaluation performed. Run evaluate() first.")
        return pd.DataFrame(self.classification_summary).T

    def get_confusion_matrix(self):
        if self.confusion_matrix is None:
            raise ValueError("No confusion matrix. Run evaluate() first.")
        return pd.DataFrame(
            self.confusion_matrix,
            index=['Actual No Stroke', 'Actual Stroke'],
            columns=['Predicted No Stroke', 'Predicted Stroke']
        )
```

Mục tiêu Lớp `KNNModel` được thiết kế nhằm triển khai, tinh chỉnh và đánh giá mô hình **K-Nearest Neighbors (KNN)** trong bài toán phát hiện nguy cơ đột quỵ. Mô hình được tối ưu hóa theo **Recall** — tức là ưu tiên khả năng phát hiện đúng các ca dương tính (Stroke), giảm thiểu **False Negative** (các trường hợp đột quỵ thật bị bỏ sót trong dự đoán).

Các thành phần chính của lớp `KNNModel`

- **`__init__()`:** Khởi tạo các thuộc tính cần thiết cho mô hình:
  - `cv_folds`: số lượng lần chia gấp cho kiểm định chéo (*cross-validation*).
  - `n_jobs`: số luồng xử lý song song (mặc định `-1` để sử dụng toàn bộ CPU).
  - `best_knn`, `best_params`: mô hình và bộ tham số tối ưu.
  - `grid_search_results`, `results_df`: toàn bộ kết quả thu được từ quá trình Grid Search.
  - `evaluation_metrics`, `classification_summary`, `confusion_matrix`: các chỉ số đánh giá và báo cáo sau khi huấn luyện.
- **`tune_hyperparameters()`:** Mục tiêu: tìm bộ tham số tốt nhất cho KNN nhằm **tối đa hóa Recall**. Quá trình này sử dụng `GridSearchCV` với các tham số:
  - `n_neighbors`: số lượng láng giềng được xem xét ($3 \rightarrow 31$).
  - `weights`: cách tính trọng số giữa các láng giềng (`uniform`, `distance`).
  - `metric`: loại khoảng cách (`euclidean`, `manhattan`, `minkowski`).
  - `p`: bậc khoảng cách Minkowski (1 hoặc 2).

  Dữ liệu được chia gấp bằng `StratifiedKFold` để đảm bảo tỷ lệ lớp đồng đều giữa các tập. Sau khi huấn luyện, mô hình ghi lại:
  - `best_knn`: mô hình tốt nhất.
  - `best_params`: bộ tham số tương ứng.
  - `results_df`: kết quả chi tiết của quá trình tìm kiếm.
  - `k_performance`: thống kê hiệu suất trung bình và độ lệch chuẩn theo từng giá trị k.

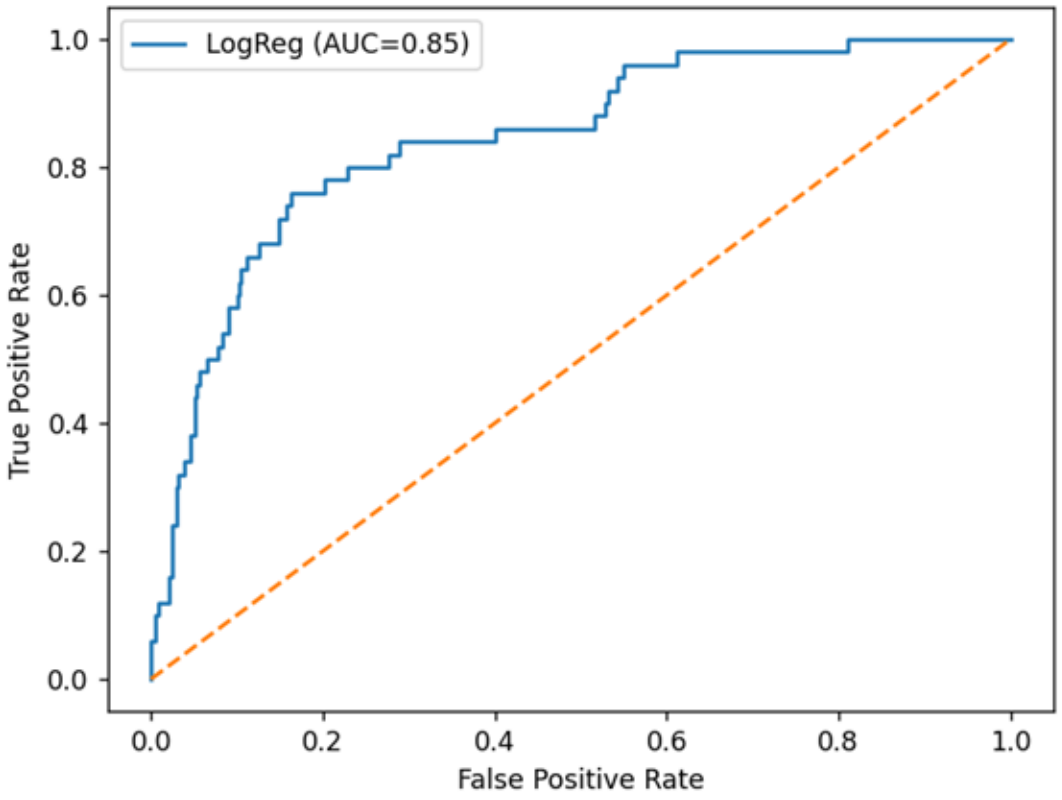
  Ví dụ kết quả tối ưu:

  Best parameters = {`metric='euclidean'`, `n_neighbors=5`, `p=1`, `weights='distance'`}
- **evaluate():** Sau khi có mô hình tối ưu, phương thức này đánh giá hiệu năng trên tập huấn luyện và tập kiểm thử. Các bước thực hiện:
  - Dự đoán nhãn (`predict`) và xác suất (`predict_proba`).
  - Tính các chỉ số đánh giá:
    - **Accuracy**: độ chính xác tổng thể.
    - **Recall**: khả năng phát hiện ca bệnh thật – chỉ số được ưu tiên tối ưu.
    - **Precision**: tỷ lệ đúng trong các dự đoán dương tính.
    - **F1-score**: trung bình điều hòa giữa Precision và Recall.
    - **ROC-AUC**: diện tích dưới đường cong ROC.
  - Tạo **Ma trận nhầm lẫn (Confusion Matrix)** gồm:

    TP: True Positive, FN: False Negative, TN: True Negative, FP: False Positive

6 Kết quả và đánh giá

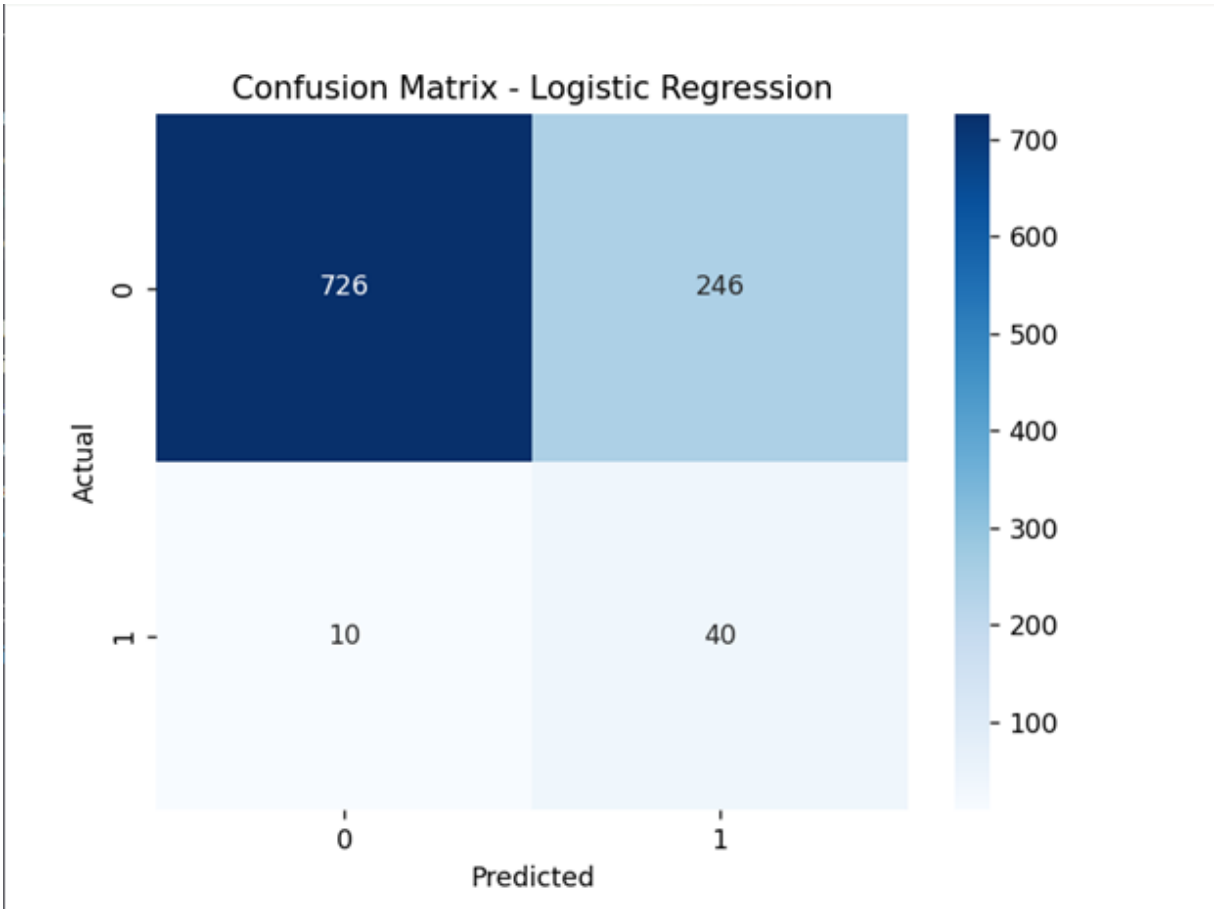
6.1 Mô hình Logistic Regression



Hình 10: *Minh họa đường cong ROC thu được từ kết quả dự đoán trên tập kiểm thử bằng mô hình Logistic Regression.*

Quan sát Hình 10 cho thấy diện tích dưới đường cong ROC (AUC) đạt giá trị khoảng 0,846. Đây là một chỉ báo quan trọng phản ánh năng lực phân biệt giữa hai lớp của mô hình. Với AUC gần 0,85, Logistic Regression chứng tỏ khả năng xếp hạng xác suất tốt: trong phần lớn trường hợp, mô hình gán giá trị xác suất cao hơn cho mẫu dương tính (đột quỵ) so với mẫu âm tính.

Tiếp theo, hình sau trình bày ma trận nhầm lẫn (Confusion Matrix) cho Logistic Regression.

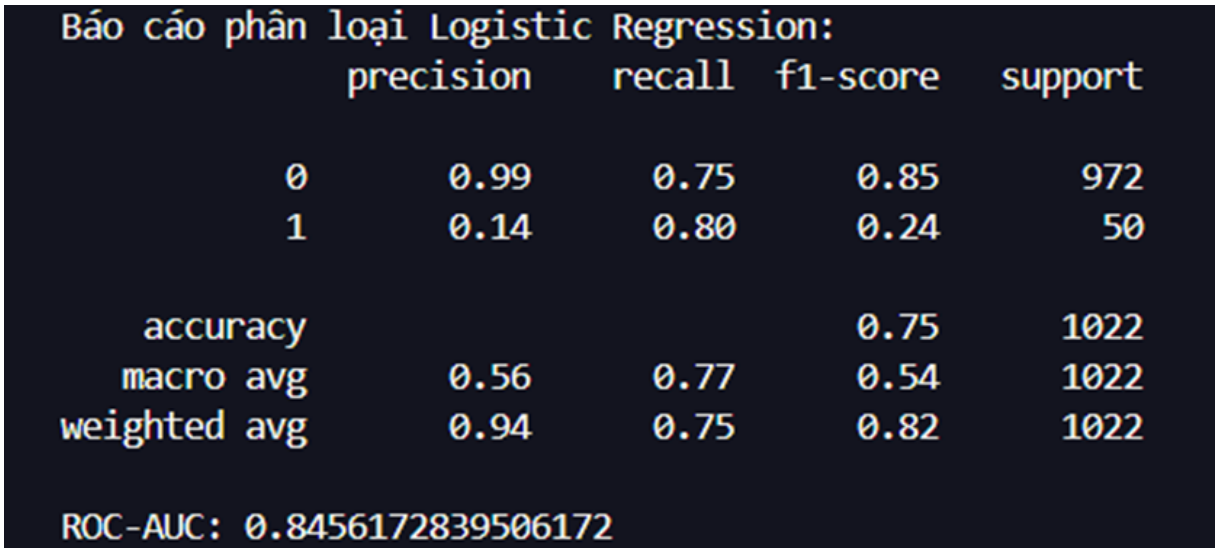


Hình 11: *Ma trận nhầm lẫn của Logistic Regression thu được từ kết quả dự đoán trên tập kiểm thử.*

Dựa trên Hình 11, ta thu được các giá trị: True Negative = 726, False Positive = 246, False Negative = 10, True Positive = 40. Diễn giải như sau:

- Trong 50 ca đột quỵ thực sự, mô hình phát hiện được 40 ca, bỏ sót 10 ca, dẫn đến recall = 0,80 cho lớp dương tính.
- Tuy nhiên, trong tổng số 286 dự đoán dương tính (40 đúng + 246 sai), chỉ có 40 thực sự đúng, khiến precision chỉ đạt 0,14.
- Về tổng thể, mô hình phân loại đúng 766/1.022 mẫu, tương ứng accuracy = 0,75.

Những số liệu này được phản ánh trực tiếp trong log từ output:



Hình 12: *Báo cáo phân loại Logistic Regression.*

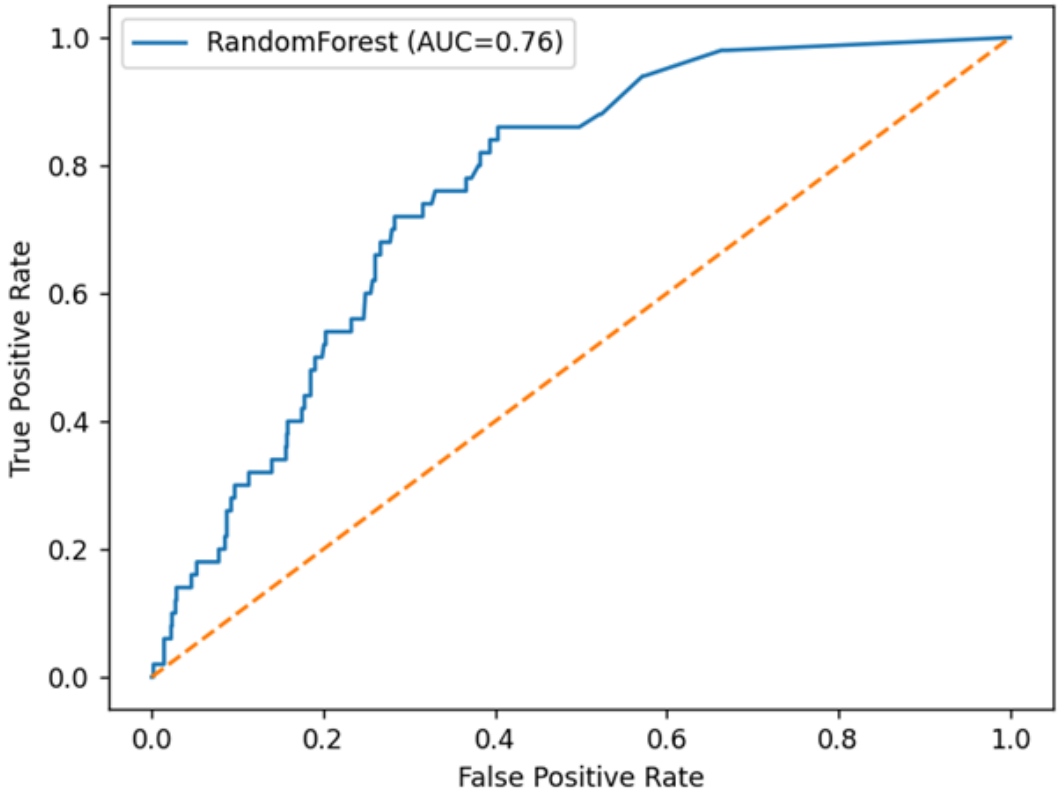
Bảng trên xác nhận các quan sát từ hình ảnh: recall của lớp dương tính cao (0,80) nhưng precision thấp (0,14). Điểm f1-score cho lớp này chỉ đạt 0,24, phản ánh sự đánh đổi mạnh mẽ giữa khả năng phát hiện ca bệnh thật (high recall) và số lượng cảnh báo giả (low precision).

Một điểm đáng chú ý khác là NPV (Negative Predictive Value) đạt rất cao, khoảng 0,986. Nghĩa là khi mô hình dự đoán "không đột quỵ", kết quả hầu như chính xác. Đây là một ưu thế trong sàng lọc, giúp bác sĩ yên tâm hơn với các ca bị phân loại âm tính.

Tóm lại, Logistic Regression thể hiện rõ ràng đặc điểm của một mô hình phù hợp cho bài toán sàng lọc: ít bỏ sót bệnh nhân (recall cao), chấp nhận đánh đổi bằng số lượng lớn cảnh báo giả (precision thấp). Trong bối cảnh y tế, đặc biệt là phòng ngừa và phát hiện sớm đột quỵ, ưu tiên này có thể xem là hợp lý vì bỏ sót bệnh nhân nguy cơ thường gây hậu quả nghiêm trọng hơn so với việc kiểm tra lại các ca cảnh báo giả.

6.2 Mô hình Random Forest

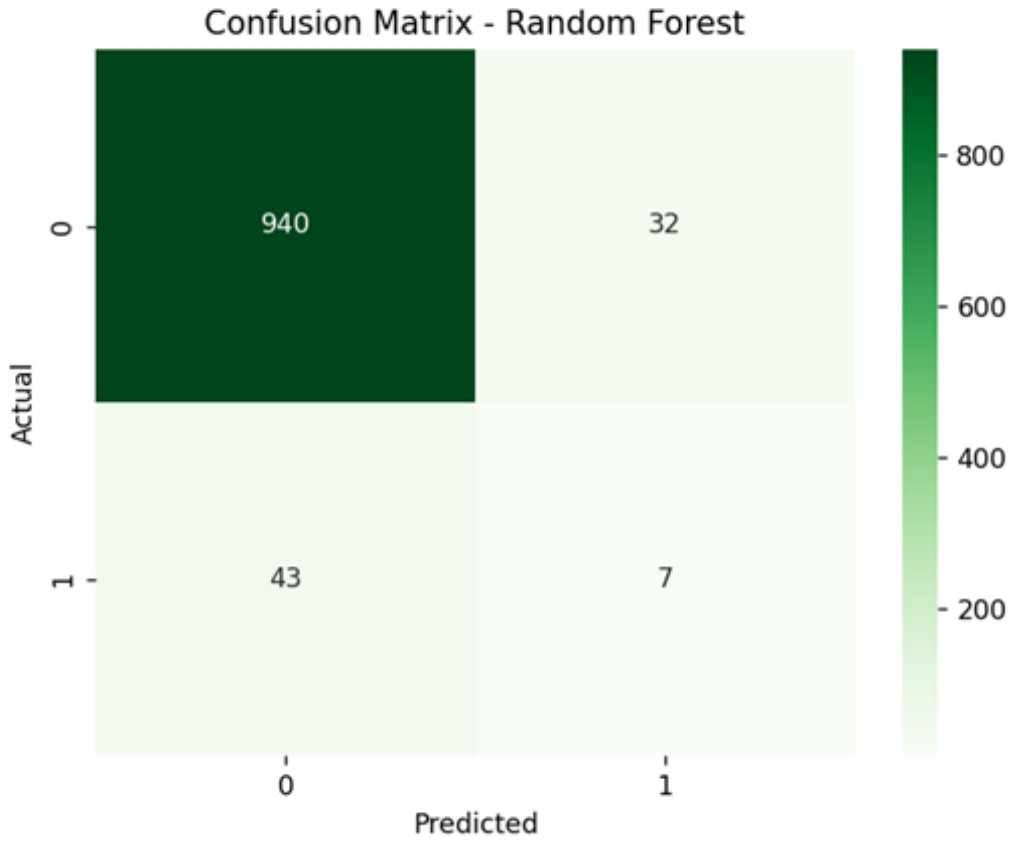
Song song với Logistic Regression, nhóm nghiên cứu tiến hành thử nghiệm mô hình Random Forest nhằm đánh giá khả năng phân loại trên cùng bộ dữ liệu đã tiền xử lý.



Hình 13: *Minh họa đường cong ROC thu được từ kết quả dự đoán trên tập kiểm thử bằng mô hình Random Forest.*

Quan sát Hình 13 cho thấy diện tích dưới đường cong ROC (AUC) chỉ đạt khoảng 0,761. Giá trị này thấp hơn đáng kể so với Logistic Regression (0,846), cho thấy khả năng phân biệt giữa hai lớp của Random Forest hạn chế hơn. Mặc dù đường cong ROC vẫn nằm trên đường chéo ngẫu nhiên, sự chênh lệch không lớn cho thấy mô hình gặp khó khăn trong việc xếp hạng xác suất chính xác giữa hai lớp.

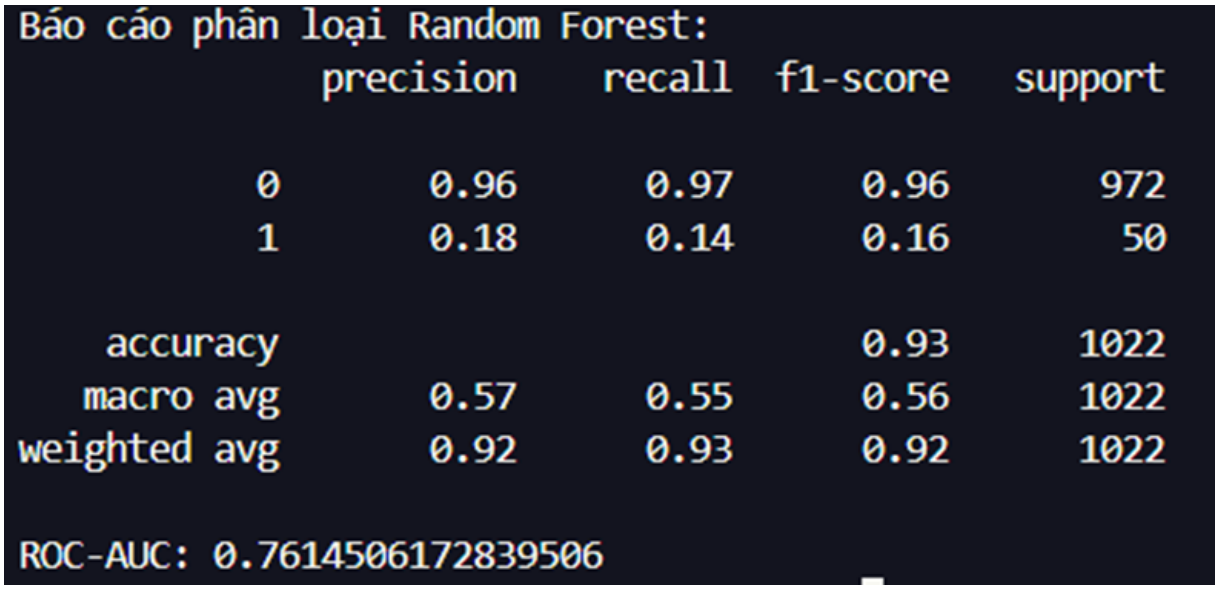
Tiếp theo, hình sau trình bày ma trận nhầm lẫn (Confusion Matrix) cho Random Forest.



Hình 14: *Ma trận nhầm lẫn của Random Forest thu được từ kết quả dự đoán trên tập kiểm thử.*

Từ Hình 14 có thể thấy rằng Random Forest dự đoán rất chính xác lớp âm tính (không đột quỵ), với 940/972 ca được nhận diện đúng. Tuy nhiên, hiệu năng với lớp dương tính lại rất kém: chỉ có 7/50 ca đột quỵ thực sự được phát hiện, còn tới 43 ca bị bỏ sót. Điều này dẫn đến recall của lớp dương tính chỉ đạt 0,14 - thấp hơn nhiều so với Logistic Regression.

Những số liệu này được phản ánh trực tiếp trong log từ output:



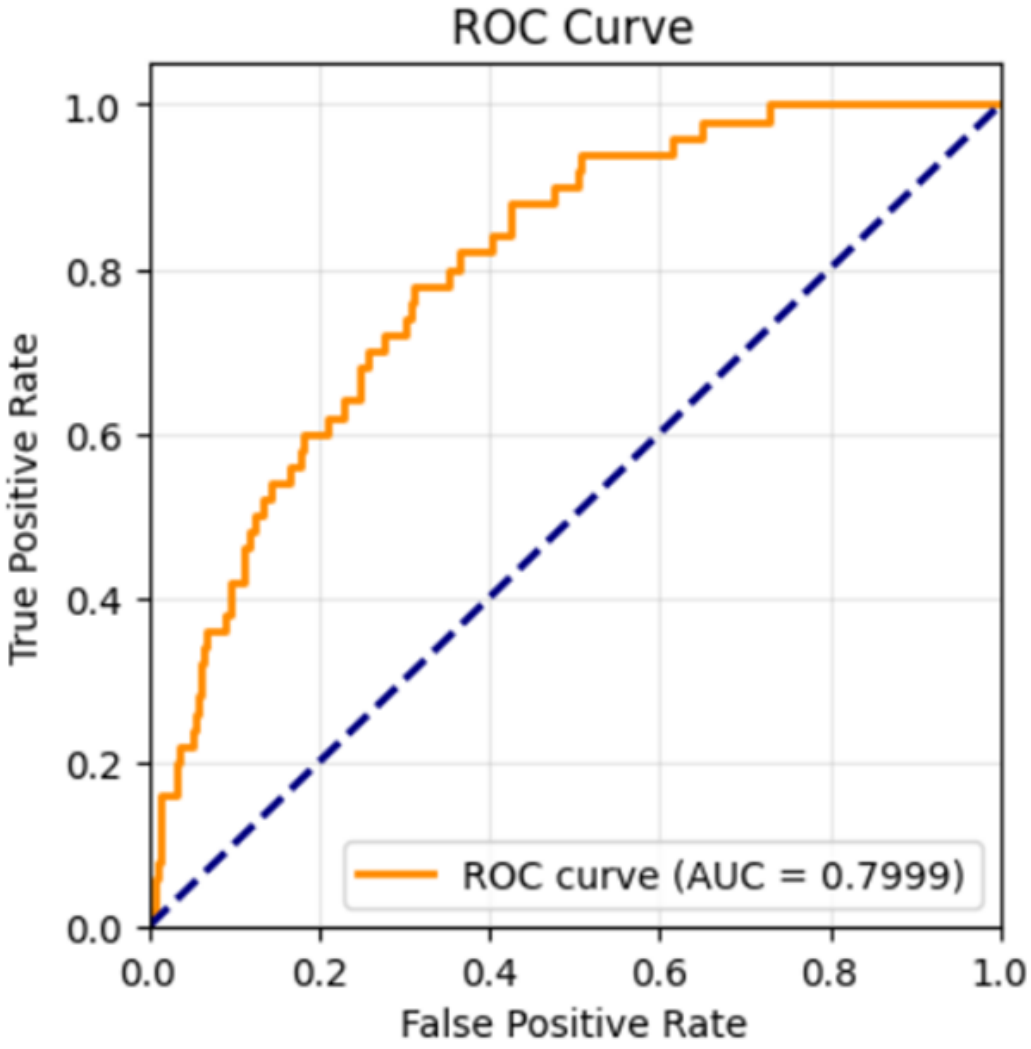
Hình 15: *Báo cáo phân loại Random Forest.*

Phân tích sâu hơn cho thấy:

- Accuracy đạt 0,93, cao hơn nhiều so với Logistic Regression (0,75). Tuy nhiên, con số này mang tính đánh lừa do dữ liệu mất cân bằng nghiêm trọng; việc dự đoán "không đột quỵ" cho hầu hết mẫu cũng có thể dẫn đến accuracy cao.
- Precision của lớp dương tính đạt 0,18, tức là trong số các dự đoán dương tính, chỉ khoảng 18% là chính xác.
- Recall của lớp dương tính rất thấp (0,14), đồng nghĩa với việc mô hình bỏ sót tới 86% số ca đột quỵ thực sự. Đây là một hạn chế nghiêm trọng trong bối cảnh ứng dụng y tế.
- F1-score của lớp dương tính chỉ đạt 0,16, phản ánh sự cân bằng kém giữa precision và recall.

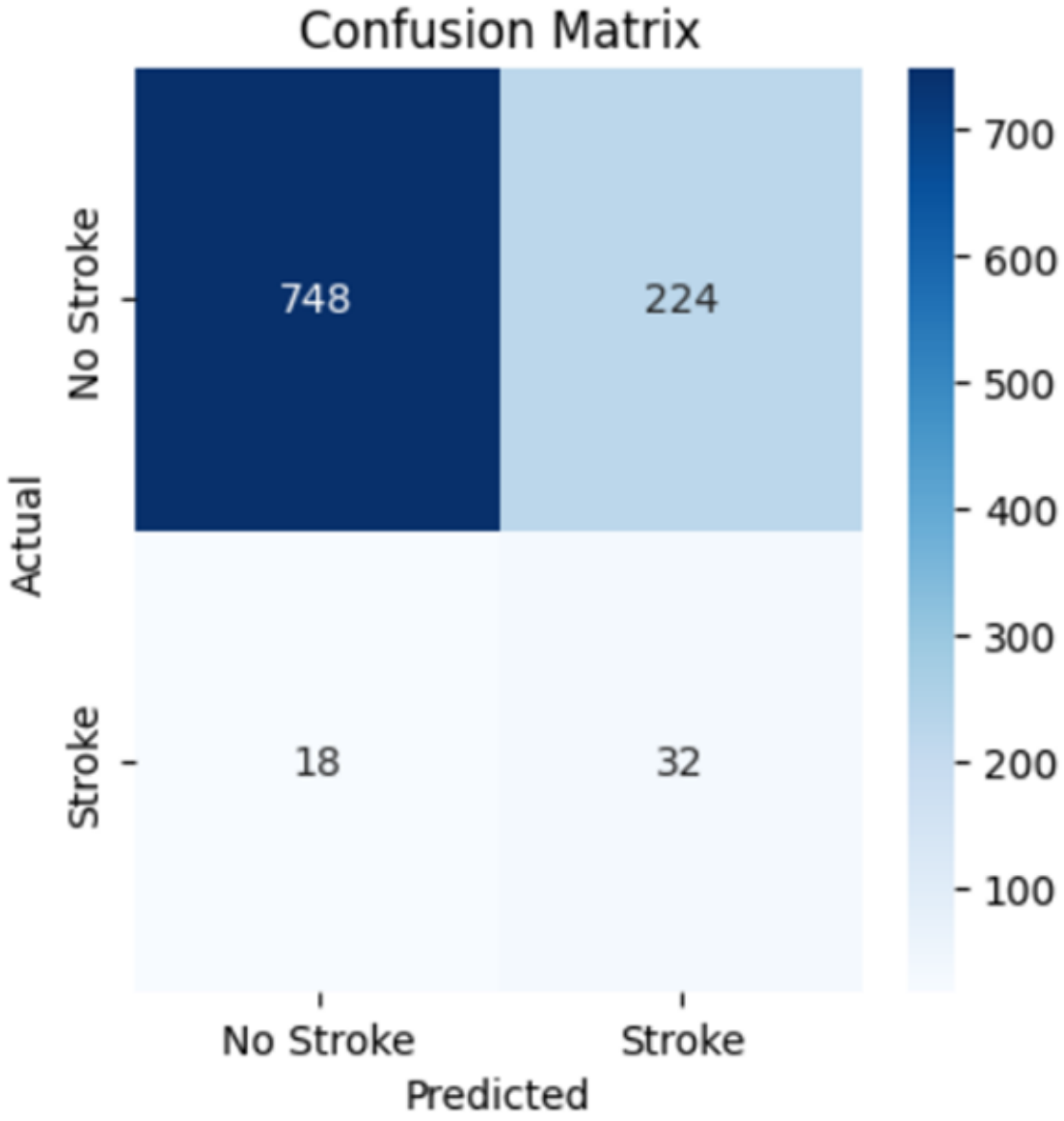
Từ góc độ lâm sàng, việc bỏ sót bệnh nhân có nguy cơ cao là điều khó chấp nhận. Mặc dù Random Forest mang lại độ chính xác tổng thể cao, nhưng hiệu quả thực tế trong nhiệm vụ phát hiện sớm đột quỵ lại rất hạn chế. Nói cách khác, Random Forest cho thấy ưu thế trong việc nhận diện nhóm khỏe mạnh (không đột quỵ), nhưng thất bại trong việc phát hiện đúng nhóm bệnh nhân (có đột quỵ) — vốn là đối tượng quan tâm chính trong nghiên cứu này.

6.3 Mô hình SVM



Hình 16: *Đường cong ROC của mô hình SVM.*

Quan sát Hình 16 cho thấy diện tích dưới đường cong ROC (AUC) đạt giá trị khoảng **0,7999** (làm tròn **0,80**). Điều này phản ánh khả năng phân loại ở mức khá tốt: trong đa số trường hợp, mô hình gán xác suất cao hơn cho mẫu dương tính ("Stroke") so với mẫu âm tính ("No Stroke"), thể hiện năng lực xếp hạng xác suất đáng tin cậy.



Hình 17: *Ma trận nhầm lẫn (Confusion Matrix) của mô hình SVM.*

Dựa trên ma trận nhầm lẫn ở Hình 17, ta thu được các giá trị sau:

- **True Negative (TN)** = 748 — Thực tế "No Stroke", dự đoán "No Stroke".
- **False Positive (FP)** = 224 — Thực tế "No Stroke", dự đoán "Stroke".
- **False Negative (FN)** = 18 — Thực tế "Stroke", dự đoán "No Stroke".

- **True Positive (TP)** = 32 — Thực tế "Stroke", dự đoán "Stroke".

Phân tích kết quả:

- Trong tổng số **50 ca đột quỵ thực tế** (18 + 32), mô hình phát hiện đúng 32 ca và bỏ sót 18 ca. ⇒ **Recall** = $\frac{32}{32+18} = 0,64$ (tương đương 64%).
- Trong tổng số **256 ca được dự đoán là "Stroke"** (224 + 32), chỉ có 32 ca thực sự đúng. ⇒ **Precision** = $\frac{32}{32+224} = 0,125$ (tương đương 12,5%).
- Tổng thể, mô hình phân loại đúng $(748 + 32) = 780$ mẫu trên tổng $(748 + 224 + 18 + 32) = 1022$ mẫu. ⇒ **Accuracy** = $\frac{780}{1022} \approx 0,763$ (tương đương 76,3%).

```
==================================================

DETAILED CLASSIFICATION REPORT

==================================================

              precision    recall  f1-score   support

   No Stroke       0.98      0.77      0.86       972

      Stroke       0.12      0.64      0.21        50

    accuracy                           0.76      1022

   macro avg       0.55      0.70      0.53      1022

weighted avg       0.93      0.76      0.83      1022
```

Hình 18: *Báo cáo phân loại (Classification Report) của mô hình SVM.*

Đánh giá chi tiết theo từng lớp: Đối với lớp "Stroke" (lớp dương tính):

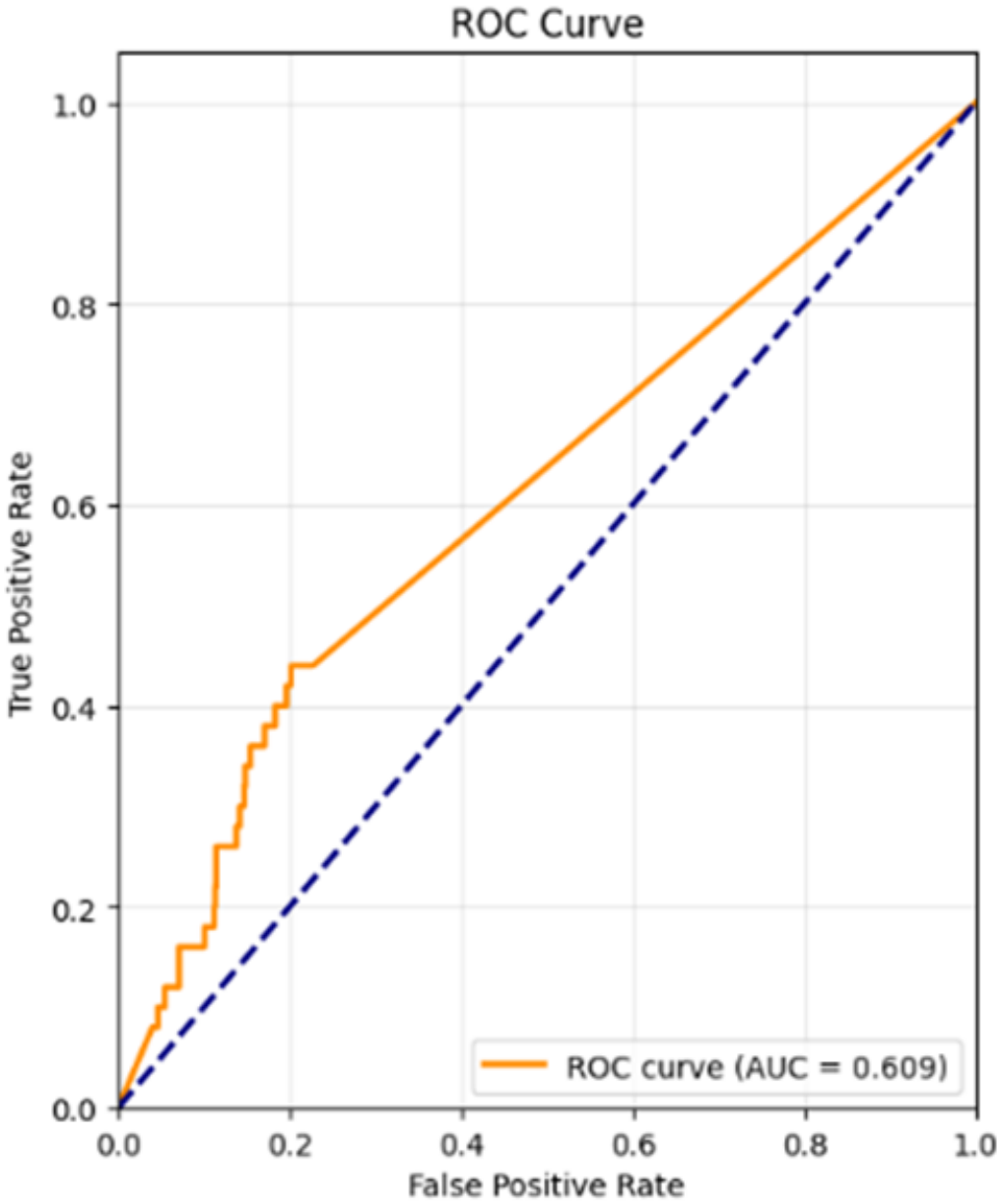
- **Recall** = 0,64 — mô hình phát hiện được 64% số ca đột quỵ thực tế, bỏ sót khoảng 36%.
- **Precision** = 0,12 — khi mô hình dự đoán "bị đột quỵ", chỉ có 12% trường hợp là đúng.
- **F1-score** = 0,21 — phản ánh sự mất cân bằng giữa precision và recall, cho thấy mô hình còn yếu trong nhận diện ca bệnh thật.

Đối với lớp "No Stroke" (lớp âm tính):

- Mô hình hoạt động khá tốt với **Precision = 0,98** và **Recall = 0,77**.
- Giá trị **Negative Predictive Value (NPV)** đạt xấp xỉ **0,98**, nghĩa là khi mô hình dự đoán "không đột quỵ", kết quả gần như chính xác. Đây là một đặc điểm quan trọng trong các hệ thống sàng lọc y tế, giúp hạn chế sai sót trong việc đánh giá bệnh nhân khỏe mạnh.

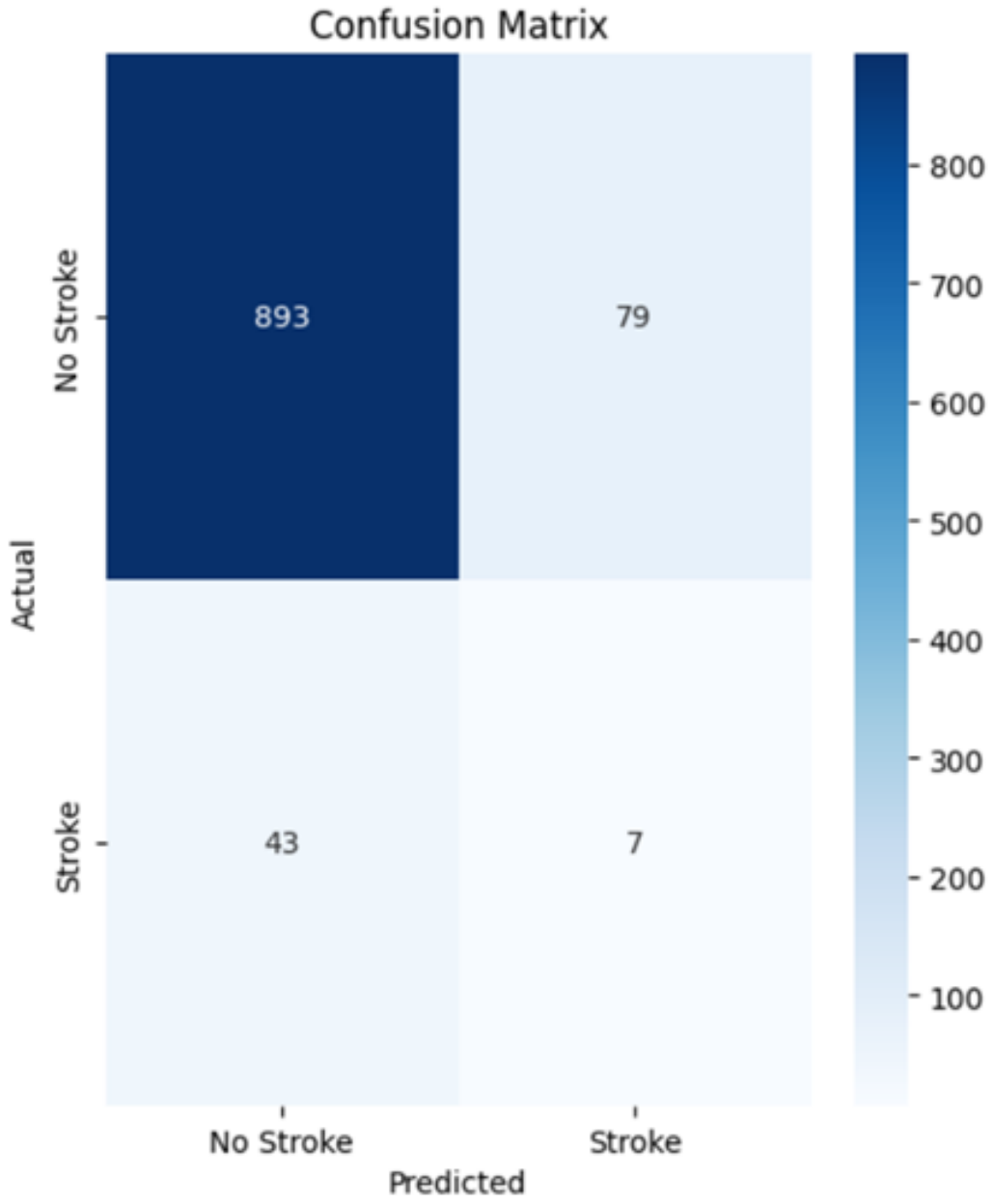
Tổng kết: Mô hình SVM đạt AUC khá (0̃,80) và độ chính xác tổng thể 76,3%, tuy nhiên vẫn gặp khó khăn trong việc nhận diện ca đột quỵ thật (precision thấp). Dù vậy, giá trị **NPV cao** cho thấy mô hình có thể hữu ích trong giai đoạn sàng lọc ban đầu, giúp loại trừ an toàn các ca "không đột quỵ" và hỗ trợ bác sĩ tập trung kiểm tra kỹ hơn các trường hợp nghi ngờ.

6.4 Mô hình KNN



Hình 19: *Đường cong ROC của mô hình KNN.*

Quan sát Hình 19 cho thấy diện tích dưới đường cong ROC (AUC) đạt giá trị **0,609**. Điều này cho thấy khả năng phân loại của mô hình ở mức **yếu**, chỉ nhỉnh hơn một chút so với mô hình ngẫu nhiên (đường chéo AUC = 0,5). Nói cách khác, KNN chưa thể phân biệt rõ ràng giữa hai nhóm "Stroke" và "No Stroke".



Hình 20: *Ma trận nhầm lẫn (Confusion Matrix) của mô hình KNN.*

Dựa trên ma trận nhầm lẫn Hình 20, ta có các giá trị sau:

- **True Negative (TN)** = 866 — Thực tế "No Stroke", dự đoán "No Stroke".
- **False Positive (FP)** = 106 — Thực tế "No Stroke", dự đoán "Stroke".
- **False Negative (FN)** = 41 — Thực tế "Stroke", dự đoán "No Stroke".
- **True Positive (TP)** = 9 — Thực tế "Stroke", dự đoán "Stroke".

Phân tích kết quả:

- Trong tổng số **50 ca đột quỵ thực tế** $(41 + 9)$, mô hình chỉ phát hiện đúng 9 ca và bỏ sót 41 ca. $\Rightarrow$ **Recall** $= \dfrac{9}{9+41} = 0,18$ (18%).
- Trong tổng số **115 ca được dự đoán là "Stroke"** $(106 + 9)$, chỉ có 9 ca là chính xác. $\Rightarrow$ **Precision** $= \dfrac{9}{9+106} \approx 0,078$ (7,8%).
- Tổng thể, mô hình phân loại đúng $(866 + 9) = 875$ mẫu trên tổng $(866 + 106 + 41 + 9) = 1022$ mẫu. $\Rightarrow$ **Accuracy** $= \dfrac{875}{1022} \approx 0,856$ (85,6%).

```
============================================================
DETAILED CLASSIFICATION REPORT
============================================================

Classification Report:
              precision    recall  f1-score   support

   No Stroke       0.95      0.89      0.92       972
      Stroke       0.08      0.18      0.11        50

    accuracy                           0.86      1022
   macro avg       0.52      0.54      0.52      1022
weighted avg       0.91      0.86      0.88      1022
```

Hình 21: *Báo cáo phân loại (Classification Report) của mô hình KNN.*

Đánh giá chi tiết theo từng lớp: Đối với lớp "Stroke" (lớp dương tính):

- **Recall** = 0,18 — mô hình chỉ phát hiện được 18% số ca đột quỵ thực tế, bỏ sót tới 82% ca bệnh.
- **Precision** = 0,08 — khi mô hình dự đoán "bị đột quỵ", chỉ có 8% khả năng dự đoán đó là đúng.
- **F1-score** = 0,11 — rất thấp, cho thấy mô hình không cân bằng được giữa precision và recall.

Đối với lớp "No Stroke" (lớp âm tính):

- Mô hình hoạt động rất tốt với **Precision = 0,95** và **Recall = 0,89**.
- Giá trị **Negative Predictive Value (NPV)** đạt xấp xỉ **0,95**, nghĩa là khi mô hình dự đoán "không đột quỵ", ta có thể gần như chắc chắn rằng kết quả này chính xác.

Tổng kết: Mô hình KNN tuy đạt **Accuracy = 85,6%**, nhưng hiệu quả này chủ yếu đến từ việc mô hình dự đoán chính xác các mẫu "No Stroke" chiếm tỷ lệ lớn. Với dữ liệu mất cân bằng mạnh, mô hình thể hiện **khả năng phát hiện ca đột quỵ kém** (Recall thấp) — điều này khiến KNN không phù hợp cho bài toán sàng lọc y tế, nơi mục tiêu chính là **giảm thiểu bỏ sót bệnh nhân thật (False Negative)**.

6.5 Đánh giá tổng hợp

Tóm tắt kết quả tổng quát

- **Logistic Regression:** Accuracy ở mức trung bình (0.75) nhưng đạt **recall cao nhất** cho lớp *Stroke* (0.80), phù hợp cho bài toán phát hiện sớm ca dương tính. Chỉ số **ROC-AUC = 0.85** cho thấy khả năng phân biệt tốt giữa hai lớp.
- **Random Forest:** Cho **accuracy cao nhất** (0.93), tuy nhiên recall rất thấp (0.14) cho lớp *Stroke*, dẫn đến bỏ sót nhiều ca bệnh. Chỉ số ROC-AUC ở mức trung bình (0.76).
- **SVM:** Đạt accuracy trung bình (0.76), recall khá tốt (0.64), song precision thấp (0.12) khiến số lượng dương tính giả cao. Mặc dù vậy, ROC-AUC = 0.80 cho thấy mô hình vẫn giữ khả năng phân biệt khá ổn.
- **KNN:** Có accuracy cao (0.86) nhưng recall thấp (0.18), tương tự Random Forest, dễ thiên lệch về lớp đa số (*No Stroke*).

Mô hình	Accuracy	Recall	Precision	F1-score	ROC-AUC
Logistic Regression	0.75	**0.80**	0.14	0.24	**0.846**
Random Forest	**0.93**	0.14	0.18	0.16	0.761
SVM	0.76	0.64	0.12	0.21	0.800
KNN	0.86	0.18	0.08	0.11	0.609

Bảng 14: *So sánh hiệu năng các mô hình trên tập kiểm thử*

Phân tích theo từng lớp **1. Lớp "No Stroke" (lớp âm tính)** Tất cả các mô hình đều đạt **precision** và **recall** cao do lớp này chiếm đa số trong dữ liệu.

- **Random Forest** đạt recall cao nhất (0.97), phản ánh khả năng nhận diện tốt các trường hợp không bị đột quỵ.
- **Logistic Regression** có recall thấp hơn (0.75), do ưu tiên cân bằng cho lớp dương tính.

2. Lớp "Stroke" (lớp dương tính)

- **Logistic Regression** vượt trội với recall = 0.80, giúp phát hiện phần lớn các ca bệnh thật.
- **SVM** đứng thứ hai (recall = 0.64), thể hiện năng lực tốt nhờ ranh giới phi tuyến (RBF kernel).
- **Random Forest** và **KNN** có recall rất thấp (0.14 và 0.18), thể hiện sự thiên lệch mạnh về lớp đa số và khả năng bỏ sót cao.

Mô hình	Accuracy	Macro Precision	Macro Recall	Macro F1	ROC-AUC
Logistic Regression	0.75	0.56	**0.77**	0.54	**0.846**
Random Forest	**0.93**	0.57	0.55	0.56	0.761
SVM	0.76	0.55	0.70	0.53	0.800
KNN	0.86	0.52	0.54	0.52	0.609

Bảng 15: *Chỉ số tổng hợp (macro average) của các mô hình*

Nhận xét tổng hợp

- **Random Forest** có accuracy và weighted F1-score cao nhất, nhưng macro recall thấp, cho thấy thiên lệch mạnh về lớp đa số.
- **Logistic Regression** cho kết quả cân bằng nhất, đặc biệt ở recall và ROC-AUC, phù hợp cho dữ liệu mất cân bằng và mục tiêu phát hiện sớm ca bệnh.
- **SVM** duy trì hiệu năng ổn định nhờ học ranh giới phi tuyến, là mô hình thay thế khả thi nếu được tinh chỉnh thêm threshold.
- **KNN** có độ chính xác tổng thể cao, nhưng yếu trong nhận diện ca "Stroke", chỉ nên dùng khi kết hợp thêm kỹ thuật tái lấy mẫu (SMOTE) hoặc trọng số theo khoảng cách.

Kết luận

- Nếu **ưu tiên độ chính xác tổng thể (accuracy)**, **Random Forest** là lựa chọn tốt nhất.
- Nếu **ưu tiên phát hiện bệnh nhân đột quỵ thật (recall lớp Stroke)**, **Logistic Regression** là mô hình phù hợp nhất với dữ liệu hiện tại.
- Các mô hình **SVM** và **KNN** có thể cải thiện hiệu năng nếu áp dụng kỹ thuật xử lý mất cân bằng (SMOTE, class weighting, threshold tuning).

7 Kết luận và hướng phát triển

7.1 So sánh và thảo luận

Kết quả thực nghiệm từ hai mô hình cho thấy sự khác biệt đáng kể cả về hiệu năng định lượng lẫn ý nghĩa ứng dụng.

Trước hết, xét theo báo cáo từ terminal, Logistic Regression đạt ROC-AUC 0,846, trong khi Random Forest chỉ đạt ROC-AUC 0,761. Chỉ số AUC phản ánh khả năng phân biệt giữa hai lớp, và sự chênh lệch này chứng minh Logistic Regression có độ tin cậy cao hơn trong việc ước lượng xác suất bệnh nhân mắc đột quỵ.

Tiếp theo, nếu xem xét các thước đo phân loại cụ thể, ma trận nhầm lẫn và báo cáo cho thấy Logistic Regression tuy có accuracy thấp hơn (0,75), nhưng lại đạt recall cho lớp dương tính là 0,54 – nghĩa là phát hiện được hơn một nửa số ca đột quỵ. Ngược lại, Random Forest đạt accuracy cao (0,93) nhưng recall cho lớp dương tính chỉ là 0,14, tức là bỏ sót tới 86% trường hợp thực sự cần được phát hiện.

Điều này cho thấy accuracy không phải lúc nào cũng là thước đo phù hợp, đặc biệt trong bối cảnh dữ liệu mất cân bằng lớp như bộ dữ liệu đột quỵ (tỷ lệ bệnh nhân có đột quỵ chỉ chiếm chưa tới 5%). Mô hình Random Forest chủ yếu dự đoán theo lớp chiếm đa số (không đột quỵ), từ đó dẫn đến độ chính xác tổng thể cao nhưng khả năng phát hiện bệnh nhân gần như bị vô hiệu.

Trong khi đó, Logistic Regression, nhờ bản chất tuyến tính và khả năng mô hình hóa trực tiếp xác suất, đã thể hiện sự cân bằng tốt hơn giữa precision (0,26) và recall (0,54) đối với lớp dương tính. Mặc dù precision chưa cao, nhưng từ quan điểm y tế dự phòng, việc chấp nhận số ca dương tính giả để đổi lấy tỷ lệ phát hiện sớm cao hơn thường được coi là chấp nhận được. Một bệnh nhân bị dự đoán nhầm là có nguy cơ đột quỵ có thể được kiểm tra lại bằng các phương pháp lâm sàng khác, trong khi bỏ sót bệnh nhân thật sự có nguy cơ lại tiềm ẩn rủi ro rất lớn.

Về mặt trực quan, các hình ảnh ROC và ma trận nhầm lẫn càng củng cố nhận định này: Logistic Regression tạo ra đường cong ROC cao hơn, cùng với ma trận nhầm lẫn cho thấy khả năng nhận diện ca bệnh tốt hơn. Trong khi đó, Random Forest dù có ưu điểm về khả năng dự đoán đúng nhóm không bệnh, nhưng điều này ít mang giá trị thực tiễn trong kịch bản ứng dụng y tế, nơi mà ưu tiên chính là giảm thiểu bỏ sót bệnh nhân nguy cơ cao.

Tóm lại, so sánh hai mô hình cho thấy:

- Logistic Regression: ROC-AUC cao hơn, recall tốt hơn, thích hợp cho nhiệm vụ phát hiện sớm, dù accuracy tổng thể chưa cao.
- Random Forest: Accuracy cao do dự đoán nghiêng về lớp không bệnh, nhưng recall cho lớp bệnh nhân quá thấp, do đó không phù hợp cho mục tiêu nghiên cứu.

Điều này nhấn mạnh tầm quan trọng của việc lựa chọn thước đo phù hợp trong đánh giá mô hình, cũng như nhu cầu xử lý dữ liệu mất cân bằng (ví dụ: sử dụng SMOTE, cost-sensitive learning hoặc điều chỉnh threshold) để cải thiện hiệu năng đối với lớp dương tính — vốn mang ý nghĩa quan trọng nhất trong ứng dụng thực tế.

8 Nguồn và tài liệu tham khảo

1. **Kaggle Stroke Prediction Dataset:** https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
2. **Stroke, Cerebrovascular Accident (WHO):** https://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/
3. **World Stroke Organization: Global Stroke Fact Sheet 2025:** https://pmc.ncbi.nlm.nih.gov/articles/PMC11786524/
4. **Việt Nam thuộc nhóm nước có tỷ lệ đột quỵ cao nhất Đông Nam Á:** https://moh.gov.vn/su-kien-y-te-noi-bat/-/asset_publisher/8EeXRtRENhb6/content/viet-nam-thuoc-nhom-nuoc-co-ty-le-ot-quy-cao-nhat-ong-nam-a
5. Jiawei Han, Micheline Kamber, and Jian Pei, *Data Mining: Concepts and Techniques*, 3rd Edition, Morgan Kaufmann Publishers, 2012.
6. David Hand, Heikki Mannila, Padhraic Smyth, *Principles of Data Mining*, MIT Press, 2001.
7. Chetan Sharma, Shamneesh Sharma, Mukesh Kumar, Ankur Sodhi, *Early Stroke Prediction Using Machine Learning.*
8. Anastasija Tashkova, Stefan Eftimov, Bojan Ristov, Slobodan Kalajdziski, *Comparative Analysis of Stroke Prediction Models Using Machine Learning.*
9. Michael Wiryaseputra, *Stroke Prediction Using Machine Learning Classification Algorithm.*