HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INTERNATION EDUCATION

REPORT

MACHINE LEARNING

Lecture: PhD. Vu Quang Huy

Name:	**ID:**
Nguyễn Châu Tấn Cường	**23146007**
Lê Bình Nguyên	**23146019**

Ho Chi Minh, 17 May, 2025

Table of Contents

I. Overview

1. Topic

In recent years, cardiovascular diseases have become the leading cause of death worldwide, accounting for approximately 17.9 million deaths annually, according to statistics from the World Health Organization (WHO). Timely detection and diagnosis of heart diseases play a crucial role in reducing mortality rates and increasing patient survival. However, in clinical practice, the diagnosis process often faces significant challenges due to the complexity, inconsistency, and lack of labeled medical data.

In the era of digital transformation and the rapid advancement of artificial intelligence, machine learning methods are opening new directions for modern medicine. Particularly, unsupervised learning a subfield of machine learning that handles unlabeled data enables the extraction of hidden knowledge and patterns from raw datasets. This is especially beneficial in the medical field, where collecting and labeling data is time-consuming and costly.

The topic "***Heart Disease Diagnosis Using Unsupervised Machine Learning***" aims to study and apply unsupervised learning algorithms to cluster patients based on medical information, thereby supporting the identification of high-risk groups for heart disease. The approach of not relying on labeled data reflects more realistically the current situation of many healthcare data systems. As a result, the topic has high practical relevance and contributes to the ongoing trend of data-driven, label-free disease diagnosis.

2. Reason for Choosing the Topic

The choice of this topic is based on several key reasons:

- *Urgency and societal importance*: Cardiovascular disease remains one of the most critical health issues, particularly in developing countries. Applying technology to early disease detection can significantly enhance life quality and reduce the burden on healthcare systems.
- *Addressing the lack of labeled data in healthcare*: In real-world settings, medical datasets are often incomplete or lack clearly defined labels due to the absence of formal diagnoses or data entry inconsistencies. Studying unsupervised models provides a powerful way to uncover meaningful patterns and groupings without needing labeled input.
- *Scalability and broader application potential*: Unsupervised learning techniques are not limited to healthcare—they are widely applicable in domains like finance, agriculture, and energy. Understanding the performance and limitations of these algorithms brings long-term value.
- *Availability of standardized real-world datasets*: This project will utilize well-known datasets for heart disease from the UCI Machine Learning Repository, including Cleveland, Hungary, Switzerland, and Long Beach V. These datasets are trusted, diverse, and representative of various patient demographics and health profiles.

3. Expected Content and Implementation Plan

To achieve the research objectives, the project will include the following major components:

- *Theoretical foundation and literature review*: Study basic knowledge of cardiovascular diseases, their physiological indicators, and common medical features

related to heart disease (such as blood pressure, cholesterol, heart rate, age, gender, etc.).

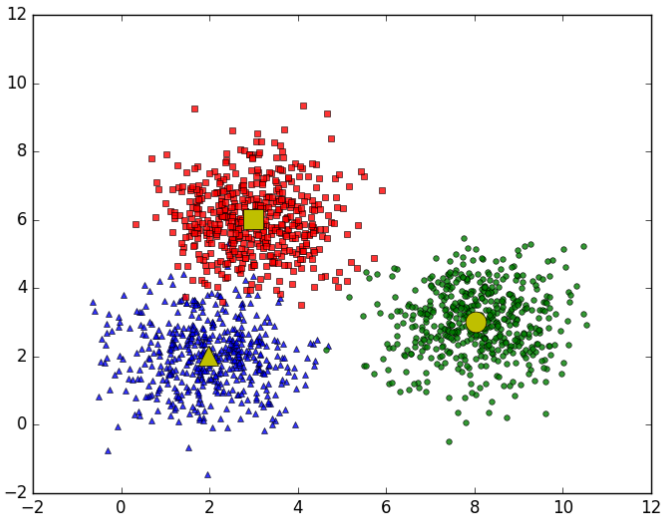
- *Data analysis and preprocessing*:
  - Merge and harmonize the Cleveland, Hungary, Switzerland, and Long Beach V datasets.
  - Perform preprocessing tasks including handling missing values, encoding categorical variables, and feature normalization.
  - Conduct descriptive statistical analysis to understand data distributions.
- *Applying unsupervised learning algorithms*:
  - Implement clustering methods such as *K-Means*, *DBSCAN*, *Gaussian Mixture Models (GMM)* and *Hierarchical Clustering*, ...
  - Explore algorithm variants and improvements such as *K-Means++*, *MiniBatch K-Means*, *Spectral Clustering*, and *Mean Shift* to improve clustering performance.
- *Model evaluation*:
  - Evaluate clustering performance using metrics such as *Silhouette Score*, *Davies-Bouldin Index* and *Calinski-Harabasz Index*.
  - Analyze clustering results to determine whether clusters align with known risk factors or clinical patterns for heart disease.
- *Comparison and result synthesis*:
  - Compare the performance of different algorithms and their variations on the same datasets.
  - Identify strengths and limitations of each method in the context of heart disease clustering.
- *Conclusion and future recommendations*:
  - Recommend the most suitable unsupervised model for heart disease data analysis.
  - Suggest potential applications in clinical decision support systems and directions for further research, such as hybrid or semi-supervised models.

II. Theoretical

1. K-Means Clustering

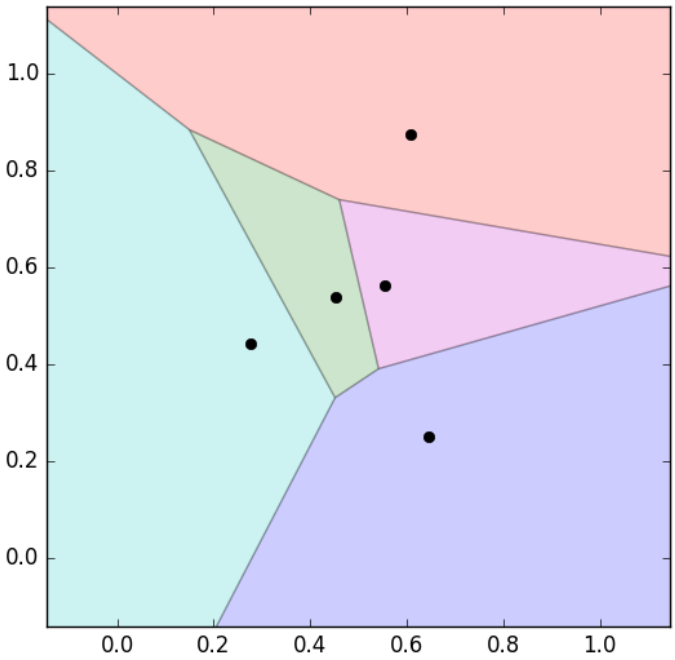
1.1. Method

K-Means is a partition-based clustering algorithm. It divides data into K non-overlapping clusters where each observation belongs to the cluster with the nearest mean (centroid).
In the *K-means* clustering algorithm, we do not know the labels of each data point. The goal is to partition the data into different clusters such that the data points within the same cluster share similar characteristics.
The simplest idea of a cluster is a group of points that are close to each other in some space (this space can be high-dimensional in cases where each data point contains a large amount of information). The image below is an example of 3 data clusters (from now on, I will simply refer to them as clusters).



Suppose each cluster has a representative point (center) marked in yellow. The surrounding points near each center belong to the same group as that center. A simple way to determine this is: for any given point, we check which center it is closest to, and assign it to the same group as that center.

At this point, we arrive at an interesting problem: On a large square sea area, there are three islands shaped like a square, triangle, and circle, all marked in yellow as shown in the image above. A point on the sea is said to belong to the territorial waters of an island if it is closer to that island than to the other two. Determine the boundary of each island's territorial waters. The image below illustrates how territorial waters are divided when there are 5 different islands represented by black circles:



We can see that the boundaries between the territorial waters are straight lines (more precisely, they are perpendicular bisectors of pairs of nearby points). Therefore, the territorial waters of an island form a polygon.
This method of partitioning is known in mathematics as a *Voronoi Diagram*.
In three-dimensional space for example, with planets the "territorial space" of each planet would be a polyhedron. In higher-dimensional spaces, we get what I refer to as hyperpolygons (more formally, polytopes or Voronoi cells in higher dimensions).
1.2. Algorithm

$$J = \sum_{i=1}^{m} \sum_{k=1}^{m} w_{ik} \|x_i - c_k\|^2$$

$$w_{ik} = 0 \; if \; the \; data \; point \; does \; not \; belong \; to \; the \; cluster$$
$$w_{ik} = 1 \; if \; the \; data \; point \; belongs \; to \; the \; cluster$$

- x_i là điểm dữ liệu.
- c_k là tâm cụm thứ k.
- W_{ik} là chỉ số phân bổ của điểm x_i vào cụm k.

Advantages	Disadvantages
• Easy to understand and implement. • Fast computation.	• The number of clusters K. • K must be predefined. • Sensitive to initial centroid placement. • Struggles with clusters that have irregular shapes or varying sizes.

1.3. Tuning Parameters
- K: The number of clusters.
- The algorithm requires a random initialization of centroids.

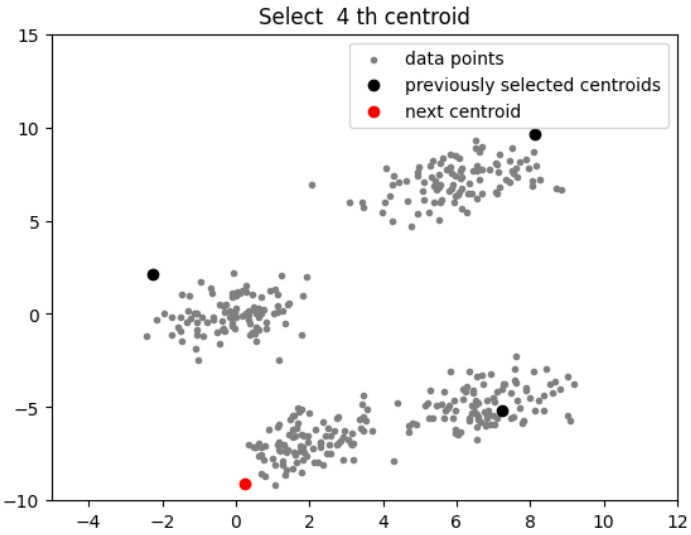
Applications : Customer segmentation, market analysis, pattern recognition in image data.

2. K-Means++

2.1. Method
KMeans++ is an improved version of the *K-Means* algorithm that automatically chooses better starting points instead of selecting them randomly. The key idea behind *KMeans++* is that it chooses the initial cluster centers in a smart way to ensure they are spread out which helps the algorithm converge faster and gives better clustering results.
2.2. Algorithm
How K-mean++ Algorithm Works?
The *KMeans++* algorithm works in two steps:
Step 01 Initialization:
- Choose the first cluster center randomly from the data points.
- For each remaining cluster center select the next center based on the probability that is proportional to the square of the distance between the data point and the closest selected center.

Step 02 Clustering:
- After selecting the initial centers *KMeans++* performs clustering the same way as *K-Means*:

- Assign each data point to the nearest cluster center.
- Recalculate cluster centers by finding the average of all points in each cluster.
- Repeat the steps until the cluster centers do not change or a fixed number of iterations is reached.



Advantages	Disadvantages
• *K-Means++* intelligently selects initial centroids, which helps avoid falling into local extrema and improves clustering results. • Due to better initialization, *K-Means++* often gives more accurate results than regular *K-Means*. • Although the initialization is more complex than regular *K-Means*, the total execution time does not increase much and the convergence time can be reduced. • Retains the simplicity of *K-Means*, easy to integrate into real-world applications and popular machine learning libraries.	• The number of clusters (*k*) needs to be determined in advance, which may be unclear or require multiple trials. • *K-Means++* still inherits the drawbacks of the original *K-Means*: it does not perform well when clusters are non-spherical in shape or have very different sizes. • Like *K-Means*, *K-Means++* is susceptible to noisy data points or outliers, as they can pull the centroids away from the logical position.

2.3. Tuning Parameters

- *K*: The number of clusters.

Applications : Similar to *K-Means*, but with improved clustering quality.

3. K-Medoids

3.1. Method

K-Medoids (also called Partitioning Around Medoid) algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as a point in the cluster, whose dissimilarities with all the other points in the cluster are minimum. The dissimilarity of the medoid(C_i) and object(P_i) is calculated by using $E = |P_i – C_i|$.

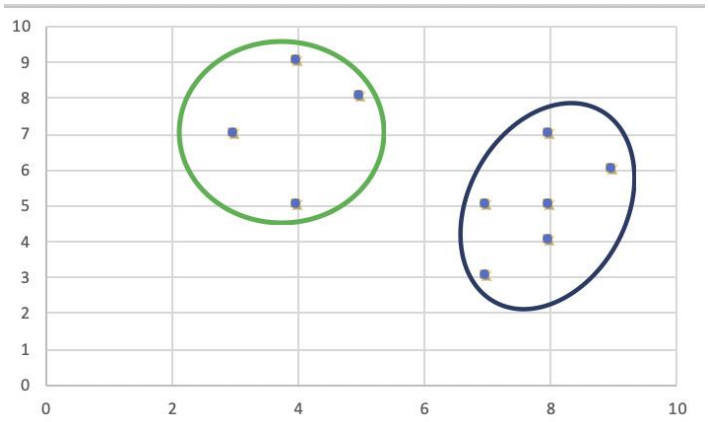
3.2. Algorithm

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, m_i)$$

Where:

- k = number of clusters
- x = data points
- C_i = set of points in the i^{th} cluster
- m_i = medoid of cluster i
- $d(x, m_i)$ = dissimilarity (distance) between a point and its medoid (e.g., Euclidean or Manhattan distance).

1. Initialize: select k random points out of the n data points as the medoids.
2. Associate each data point to the closest medoid by using any common distance metric methods.
3. While the cost reduces: For each medoid m, for each data o point which is not a medoid:
    - Swap m and o, associate each data point to the closest medoid, and recompute the cost.
    - If the total cost is more than that in the previous step, undo the swap.



Advantages	Disadvantages
• It is simple to understand and easy to implement. • *K-Medoid* Algorithm is fast and converges in a fixed number of steps. • PAM is less sensitive to outliers than other partitioning algorithms.	• The main disadvantage of *K-Medoid* algorithms is that it is not suitable for clustering non-spherical (arbitrarily shaped) groups of objects. This is because it relies on minimizing the distances between the non-medoid objects and the medoid (the cluster center) – briefly, it uses compactness as clustering criteria instead of connectivity. • It may obtain different results for different runs on the same dataset because the first k medoids are chosen randomly.

3.3. Tuning Parameters

- *K*: The number of clusters.

Applications : Customer segmentation in commercial sectors, biodata analysis.

4. Fuzzy C-Means (FCM)

4.1. Method

Fuzzy Clustering is a type of clustering algorithm in machine learning that allows a data point to belong to more than one cluster with different degrees of membership. Unlike traditional clustering (like *K-Means*), where each data point belongs to only one cluster, fuzzy clustering allows a data point to belong to multiple clusters with different membership levels.

4.2. Algorithm

How Does Fuzzy Clustering Work?

Fuzzy clustering follows an iterative optimization process where data points are assigned membership values instead of hard cluster labels. Here's a step-by-step breakdown of how it works:

Step 01: Initialize Membership Values Randomly: Each data point is assigned a membership degree for all clusters. These values indicate the probability of the data point belonging to each cluster. Unlike hard clustering (where a point strictly belongs to one cluster), *fuzzy clustering* allows partial membership.

Step 02: Compute Cluster Centroids: The centroids of the clusters are calculated based on the weighted sum of all data points, where weights are determined by membership values. This ensures that points with higher membership contribute more to the centroid.

The formula for finding out the centroid (V) is:

$$\mathbf{v}_i = \frac{\sum_{j=1}^{n} (u_{ij})^m \mathbf{x}_j}{\sum_{j=1}^{n} (u_{ij})^m}, \ \forall i$$

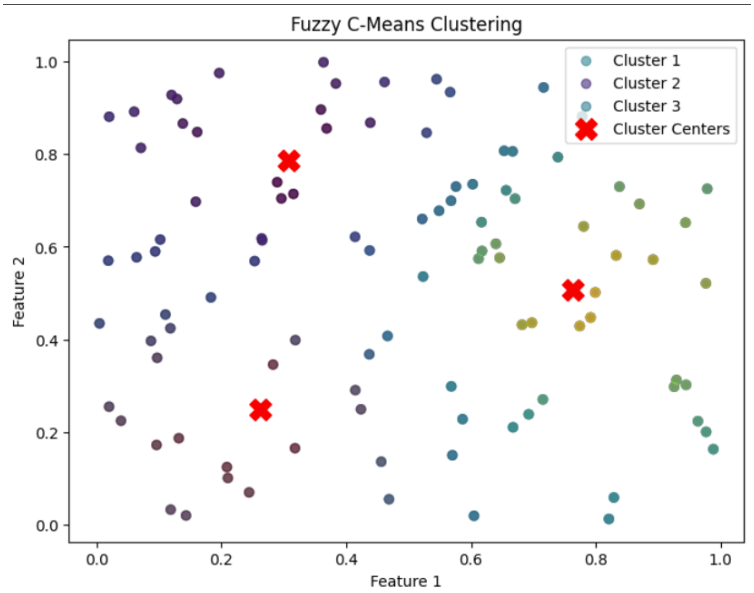
Where, μ is fuzzy membership value of the data point, m is the fuzziness parameter (generally taken as 2), and x_k is the data point.

Step 03: Calculate Distance Between Data Points and Centroids: The Euclidean distance (or another distance metric) between each data point and the centroids is computed. This helps in updating the membership values.

Step 04: Update Membership Values: The membership values are recalculated based on how close a point is to a centroid relative to the other centroids. The farther a point is from a centroid, the lower its membership value for that cluster.

$$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{||\mathbf{x}_j - \mathbf{v}_i||}{||\mathbf{x}_j - \mathbf{v}_k||} \right)^{\frac{2}{m-1}} \right]^{-1}, \ \forall i, j$$

Step 05: Repeat Until Convergence: Steps 2–4 are repeated until the membership values stabilize, meaning there are no significant changes from one iteration to the next. This indicates that the clustering has reached an optimal state.



Advantages	Disadvantages
• *Fuzzy clustering* allows for overlapping clusters, which can be useful when the data has a complex structure or when there are ambiguous or overlapping class boundaries. • *Fuzzy clustering* can be more robust to outliers and noise in the data, as it allows for a more gradual transition from one cluster to another. • *Fuzzy clustering* provides a more nuanced understanding of the structure of the data, as it allows for a more detailed representation of the relationships between data points and clusters.	• *Fuzzy clustering* algorithms can be computationally more expensive than traditional clustering algorithms, as they require optimization over multiple membership degrees. • Choosing the right number of clusters and membership functions can be challenging, and may require expert knowledge or trial and error.

4.3. Tuning Parameters

- *K*: The number of clusters.
- m: The fuzziness parameter.

Applications : Medical image analysis, signal processing, clustering uncertain data.

5. Gaussian Mixture Models (GMM)

5.1. Method

Gaussian Mixture Models (GMM) use soft clustering, where data points can belong to multiple clusters with a certain probability. This provides a more flexible and nuanced way to handle clusters, especially when points are close to multiple centroids.

5.2. Algorithm

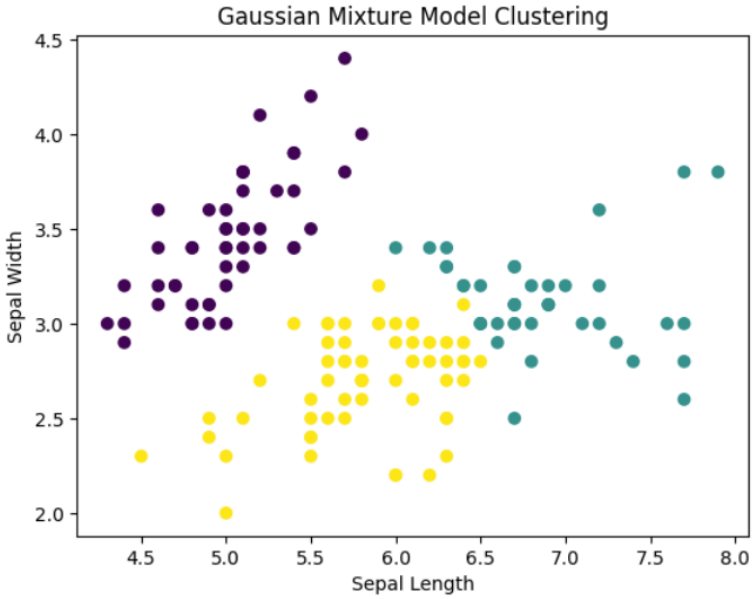
How GMM Works?

Here's a simple breakdown of the process:

1. Initialization: Start with initial guesses for the means, covariances, and mixing coefficients of each Gaussian distribution.
2. E-step: For each data point, calculate the probability of it belonging to each Gaussian distribution (cluster).
3. M-step: Update the parameters (means, covariances, mixing coefficients) using the probabilities calculated in the E-step.
4. Repeat: Continue alternating between the E-step and M-step until the log-likelihood of the data (a measure of how well the model fits the data) converges.

$$f(\mathbf{y}_i; \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j \phi^{(p)}(\mathbf{y}_i; \mu_j, \Sigma_j),$$

The E-step computes the probabilities that each data point belongs to each Gaussian, while the M-step updates the parameters μ_j, Σ_j , and y_j based on these probabilities.



Advantages	Disadvantages
• Unlike *K-Means*, which assumes spherical clusters, GMM can model clusters with arbitrary shapes. • *GMM* assigns a probability for each data point to belong to each cluster, while K-Means assigns each point to exactly one cluster. • *GMM* performs well when clusters overlap or have varying densities. Since it uses probability distributions, it can assign a point to multiple clusters with different probabilities.	• *GMM* tends to be computationally expensive, particularly with large datasets, as it requires iterative processes like the Expectation-Maximization (EM) algorithm to estimate the parameters. • Like other clustering methods, *GMM* requires you to specify the number of clusters beforehand. However, methods like the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) can help in selecting the optimal number of clusters based on the data.

5.3. Tuning Parameters

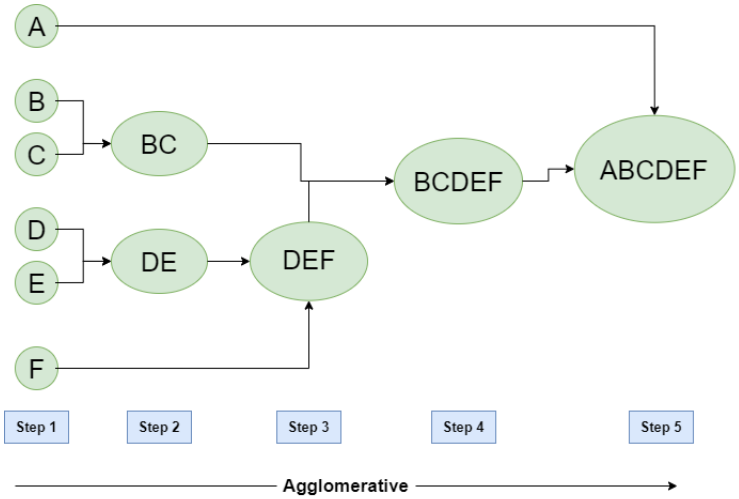
- K: The number of clusters.
- μ_j, Σ_j , and y_j: Parameters of the Gaussian distribution.

Applications : Pattern recognition, signal analysis, object recognition in images.

6. Agglomerative Hierarchical Clustering (AHC)

6.1. Method

Unlike flat clustering *hierarchical clustering* provides a structured way to group data. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.



6.2. Workflow

1. Start with individual points: Each data point is its own cluster. For example if you have 5 data points you start with 5 clusters each containing just one data point.
2. Calculate distances between clusters: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. Merge the closest clusters: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. Update distance matrix: After merging you now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. Repeat steps 3 and 4: Keep merging the closest clusters and updating the distance matrix until you have only one cluster left.
6. Create a dendrogram: As the process continues you can visualize the merging of clusters using a tree-like diagram called a dendrogram. It shows the hierarchy of how clusters are merged.

Advantages	Disadvantages
• Does not require the number of clusters to be predefined. • Suitable for data with hierarchical structure.	• Computationally expensive for large datasets. • Sensitive to outliers.

6.3. Tuning Parameters

- Linkage method: Single linkage, average linkage, complete linkage, etc.

Applications : Organizational structure analysis, biological clustering, data relationships.

III. Data

1. Description

The dataset is obtained from the UCI Machine Learning Repository and includes patient data from four hospitals:

1. Cleveland Clinic Foundation (Cleveland)
2. Hungarian Institute of Cardiology (Hungary)
3. University Hospital, Zurich, Switzerland
4. VA Medical Center, Long Beach, California (Long Beach V)

In this project:

- The Cleveland dataset is used only for testing
- The remaining datasets (Hungary, Switzerland, Long Beach) are used for training in an unsupervised learning context.

2. Input and Output Feature Description

Although the original dataset includes 76 attributes, most research focuses on the following 13 or 14 key features:

No	Feature	Description	Type
1	*AGE*	Age of the patient	Continuous
2	*SEX*	Sex (1 = male, 0 = female)	Binary
3	*CP*	Chest pain type (0–3)	Categorical
4	*TRESTBPS*	Resting blood pressure (mm Hg)	Continuous
5	*CHOL*	Serum cholesterol (mg/dl)	Continuous
6	*FBS*	Fasting blood sugar > 120 mg/dl (1 = true, 0 = false)	Binary
7	*RESTECG*	Resting electrocardiographic results	Categorical
8	*THALACH*	Maximum heart rate achieved	Continuous
9	*EXANG*	Exercise-induced angina (1 = yes, 0 = no)	Binary
10	*OLDPEAK*	ST depression induced by exercise	Continuous
11	*SLOPE*	Slope of peak exercise ST segment	Categorical
12	*CA*	Number of major vessels colored by fluoroscopy (0–3)	Discrete
13	*THAL*	Type of thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect)	Categorical
14	*TARGET*	**Output**: Presence of heart disease (0 = no, 1 = yes)	Binary

3. Structure and Properties

- *Size*: The combined dataset contains over 900 records; Cleveland subset has 303.
- *Data Types*: Mostly numeric (both continuous and categorical).
- *Missing Values*: Present in attributes such as ca and thal, sometimes denoted by NaN or None.
- *Class Imbalance*: The distribution of patients with and without heart disease may be imbalanced depending on the subset.

- *Correlations*: Features such as thalach, oldpeak, and cp often show strong correlation with the target.

4. Data Preprocessing

Splitting the Dataset:

- Cleveland dataset is used solely for testing and visualization.
- The remaining records (Hungary, Switzerland, Long Beach) are used for training.

Handling Missing Values:

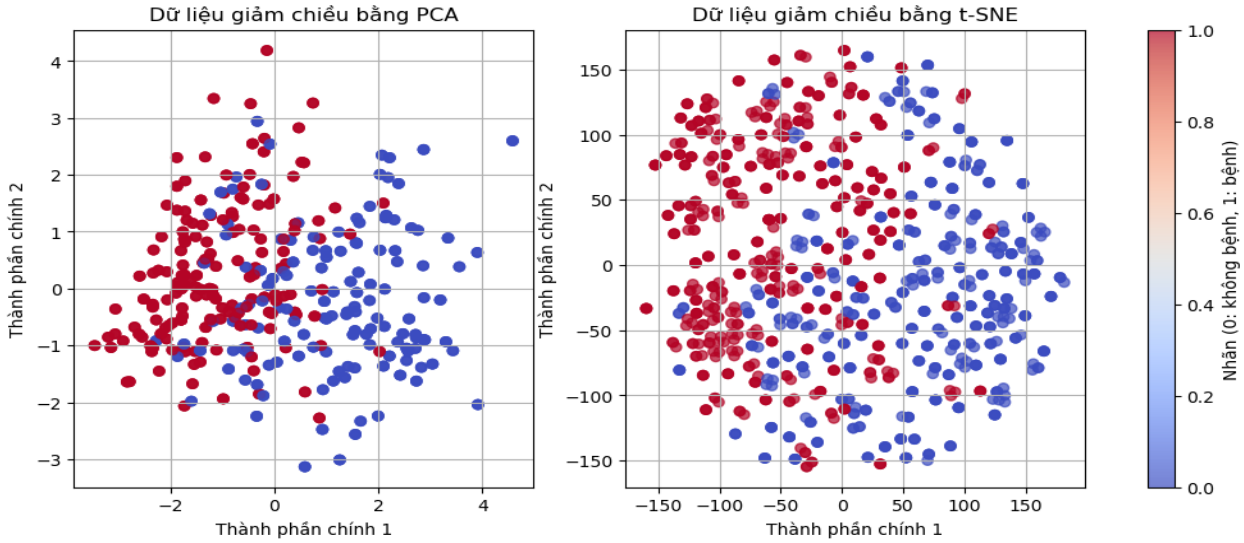
- All missing or NaN values are replaced with 0.

Dimensionality Reduction & Visualization:

- PCA (Principal Component Analysis) is applied to reduce the dimensionality of the data while retaining as much variance as possible.
- t-SNE (t-distributed Stochastic Neighbor Embedding) is used to visualize clusters and observe potential groupings in the data.

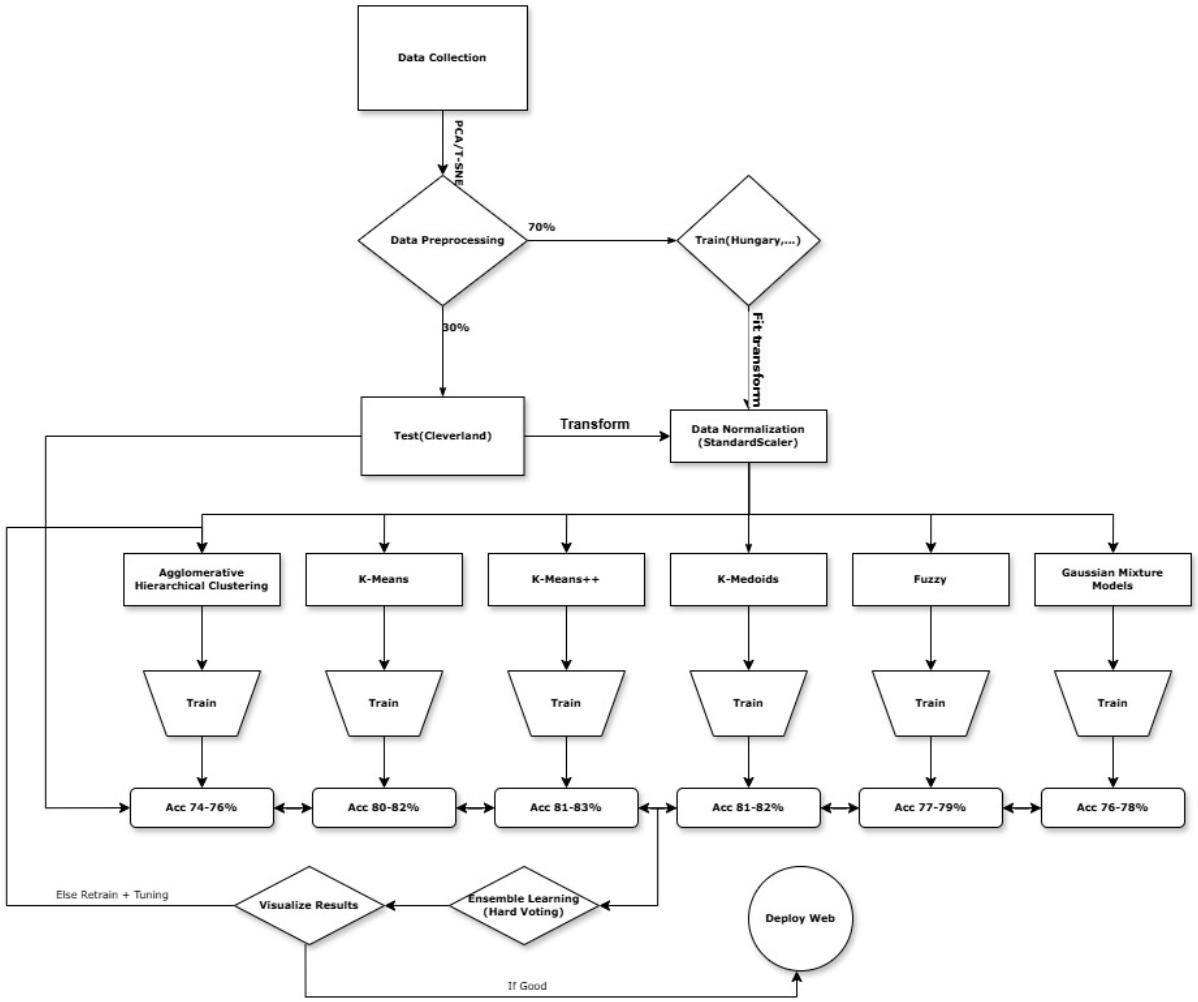
If any outliers or anomalies are noticed during visualization, the data is reviewed and corrected accordingly.

Data after Preprocessing:



IV. Algorithm Flowchart

1. Pipeline



2. Explanation

First, I collected and selected data from the UCI Heart Disease dataset. After reading the data, I applied dimensionality reduction techniques such as *PCA* or *T-SNE* to visualize the dataset and check for any noisy or unnecessary features, which I then removed to improve data quality.

Once preprocessing was complete, I split the dataset into 70% for training and 30% for evaluation. I then standardized the data using `*StandardScaler*` to bring all features to the same scale, which helps the clustering algorithms perform more effectively.

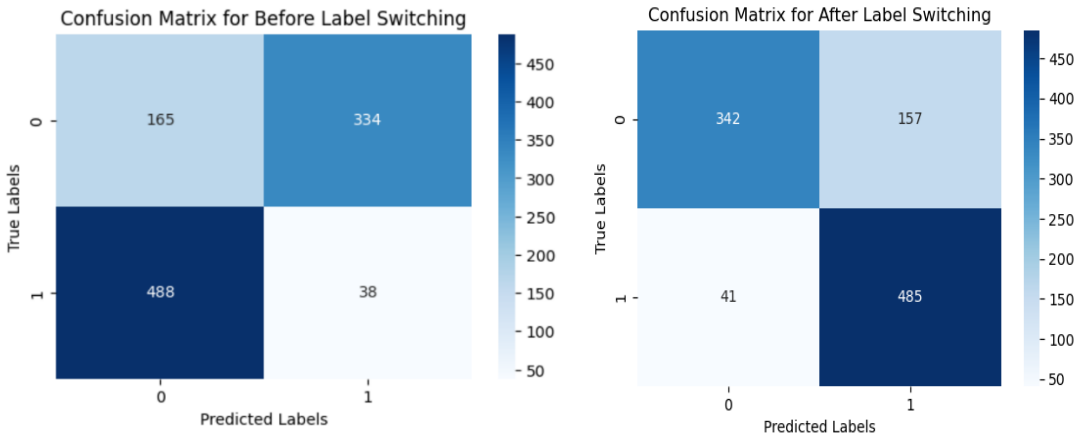
Next, I trained the preprocessed data using various clustering algorithms, including *KMeans*, *KMeans++*, *KMedoids*, *Gaussian Mixture Model (GMM)*, *Agglomerative Hierarchical Clustering (AHC)*, and *Fuzzy C-Means*.

Each algorithm assigned cluster labels to the data samples. I evaluated the performance of each model using metrics such as accuracy or clustering evaluation scores like Adjusted Rand Index (ARI). Models with accuracy below 70% were fine-tuned and retrained, while those with accuracy above 70% were retained for the next step.

After selecting the best-performing models, I used a Hard Voting approach to combine their predictions. Specifically, I collected the cluster labels predicted by each model and used the

majority vote to determine the final label for each sample. I then tested this combined model on the 30% test set to evaluate its overall accuracy and stability. If the ensemble model achieved an accuracy higher than 70%, I considered it reliable and deployed it to a web platform for public testing. If not, I went back to fine-tune and retrain the models until satisfactory performance was achieved.

Note: It is important to note that when using unsupervised learning algorithms, cluster labels are assigned arbitrarily. Therefore, after training, it is necessary to verify whether the assigned labels correspond correctly to the actual classes. If not, label permutation or alignment may be required in order to accurately evaluate the model's performance. Here is an example :



V. Model Training Code

1. Common Step

Step 1 Import shared library :

```
from sklearn.preprocessing import StandardScaler  # Data normalization
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
silhouette_score, davies_bouldin_score  # Clustering evaluation
import pandas as pd  # Table data processing
import numpy as np  # Arithmetic and array processing
import matplotlib.pyplot as plt  # Plotting charts
import seaborn as sns  # Advanced plotting
from scipy.stats import mode
```

Step 2 Import data :

```
# Read data from CSV file
df_test = pd.read_csv(".\\data\\heart.csv")  # Change the path if error occurs
# Separate features (X) and labels (y)
X_CSV = df_test.drop(columns=["target"])  # Input features (remove 'target' column)
y_CSV_test = df_test["target"].values  # True labels: 0 (no disease), 1 (disease)
```

Step 3 Standardize data :

```
# Standardize data to ensure features have the same scale, avoiding bias when calculating distances
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_CSV)  # Fit and transform the input data
```

2. K-Means Clustering

Step 1 Import K-Means Library :

```
from sklearn.cluster import KMeans  # KMeans clustering algorithm
```

Step 2 Train / Predict :

```
# Initialize KMeans model with 2 clusters (assuming division into 2 groups: disease and no disease)
kmeans = KMeans(n_clusters=2, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)  # Predict cluster for each sample
```

Step 3 Label Switching :

```
# Note: Since KMeans is unsupervised, cluster labels (0/1) may not match true labels.
# Therefore, calculate accuracy for both original and flipped labels.
acc_original = accuracy_score(y_CSV_test, y_kmeans)
acc_flipped = accuracy_score(y_CSV_test, 1 - y_kmeans)
# If flipping labels gives higher accuracy, perform the flip
if acc_flipped > acc_original:
    y_kmeans = 1 - y_kmeans
```

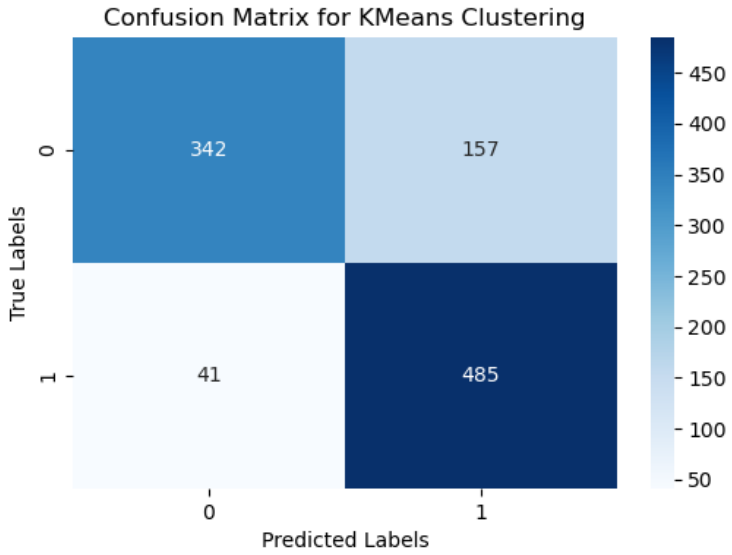
Step 4 Print Score :

```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("KMeans Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_kmeans))
# Print classification report including precision, recall, and f1-score for each label
print("\nKMeans Classification Report:")
print(classification_report(y_CSV_test, y_kmeans))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_kmeans)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_kmeans)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	*Precision*	*Recall*	*F1-score*	*Support*
0	*0.89*	*0.69*	*0.78*	*499*
1	*0.76*	*0.92*	*0.83*	*526*
Accuracy			*0.81*	*1025*
Macro avg	*0.82*	*0.80*	*0.80*	*1025*
Weighted avg	*0.82*	*0.81*	*0.80*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_kmeans), annot=True, fmt='d',
cmap='Blues')
plt.title("Confusion Matrix for KMeans Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
plt.show()
```



3. K-Means++

Step 1 Import K-Means++ Library :

```
from sklearn.cluster import KMeans  # KMeans clustering algorithm
```

Step 2 Train / Predict :

```
# - n_clusters=2: aim to divide the data into 2 clusters (corresponding to 0/1)
# - init='k-means++': improves centroid initialization to avoid poor results
# - random_state=42: ensures reproducibility of results
kmeans_plus = KMeans(n_clusters=2, init='k-means++', random_state=42)
```

```
y_kmeans_plus = kmeans_plus.fit_predict(X_scaled)  # Cluster labels returned as an array of 0s and 1s
```

Step 3 Label Switching :

```
# Note: Since KMeans++ is unsupervised, cluster labels (0/1) may not match true labels.
# Therefore, calculate accuracy for both original and flipped labels.
acc_original = accuracy_score(y_CSV_test, y_kmeans_plus)
acc_flipped = accuracy_score(y_CSV_test, 1 - y_kmeans_plus)
# If flipping labels gives higher accuracy, perform the flip
if acc_flipped > acc_original:
    y_kmeans_plus = 1 - y_kmeans_plus
```

Step 4 Print Score :

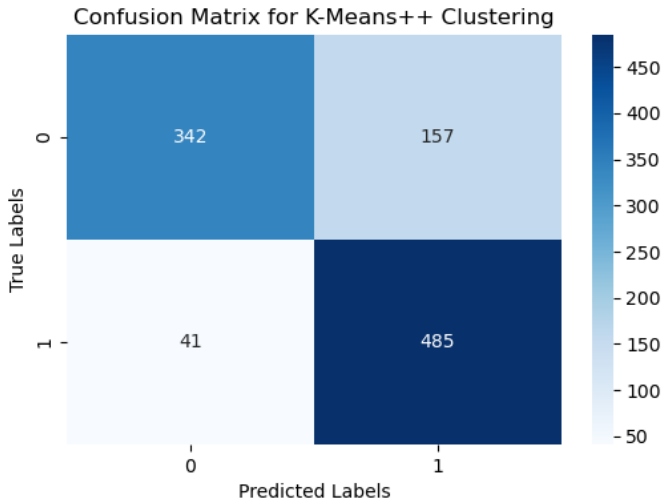
```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("KMeans++ Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_kmeans_plus))
# Print classification report including precision, recall, and f1-score for each label
print("\nKMeans++ Classification Report:")
print(classification_report(y_CSV_test, y_kmeans_plus))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_kmeans_plus)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_kmeans_plus)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.89*	*0.69*	*0.78*	*499*
1	*0.76*	*0.92*	*0.83*	*526*
Accuracy			*0.81*	*1025*
Macro avg	*0.82*	*0.80*	*0.80*	*1025*
Weighted avg	*0.82*	*0.81*	*0.80*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_kmeans_plus), annot=True, fmt='d', cmap='Blues')
```

```
plt.title("Confusion Matrix for KMeans++ Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
plt.show()
```



4. K-Medoids

Step 1 Import K-Medoids Library :

```
from pyclustering.cluster.kmedoids import kmedoids  # K-Medoids clustering algorithm
```

Step 2 Train / Predict :

```
# ------------------------------------------------------------
# Select initial points for K-Medoids
# K-Medoids requires pre-selecting some points as "medoids" – cluster representatives
# Here, we assume the first 2 points are selected as initial medoids (you can choose others)
initial_medoids = [0, 1]
# ------------------------------------------------------------
# Initialize and run the K-Medoids algorithm
# kmedoids is the algorithm object, provided with data and initial medoids
kmedoids_instance = kmedoids(X_scaled, initial_medoids)
kmedoids_instance.process()  # Execute the algorithm
# ------------------------------------------------------------
# Retrieve clustering results
clusters = kmedoids_instance.get_clusters()  # Returns a list of clusters, each cluster is a list of data point indices
# Initialize the array for cluster labels, set all to 0 initially
y_kmedoids = np.zeros(X_scaled.shape[0])
```

```
# Assign cluster labels to each data point based on the clustering result
for idx, cluster in enumerate(clusters):
    for data_index in cluster:
        y_kmedoids[data_index] = idx
```

Step 3 Label Switching :

```
# Note: Since K-Medoids is unsupervised, cluster labels (0/1) may not match true labels.
# Therefore, calculate accuracy for both original and flipped labels.
acc_original = accuracy_score(y_CSV_test, y_kmedoids)
acc_flipped = accuracy_score(y_CSV_test, 1 - y_kmedoids)
# If flipping labels gives higher accuracy, perform the flip
if acc_flipped > acc_original:
    y_kmedoids = 1 - y_kmedoids
```

Step 4 Print Score :

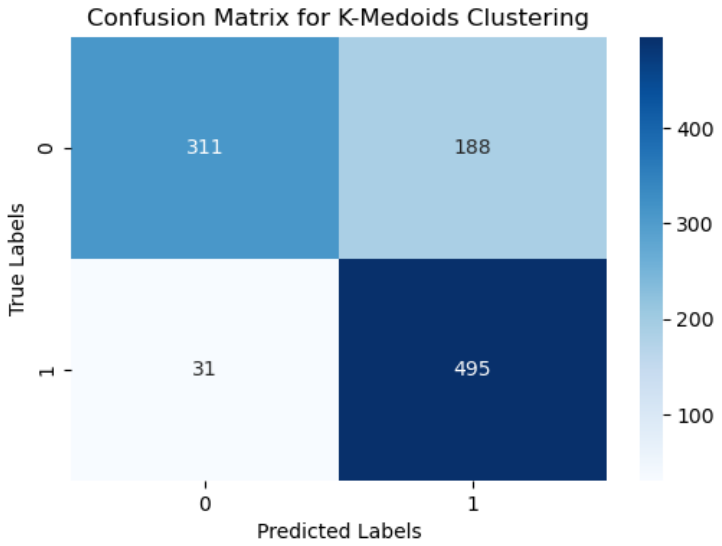
```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("K-Medoids Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_kmedoids))
# Print classification report including precision, recall, and f1-score for each label
print("\nK-Medoids Classification Report:")
print(classification_report(y_CSV_test, y_kmedoids))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_kmedoids)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_kmedoids)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.91*	*0.62*	*0.74*	*499*
1	*0.72*	*0.94*	*0.82*	*526*
Accuracy			*0.79*	*1025*
Macro Avg	*0.82*	*0.78*	*0.78*	*1025*
Weighted Avg	*0.81*	*0.79*	*0.78*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
```

```
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_kmedoids), annot=True, fmt='d',
cmap='Blues')
plt.title("Confusion Matrix for K-Medoids Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
plt.show()
```



5. Fuzzy C-Means (FCM)

Step 1 Import Fuzzy Library :

```
import skfuzzy as fuzz  # Library for performing Fuzzy C-Means
```

Step 2 Train / Predict :

```
# ------------------------------------------------------------
# Apply the Fuzzy C-Means algorithm:
# - X_scaled.T: transpose because cmeans requires shape (features, samples)
# - 2: number of clusters
# - 2: fuzziness coefficient (m=2 is common, indicates the "softness" of clustering)
# - error=0.005: stopping threshold for the algorithm
# - maxiter=1000: maximum number of iterations
c, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T, 2, 2, error=0.005,
maxiter=1000)
# For each data point, assign the label of the cluster with the highest membership
probability
y_fuzzy = np.argmax(u, axis=0)
```

Step 3 Label Switching :

```
# Since cluster labels are unsupervised, they may be flipped (0 ↔ 1) → check to choose the optimal labeling
acc_original = accuracy_score(y_CSV_test, y_fuzzy)
acc_flipped = accuracy_score(y_CSV_test, 1 - y_fuzzy)
# Flip the labels if the flipped version gives higher accuracy
if acc_flipped > acc_original:
    y_fuzzy = 1 - y_fuzzy
```

Step 4 Print Score :

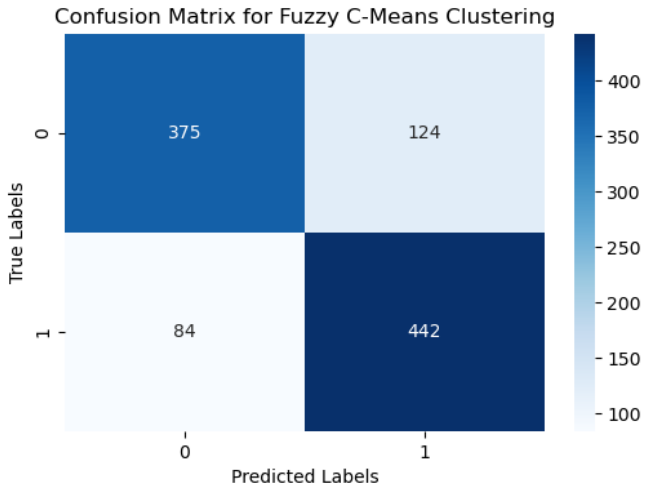
```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("Fuzzy Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_fuzzy))
# Print classification report including precision, recall, and f1-score for each label
print("\nFuzzy Classification Report:")
print(classification_report(y_CSV_test, y_fuzzy))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_fuzzy)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_fuzzy)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.79*	*0.75*	*0.77*	*499*
1	*0.77*	*0.81*	*0.79*	*526*
Accuracy			*0.78*	*1025*
Macro Avg	*0.78*	*0.78*	*0.78*	*1025*
Weighted Avg	*0.78*	*0.78*	*0.78*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_fuzzy), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix for Fuzzy Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
```

plt.show()



6. Gaussian Mixture Models (GMM)

Step 1 Import GMM Library :

```
from sklearn.mixture import GaussianMixture  # GMM – probabilistic clustering model
```

Step 2 Train / Predict :

```
# Apply GMM (Gaussian Mixture Model) with 2 clusters (since there are 2 classes: disease and no disease)
gmm = GaussianMixture(n_components=2, random_state=42)  # Initialize GMM model
y_gmm = gmm.fit_predict(X_scaled)  # Train and predict clusters
```

Step 3 Label Switching :

```
# GMM may swap labels (cluster 0 might correspond to label 1 and vice versa),
# so we try both label assignments and choose the one with higher accuracy
acc_original = accuracy_score(y_CSV_test, y_gmm)
acc_flipped = accuracy_score(y_CSV_test, 1 - y_gmm)
# Flip labels if needed to match the true labels
if acc_flipped > acc_original:
    y_gmm = 1 - y_gmm
```

Step 4 Print Score :

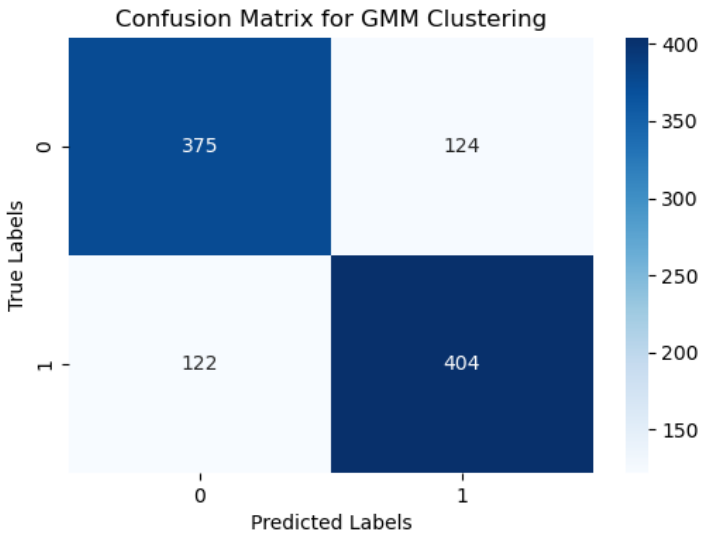
```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("GMM Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_gmm))
# Print classification report including precision, recall, and f1-score for each label
```

```
print("\nGMM Classification Report:")
print(classification_report(y_CSV_test, y_gmm))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_gmm)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_gmm)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	*Precision*	*Recall*	*F1-score*	*Support*
0	*0.75*	*0.75*	*0.75*	*499*
1	*0.77*	*0.77*	*0.77*	*526*
Accuracy			*0.76*	*1025*
Macro Avg	*0.76*	*0.76*	*0.76*	*1025*
Weighted Avg	*0.76*	*0.76*	*0.76*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_gmm), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix for GMM Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
plt.show()
```



7. Agglomerative Hierarchical Clustering (AHC)

Step 1 Import AHC Library :

```
from sklearn.cluster import AgglomerativeClustering  # Agglomerative hierarchical clustering algorithm
```

Step 2 Train / Predict :

```
agg_clustering = AgglomerativeClustering(n_clusters=2)  # Create model with 2 clusters
y_agg = agg_clustering.fit_predict(X_scaled)  # Predict cluster labels (0 or 1)
```

Step 3 Label Switching :

```
# Evaluate which label assignment is correct, since clustering doesn't know true labels and may be flipped
acc_original = accuracy_score(y_CSV_test, y_agg)  # Compare true labels with predicted cluster labels
acc_flipped = accuracy_score(y_CSV_test, 1 - y_agg)  # Flip cluster labels and compare again
# Use the flipped labels if it gives better results
if acc_flipped > acc_original:
    y_agg = 1 - y_agg
```

Step 4 Print Score :

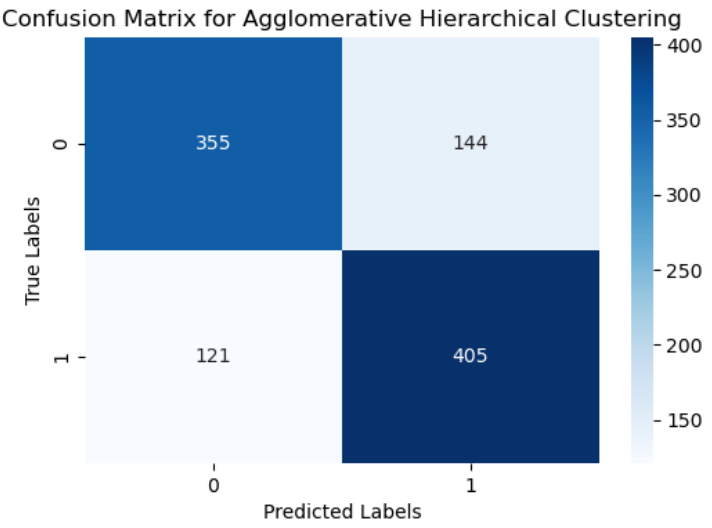
```
# Print confusion matrix to clearly see number of correct/incorrect samples between true and predicted labels
print("AHC Confusion Matrix:")
print(confusion_matrix(y_CSV_test, y_agg))
# Print classification report including precision, recall, and f1-score for each label
print("\n AHC Classification Report:")
print(classification_report(y_CSV_test, y_agg))
# Calculate clustering quality metrics
sil_score = silhouette_score(X_scaled, y_agg)  # Silhouette Score: closer to 1 is better
dbi_score = davies_bouldin_score(X_scaled, y_agg)  # Davies-Bouldin Index: lower is better
# Print the scores
print(f"Silhouette Score: {sil_score}")
print(f"Davies-Bouldin Index: {dbi_score}")
```

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.75*	*0.71*	*0.73*	*499*
1	*0.74*	*0.77*	*0.75*	*526*

Lớp	*Precision*	*Recall*	*F1-score*	*Support*
Accuracy			*0.74*	*1025*
Macro Avg	*0.74*	*0.74*	*0.74*	*1025*
Weighted Avg	*0.74*	*0.74*	*0.74*	*1025*

Step 5 Visualize :

```
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))  # Chart size
sns.heatmap(confusion_matrix(y_CSV_test, y_agg), annot=True, fmt='d',
cmap='Blues')
plt.title("Confusion Matrix for AHC Clustering")  # Chart title
plt.xlabel("Predicted Labels")  # X-axis label
plt.ylabel("True Labels")       # Y-axis label
plt.show()
```



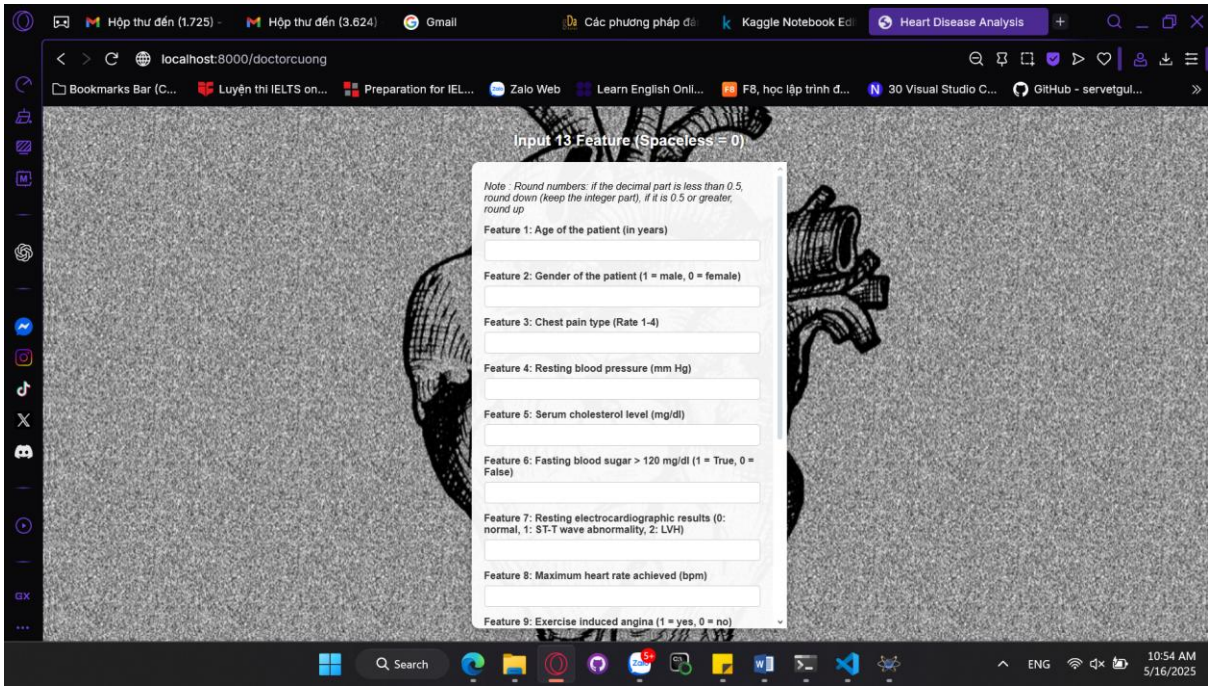
8. Ensemble (Hard Voting)

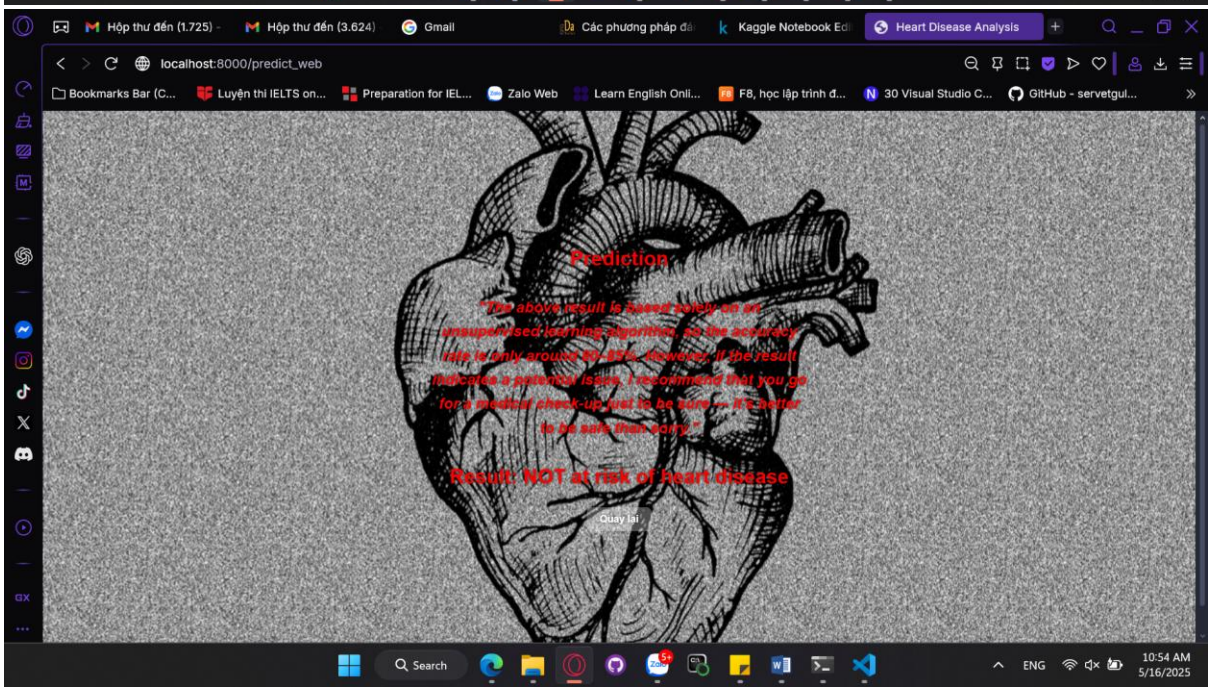
```
### ----- Hard Voting from Clustering Methods ----- ###
# Combine predictions from different clustering models into a DataFrame
labels = pd.DataFrame({
    'kmeans': y_kmeans,
    'kmeans_plus': y_kmeans_plus,
    'kmedoids': y_kmedoids,
    'Fuzzy': y_fuzzy,
    'agg': y_agg,
    'gmm': y_gmm
})
# Function to flip labels if the flipped version yields significantly better accuracy
def best_label_match(pred, y_true_test):
```

```
    acc1 = accuracy_score(y_true_test, pred)         # Accuracy with current labels
    acc2 = accuracy_score(y_true_test, 1 - pred)     # Accuracy with flipped labels
    if acc2 - acc1 > 0.05:                           # Threshold: flip only if gain > 5%
        return 1 - pred
    else:
        return pred
# Align predicted labels for each model with the true labels
for col in labels.columns:
    labels[col] = best_label_match(labels[col], y_CSV_test)
# Perform hard voting: choose the most common label among models for each sample
final_pred = mode(labels.values, axis=1)[0].flatten()
# Print final confusion matrix and classification metrics
print("Final Confusion Matrix:")
print(confusion_matrix(y_CSV_test, final_pred))
print("\nFinal Classification Report:")
print(classification_report(y_CSV_test, final_pred))
# Evaluate clustering quality using Silhouette Score (higher is better)
print("Silhouette Score:", silhouette_score(X_scaled, final_pred))
# Evaluate using Davies-Bouldin Index (lower is better)
print("Davies-Bouldin Index:", davies_bouldin_score(X_scaled, final_pred))
# Visualize final confusion matrix with a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_CSV_test, final_pred), annot=True, fmt='d',
cmap='Blues')
plt.title("Final Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.86*	*0.69*	*0.77*	*499*
1	*0.76*	*0.90*	*0.82*	*526*
Accuracy			*0.80*	*1025*
Macro Avg	*0.81*	*0.80*	*0.79*	*1025*
Weighted Avg	*0.81*	*0.80*	*0.80*	*1025*

9. Deploy Web For Testing (Optional)





VI. Analysis

1. Model Performance Overall

1.1. Accuracy Overview

Model	Accuracy
KMeans / KMeans++	*0.81*
Fuzzy C-Means	*0.79*

Model	Accuracy
K-Medoids	*0.79*
GMM	*0.76*
Agglomerative Clustering	*0.74*
Combined Model	*0.80*

We can see *K-Means* and *K-Means++* achieved the highest accuracy (0.81), followed closely by the combined model (0.80). *GMM* and *Agglomerative Clustering* performed a little bit worse.

1.2. Precision, Recall, and F1-Score

Class 0 (No Disease):

- Highest precision from *K-Medoids* (0.91), meaning fewer false positives.
- However, recall is low (0.62), indicating many actual cases were missed.

Class 1 (Disease):

- Highest recall from *K-Means/K-Means++* (0.92), which is ideal for disease detection where minimizing false negatives is crucial.

So *K-Means/K-Means++* and the combined model show better balance between precision and recall, especially for identifying diseased patients.

1.3. Clustering Quality Metrics: Silhouette Score and Davies-Bouldin Index

Model	***Silhouette Score ↑***	***Davies-Bouldin Index ↓***
KMeans / KMeans++	*0.169*	*2.208*
Fuzzy C-Means	*0.155*	*2.298*
K-Medoids	*0.164*	*2.230*
GMM	*0.111*	*2.764*
Agglomerative Clustering	*0.133*	*2.636*
Combined Model	*0.165*	*2.243*

So *K-Means/K-Means++* have the highest silhouette score and the lowest Davies-Bouldin Index, indicating the clearest and most well-separated clusters.

1.4. Combined Model Performance

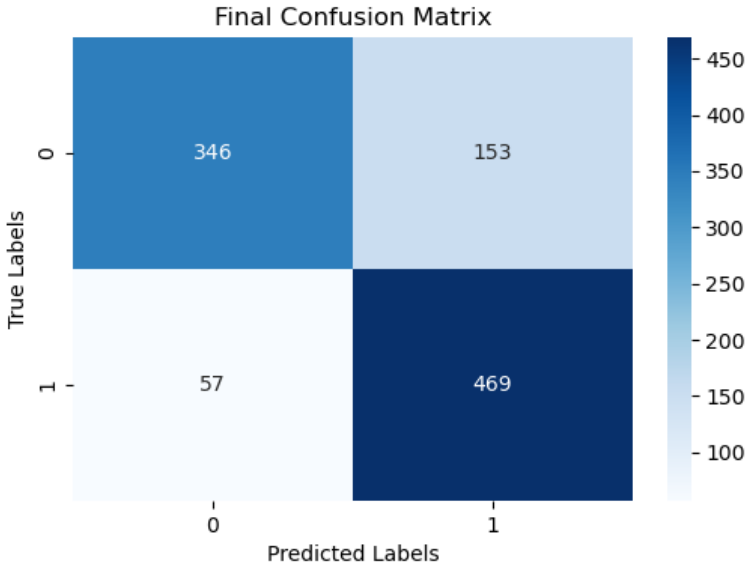
The ensemble or voting-based combination across all clustering models yields:

- Accuracy = 0.80
- F1-score = 0.77 (class 0), 0.82 (class 1)
- Silhouette = 0.165
- Davies-Bouldin Index = 2.24

Very competitive performance, maintaining balance and avoiding biases from individual models.

Lớp	***Precision***	***Recall***	***F1-score***	***Support***
0	*0.86*	*0.69*	*0.77*	*499*

Lớp	*Precision*	*Recall*	*F1-score*	*Support*
1	*0.75*	*0.89*	*0.82*	*526*
Accuracy			*0.80*	*1025*
Macro Avg	*0.81*	*0.79*	*0.79*	*1025*
Weighted Avg	*0.80*	*0.80*	*0.79*	*1025*



1.5. Conclusion and Improvements Achieved

By combining multiple clustering algorithms (*K-Means*, *K-Means++*, *Fuzzy C-Means*, *K-Medoids*, *GMM*, *Agglomerative Clustering*), the final ensemble model achieved a balanced and improved performance compared to most individual models.

Improved Generalization:

- The combined model reaches 80% accuracy, closely matching the best individual methods (*K-Means/K-Means++*).
- It avoids overfitting to any single method's bias by integrating their strengths.

Balanced Classification:

- F1-score for Class 0 (Healthy): 0.77
- F1-score for Class 1 (Diseased): 0.82
- This shows a well-balanced model, especially important in healthcare, where both false positives and false negatives can be costly.

Low False Negatives:

- Only 57 false negatives, meaning the model effectively detects actual patients with heart disease—critical for medical diagnostics.

Robust Cluster Quality:

- Silhouette Score: 0.165
- Davies-Bouldin Index: 2.24

- These metrics indicate that the clusters are reasonably well-separated and compact, supporting the model's internal consistency.

Consistency Across Metrics:

- Unlike some individual models that perform well in one metric but poorly in another, the combined model maintains stability across accuracy, precision, recall, F1-score, and clustering metrics.

Final remark is the use of a combined model demonstrates the power of ensemble techniques even in unsupervised learning. While no individual model was perfect, the aggregation of predictions led to a more stable, accurate, and clinically useful result particularly valuable for early diagnosis or decision support in cardiovascular disease prediction.

VII. Conclusion

1. Task Performed

- Collected and preprocessed the Cleveland Heart Disease dataset.
- Applied multiple unsupervised clustering algorithms: *K-Means*, *K-Means++*, *Fuzzy C-Means*, *K-Medoids*, *GMM*, and *Agglomerative Clustering*.
- Converted clustering labels to match true labels using mapping techniques.
- Evaluated models using: *Confusion Matrix*, *Classification Report* (Precision, Recall, F1-score), *Clustering metrics* (Silhouette Score, Davies-Bouldin Index)
- Implemented a voting ensemble to combine predictions from all six algorithms.

2. Achieved Results

- Combined model accuracy: 80%
- F1-score: 0.77 (Healthy), 0.82 (Disease)
- Silhouette Score: 0.165 (indicates moderate cluster quality)
- Davies-Bouldin Index: 2.24 (acceptable compactness and separation)
- Reduced false negatives (only 57), crucial for disease prediction

3. Gained Experience

- Developed strong understanding of unsupervised learning and clustering evaluation.
- Learned how to align cluster labels with true class labels using confusion matrix analysis.
- Practiced ensemble learning even in an unsupervised context to improve model robustness.
- Gained experience in model validation and performance interpretation beyond accuracy (e.g., Silhouette Score, DBI).

4. Limitations and Weaknesses

- Low Silhouette Score and High DBI: Indicates overlapping or unclear boundaries between clusters.

- Model sensitivity to initial centroids (especially in *K-Means*): Can lead to different results across runs.
- *GMM* and *Agglomerative Clustering* had lower accuracy (≈74–76%)These algorithms may not capture the data structure effectively.
- No feature selection or dimensionality reduction used: Possibly affecting clustering clarity and increasing noise.

5. Proposed Improvements

- Tune hyperparameters (distance metrics, initialization method,…) more systematically.
- Use soft clustering methods like *Fuzzy C-Means* more effectively by weighting final votes.
- Explore hybrid models (semi-supervised learning) where available labels guide clustering.
- Apply model stability techniques, such as multiple runs with averaged results, to reduce randomness.

LIST OF REFERENCES

1. UCI Machine Learning Repository. *Heart Disease Dataset*. Retrieved from https://archive.ics.uci.edu/dataset/45/heart+disease
2. Smith, J. *Heart Disease Dataset*. Kaggle. Retrieved from https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset
3. GeeksforGeeks. *K-Means Clustering – Introduction*. Retrieved from https://www.geeksforgeeks.org/k-means-clustering-introduction/
4. Tiep, L. V. (2017). *KMeans Clustering*. Machine Learning Cơ Bản. Retrieved from https://machinelearningcoban.com/2017/01/01/kmeans/
5. GeeksforGeeks. *ML – K-Means Algorithm*. Retrieved from https://www.geeksforgeeks.org/ml-k-means-algorithm/
6. GeeksforGeeks. *ML – K-Medoids Clustering with Example*. Retrieved from https://www.geeksforgeeks.org/ml-k-medoids-clustering-with-example/
7. GeeksforGeeks. *K-Means vs K-Medoids Clustering*. Retrieved from https://www.geeksforgeeks.org/k-means-vs-k-medoids-clustering/
8. GeeksforGeeks. *ML – Fuzzy Clustering*. Retrieved from https://www.geeksforgeeks.org/ml-fuzzy-clustering/
9. GeeksforGeeks. *Gaussian Mixture Models (GMM) & Covariances in Scikit-learn*. Retrieved from https://www.geeksforgeeks.org/gaussian-mixture-models-gmm-covariances-in-scikit-learn/
10. GeeksforGeeks. *Hierarchical Clustering*. Retrieved from https://www.geeksforgeeks.org/hierarchical-clustering/
11. GeeksforGeeks. *Implementing Agglomerative Clustering using Sklearn*. Retrieved from https://www-geeksforgeeks-org.translate.goog/implementing-agglomerative-clustering-using-sklearn/?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=tc
12. GeeksforGeeks. *A Comprehensive Guide to Ensemble Learning*. Retrieved from https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/
13. GeeksforGeeks. *Types of Ensemble Learning*. Retrieved from https://www.geeksforgeeks.org/types-of-ensemble-learning/

MY CODE

Note : The entire code above was written and researched by myself (with the help of AI tools during the debugging process). I did not steal anyone's intellectual property, and I firmly assure you of that.

1. Nguyễn, C. T. *Heart Valid [Notebook]*. Kaggle. https://www.kaggle.com/code/nguynchutncng/heart-valid
2. Nguyễn, C. T. *Heart Train [Notebook]*. Kaggle. https://www.kaggle.com/code/nguynchutncng/heart-train
3. Aysinemu. *Machine Learning*. GitHub. https://github.com/aysinemu/Machine-Learning