Malnad College Of Engineering , Hassan

(An Autonomous Institution under VTU, Belagavi , Karnataka)

"MACHINE LEARNING ACTIVITY"

Course Code : 22CS601

TOPIC

"Predictive Modelling for Cardiovascular Disease Risk Using Machine Learning Techniques"

Team Members :

NAME	USN
PRAVEEN H M	4MC22CS119
PRAVEEN KUMAR M R	4MC22CS120
PRIYANKA B N	4MC22CS121
R ANISHA	4MC22CS122
RAHUL NAIKAR	4MC22CS123

Under The Guidance Of :

Dr. Keerthi Kumar H M

(Associate professor)

Department Of Computer Science And Engineering

2022-2026

Malnad College Of Engineering, Hassan

Real Word Problem

“Can we predict whether a patient is at risk of cardiovascular disease using basic health check-up data?”

Dataset Chosen

Cardiovascular Disease dataset from the UCI Machine Learning Repository or from Kaggle. It’s commonly referred to as the **"cardio_train.csv"** file and is part of a dataset for predicting the risk of cardiovascular disease based on medical and lifestyle attributes.

Source:Kaggle

https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

Dataset Description

The Cardiovascular Disease dataset is a real-world medical dataset collected during routine health examinations. It consists of 13 columns and over 70,000 records, where each record represents a patient’s health profile which is sampled to 20,000 records. The features include demographic details like age and gender, physical measurements such as height and weight, and vital signs like systolic and diastolic blood pressure. It also contains important biochemical markers like cholesterol and glucose levels, as well as lifestyle factors such as smoking status, alcohol consumption, and physical activity.

The goal of this dataset is to predict the presence of cardiovascular disease, marked by the target variable cardio (0 = No disease, 1 = Has disease). This dataset is valuable for developing machine learning models that can assist in early detection and prevention of heart-related illnesses, helping healthcare professionals prioritize patients at risk and improve overall public health outcomes.

Key Features

1. **id** – Unique identifier for each patient (not used for prediction).
2. **age** – Patient's age in days (convert to years by dividing by 365).
3. **gender** – Biological sex of the patient: 1 = Female, 2 = Male.
4. **height** – Height of the patient measured in centimetres.
5. **weight** – Weight of the patient measured in kilograms.
6. **ap_hi** – Systolic blood pressure (upper value in a BP reading).
7. **ap_lo** – Diastolic blood pressure (lower value in a BP reading).
8. **cholesterol** – Cholesterol level: 1 = Normal, 2 = Above normal, 3 = Well above normal.
9. **gluc** – Blood glucose level: 1 = Normal, 2 = Above normal, 3 = Well above normal.
10. **smoke** – Smoking status: 0 = Non-smoker, 1 = Smoker.
11. **alco** – Alcohol intake: 0 = Does not consume alcohol, 1 = Does consume alcohol.
12. **active** – Physical activity: 0 = Inactive, 1 = Physically active.
13. **cardio** – Target variable: 0 = No cardiovascular disease, 1 = Has cardiovascular disease.

Algorithms Used

1. KNN (K - Nearest Neighbours)
2. Decision Tree
3. Random Forest
4. SVM (Support Vector Machine)
5. Logistic Regression

K – Nearest Neighbours Algorithm

Introduction

K-Nearest Neighbours (KNN) is a supervised learning algorithm used for classification and regression. It classifies a data point based on how its neighbours are classified. It is non-parametric and does not assume any prior distribution about the data.

How KNN Works?

1. **Feature Scaling**: Since KNN relies on distance metrics, feature scaling (e.g., Standardization) is essential.
2. **Distance Calculation**: It calculates distances (e.g., Euclidean, Manhattan) between the test point and all training points.
3. **Neighbour Voting**: The majority label among the *k* nearest neighbours determines the class of the test point.

Why Use KNN for Cardio Dataset?

1. **Simple and Intuitive**: KNN is easy to understand and implement. It predicts outcomes based on similarity, which makes it ideal for health-related datasets where similar patients may share the same diagnosis.
2. **No Training Phase**: Unlike other models, KNN doesn't require a separate training process. It uses the entire dataset during prediction, which is useful for quick testing and experimentation.
3. **Effective with Clean and Structured Data**: The cardio dataset contains clean, numerical features like age, blood pressure, and cholesterol. KNN handles this type of data well, especially after scaling.
4. **Adaptable to Different Distance Metrics**: KNN allows the use of different distance measures (like Euclidean or Manhattan), which can be tuned to improve accuracy based on the dataset's characteristics.
5. **Good Baseline Model**: KNN provides a strong starting point for classification problems. It can help evaluate whether more complex models are necessary.

Dataset Description

1. **Goal:** Predict whether a patient has cardiovascular disease (cardio: 0 = no, 1 = yes).
2. **Data:** Health records of patients with features like:
    - age (in days),
    - gender (1 = male, 2 = female),
    - height (cm),
    - Blood pressure: ap_hi (systolic), ap_lo (diastolic),
    - cholesterol and gluc levels (1=normal, 2=above normal, 3=high),
    - Lifestyle factors: smoke, alco, active (all binary).
3. **Type:** Binary classification with mixed numeric and categorical features.
4. **Purpose:** Use patient data to predict cardiovascular disease presence.

Data Preprocessing Steps

1. **Sampling:** 20,000 rows were randomly sampled from the dataset for efficiency.
2. **Feature Selection:** The id column (irrelevant) and the cardio label (target) were separated.
3. **Scaling:** Since KNN uses distance-based calculations, Standard Scaler was used to normalize feature values so that no feature dominates others.
4. **Dimensionality Reduction:** PCA (Principal Component Analysis) was applied to reduce the number of input features while keeping most of the data's variance. This improves performance and reduces noise.

KNN Model Building

1. **Data Preparation:** A cardiovascular dataset containing patient health metrics is loaded and sampled to 20,000 rows. Non-informative columns such as id are dropped. The target variable is cardio, which indicates the presence (1) or absence (0) of cardiovascular disease. Features and target labels are separated for further processing.
2. **Data Splitting:** The dataset is divided into training and testing sets using an 80-20 split. The training set is used for model training and hyperparameter tuning, while the test set is reserved for final evaluation.

3. **Feature Scaling:** All numeric features are standardized using z-score normalization. This step ensures that all features contribute equally when distance-based algorithms like KNN calculate similarity between instances.
4. **Dimensionality Reduction with PCA:** Principal Component Analysis (PCA) is applied to reduce the number of input features to 10 principal components. This helps in reducing noise, improving computational efficiency, and potentially increasing classification performance by capturing the most important variance in the data.
5. **Model Initialization and Hyperparameter Tuning:** A K-Nearest Neighbours (KNN) classifier is initialized, and a hyperparameter grid is defined. The grid includes variations in the number of neighbours (neighbours from 1 to 30), distance metrics (Euclidean, Manhattan, Minkowski), weighting schemes (uniform and distance), and the power parameter p for Minkowski distance. GridSearchCV is used to perform an exhaustive search over these combinations using 5-fold cross-validation, selecting the configuration that yields the highest accuracy.
6. **Model Training:** The KNN model is trained using the best-found hyperparameters on the PCA-transformed training data. Cross-validation ensures that the model generalizes well and is not overfitted to any specific subset of the data.
7. **Prediction and Evaluation:** The trained model is applied to the PCA-transformed test set to predict cardiovascular disease outcomes. Model performance is evaluated using accuracy, a classification report (providing precision, recall, and F1-score), and a confusion matrix.
8. **Result Visualization:** A confusion matrix heatmap provides a visual representation of classification results. Additionally, a plot showing how mean accuracy varies with the number of neighbours is generated for the specific case where weights are set to "distance" and the metric is Minkowski with p=2.
9. **Sample Prediction Output:** A small portion of actual versus predicted labels from the test set is displayed to observe individual prediction outcomes.

Model Performance and Visualization

1. **Accuracy Evaluation:** The model's predictive ability is evaluated on the test dataset using accuracy.

   Accuracy measures the proportion of correct predictions out of all predictions made.

This gives an overall idea of how well the model can distinguish between patients with and without cardiovascular disease.

2. **Classification Report:** A classification report is generated to provide deeper insights:
    - **Precision:** Indicates how many of the predicted positive cases are actually positive.
    - **Recall:** Shows how many of the actual positive cases the model successfully identified.
    - **F1-Score:** The harmonic mean of precision and recall, balancing both metrics.

    These metrics are provided for both classes:

    Class 0 (no cardiovascular disease)

    Class 1 (cardiovascular disease)

    This helps in understanding if the model is biased towards any one class or balanced in prediction.

3. **Confusion Matrix**: A confusion matrix is constructed to display:
    - True Positives (correctly predicted positive cases)
    - True Negatives (correctly predicted negative cases)
    - False Positives (incorrectly predicted as positive) correctly
    - False Negatives (incorrectly predicted as negative)
4. **Accuracy vs. K Plot:**
    - To analyse how the choice of K (number of neighbours) affects performance: A line plot is generated using results from the grid search.
    - This plot shows the mean cross-validation accuracy for different k values (1 to 30), specifically when:

        weights = 'distance'

        metric = 'Minkowski'

        p = 2 (equivalent to Euclidean distance)

    The graph helps identify the optimal value of k that gives the best balance between model complexity and performance

5. **Sample Predictions Table:** A small table is printed showing the actual vs predicted labels for the first few test samples.

   This provides a quick, interpretable snapshot of the model's prediction behaviour on individual records.

Conclusion and limitations

Conclusion:

The K-Nearest Neighbours (KNN) algorithm, when combined with preprocessing steps like feature scaling and dimensionality reduction (PCA), can effectively classify patients based on health-related features into those likely or unlikely to have cardiovascular disease.

Through hyperparameter tuning using GridSearchCV, optimal configurations such as the number of neighbours, distance metric, and weighting scheme were identified, leading to improved predictive performance.

Visualization tools like the confusion matrix and accuracy plots further aided in understanding model behaviour and making informed decisions.

Limitations:

1. **Computational Cost:** KNN can be slow and memory-intensive, especially during prediction, since it must compute distances to all training points. This becomes more problematic with larger datasets.
2. **Curse of Dimensionality:** Although PCA was applied, KNN performance can degrade in high-dimensional spaces due to the reduced usefulness of distance metrics.
3. **Imbalanced Sensitivity:** KNN may favour the majority class if the dataset is imbalanced, affecting recall or precision for minority classes.
4. **No Model Learning:** KNN is a lazy learner: it doesn't build an internal model during training, making real-time predictions slower.
5. **Choice of K and Distance Metric:** Model performance is highly sensitive to the choice of k, distance metric, and scaling. Incorrect combinations can lead to underfitting or overfitting.

6. **Affected by Noise and Outliers:** Since KNN uses neighbouring points directly, noisy or mislabelled data points can negatively affect predictions.

Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Load dataset
df = pd.read_csv('cardio_train.csv.zip', sep=';')
# Sample 20,000 instances (adjust if needed)
df_sampled = df.sample(n=20000, random_state=42)
# Features and target
X = df_sampled.drop(['id', 'cardio'], axis=1)
y = df_sampled['cardio']
# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Scale features
scaler = StandardScaler()
```

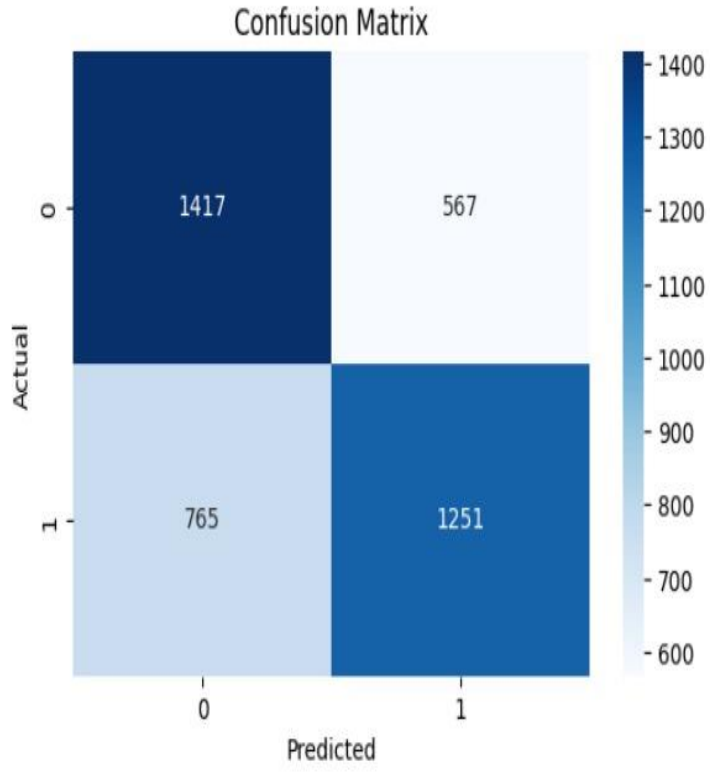
```
X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Apply PCA for dimensionality reduction (try with 10 components)

pca = PCA(n_components=10)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)

print(f"Explained variance ratio by PCA components: {pca.explained_variance_ratio_}")

# Setup KNN and parameter grid for GridSearchCV

knn = KNeighborsClassifier()

param_grid = {

    'n_neighbors': range(1, 31),

    'weights': ['uniform', 'distance'],

    'metric': ['euclidean', 'manhattan', 'minkowski'],

    'p': [1, 2]

}

# Grid search with 5-fold cross-validation

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)

grid_search.fit(X_train_pca, y_train)

print("Best parameters found:", grid_search.best_params_)

print("Best cross-validation accuracy:", grid_search.best_score_)

# Use the best estimator to predict on test data

best_knn = grid_search.best_estimator_

y_pred_final = best_knn.predict(X_test_pca)

# Test accuracy and report
```

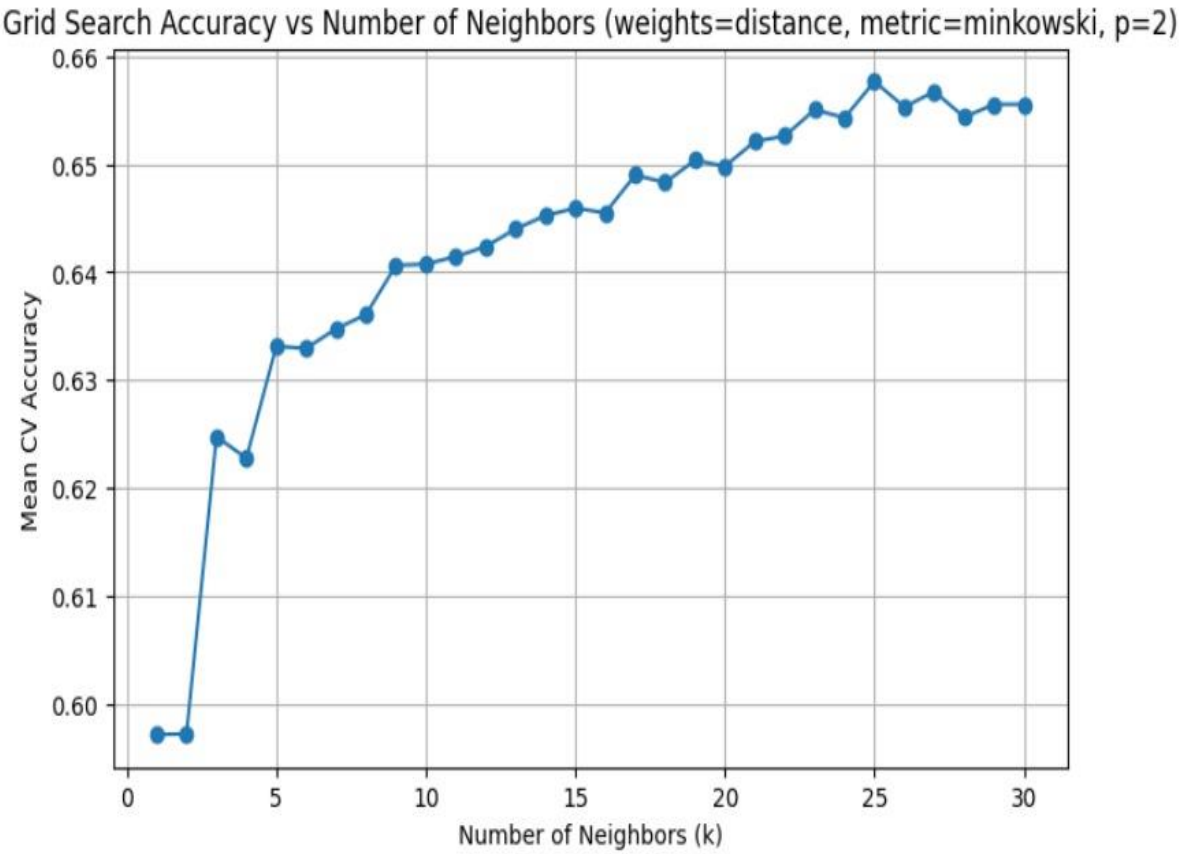
```
acc = accuracy_score(y_test, y_pred_final)

print(f"Test set accuracy with best params: {acc:.4f}")

print("\nClassification Report:\n", classification_report(y_test, y_pred_final))

# Confusion matrix plot

conf_matrix = confusion_matrix(y_test, y_pred_final)

plt.figure(figsize=(6,4))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# Optional: Plot accuracy vs K from grid search results (filter by weights='distance' and metric='minkowski' with p=2) results = pd.DataFrame(grid_search.cv_results_)

filtered = results[(results['param_weights'] == 'distance') &

                   (results['param_metric'] == 'minkowski') &

                   (results['param_p'] == 2)]

plt.figure(figsize=(8,5))

plt.plot(filtered['param_n_neighbors'].astype(int), filtered['mean_test_score'], marker='o')

plt.title('Grid Search Accuracy vs Number of Neighbors (weights=distance, metric=minkowski, p=2)')

plt.xlabel('Number of Neighbors (k)')

plt.ylabel('Mean CV Accuracy')

plt.grid(True)

plt.show()

# Show sample actual vs predicted for first 20 test samples
```

```
sample_results = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred_final}).head(30)

print("\nSample predictions:")

print(sample_results)
```

Output

```
Explained variance ratio by PCA components: [0.17587441 0.1430647  0.10205412 0.09263698 0.09057549 0.08827938
 0.08490119 0.07327328 0.05920324 0.04935363]
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
Best parameters found: {'metric': 'manhattan', 'n_neighbors': 30, 'p': 1, 'weights': 'distance'}
Best cross-validation accuracy: 0.6656875
Test set accuracy with best params: 0.6670

Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.71      0.68      1984
           1       0.69      0.62      0.65      2016

    accuracy                           0.67      4000
   macro avg       0.67      0.67      0.67      4000
weighted avg       0.67      0.67      0.67      4000
```





Sample predictions:

	Actual	Predicted
0	1	1
1	0	0
2	0	0
3	1	1
4	0	0
5	1	0
6	0	1
7	0	1
8	0	0
9	0	0
10	0	1
11	0	0
12	0	0
13	1	0
14	1	0
15	0	0
16	0	0
17	0	1
18	0	0
19	0	0
20	0	0

Decision Tree Algorithm

Introduction

A Decision Tree is a flowchart-like structure used in supervised machine learning. It breaks down a dataset into smaller subsets while developing an associated decision tree incrementally. The final result is a tree with decision nodes and leaf nodes:

Decision node: Splits the data based on a feature.

Leaf node: Represents a final class label.

How Decision Trees Work?

Step-by-step functioning:

1. Select the best feature using a criterion (e.g., Gini Impurity or Information Gain).
2. Split the dataset into subsets based on the feature's values.
3. Recursively apply the process to each derived subset.
4. Stop when a node contains samples of a single class or other stopping criteria are met.

   Splitting Criteria:

   - gini: Measures impurity. Lower is better.
   - entropy: Measures the amount of information (used in ID3 algorithm).

Why Use a Decision Tree?

1. **Interpretability:** Easy to understand and explain.
2. **No Need for Feature Scaling:** Works well with unscaled data.
3. **Handles Non-Linear Relationships:** Naturally captures complex patterns.
4. **Feature Importance:** Automatically ranks feature influence.
5. **Flexible:** Can be used for both classification and regression tasks.

Dataset Description

1. **Source:** cardio_train.csv.zip
2. **Size:** Sampled to 20,000 rows for faster training.
3. **Target Column**: cardio (Binary Classification)
    - 0: No cardiovascular disease
    - 1: Has cardiovascular disease
4. **Key Features:**
    - **Demographics:** age, gender, height, weight
    - **Health metrics:** ap_hi, ap_lo, cholesterol, gluc
    - **Lifestyle**: smoke, alco, active

Data Preprocessing

1. **Sampling:** 20,000 rows randomly selected for training and testing.
2. **Feature Selection:** Removed id as it's not predictive.
3. **Splitting:** Used train_test_split (80% train, 20% test).
4. **Scaling:** Applied StandardScaler (not mandatory for decision trees but done for consistency).

Model Building:

1. **Dataset Preparation:**
    - Loaded the cardiovascular disease dataset containing medical and lifestyle information.
    - Randomly sampled 20,000 records from the full dataset to ensure faster processing.
    - Removed non-informative features like id and isolated the target variable cardio.
2. **Train-Test Split:**
    - Divided the data into training and testing sets using an 80:20 ratio.
    - Ensured that the model could be trained on most of the data while being evaluated on unseen examples.
3. **Feature Scaling (Optional):**
    - Applied standard scaling to the feature set using StandardScaler.

- Although Decision Trees don't require scaling, it was done for consistency across models.

4. **Model Initialization and Hyperparameter Tuning:**
   - Initialized a Decision Tree Classifier.
   - Used RandomizedSearchCV to perform hyperparameter tuning efficiently.
   - Tuned parameters included:
     - max_depth: to control tree depth.
     - min_samples_split: minimum samples required to split a node.
     - min_samples_leaf: minimum samples required at a leaf node.
     - criterion: to choose between Gini impurity or entropy for node splitting.
   - Performed 20 random combinations with 3-fold cross-validation to select the best parameter set.
5. **Model Training:**
   - Trained the Decision Tree using the best parameters identified through Randomized Search.
   - Evaluated the tree structure including depth, number of leaves, and total nodes created.
6. **Prediction and Evaluation:**
   - Made predictions on the test set using the optimized Decision Tree model.
   - Evaluated performance using:
     - Accuracy: overall correctness.
     - Classification Report: includes precision, recall, and F1-score.
     - Confusion Matrix: visual representation of true/false positives and negatives.
7. **Feature Importance:**
   - Identified which features contributed most to decision-making.
   - Features like systolic blood pressure (ap_hi), cholesterol, and age had high importance.
8. **Tree Visualization:**
   - Visualized the full Decision Tree structure using a flowchart format.
   - Displayed the splitting rules, feature thresholds, class predictions, and node impurity.
   - Helped in understanding how the model made decisions for classification.

Model Performance & Visualization

1. **Performance Metrics**
    - **Accuracy Score:** Proportion of correct predictions on the test set.
    - **Classification Report:**
        - **Precision:** Accuracy of positive predictions.
        - **Recall:** Ability to find all positive cases.
        - **F1-score:** Harmonic mean of precision and recall.
2. **Confusion Matrix**
    - Matrix showing correct and incorrect predictions.
    - Helps identify false positives and false negatives.
    - Feature Importance
3. **Plotted bar chart showing the impact of each feature.**
    - Most important features often include age, ap_hi (systolic BP), cholesterol.
4. **Tree Visualization**
    - Plotted the full decision tree:
    - Nodes show feature name, threshold, impurity, and class distribution.
    - Color-coded for easier understanding of class predictions.

Conclusion and Limitations

Conclusion

1. The Decision Tree model effectively classifies patients based on medical data.
2. It provides transparency in decision-making through visual tree structure.
3. Feature importance highlights critical health indicators affecting cardiovascular health.
4. Model performs well with reasonable accuracy and interpretability.

Limitations

1. **Overfitting:** Trees can fit noise in training data if not properly pruned.
2. **Instability:** Slight changes in data can lead to a completely different tree.
3. **Bias Toward Dominant Features:** May favour features with more levels or unique values.

4. **Lower Accuracy vs Ensemble Methods:** Typically outperformed by Random Forests or Gradient Boosted Trees in complex problems.
5. **Non-smooth Decision Boundaries:** Creates axis-aligned splits, which might not generalize well on continuous data.
6. **Non-smooth Decision Boundaries:** Creates axis-aligned splits, which might not generalize well on continuous data.

Code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, RandomizedSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from scipy.stats import randint

# Load dataset

df = pd.read_csv('cardio_train.csv.zip', sep=';')

# Sample 10,000 instances for faster training

df_sampled = df.sample(n=20000, random_state=42)

# Features and targeting

X = df_sampled.drop(['id', 'cardio'], axis=1)

y = df_sampled['cardio']

# Train-test split

X_train, X_test, y_train, y_test = train_test_split(
```

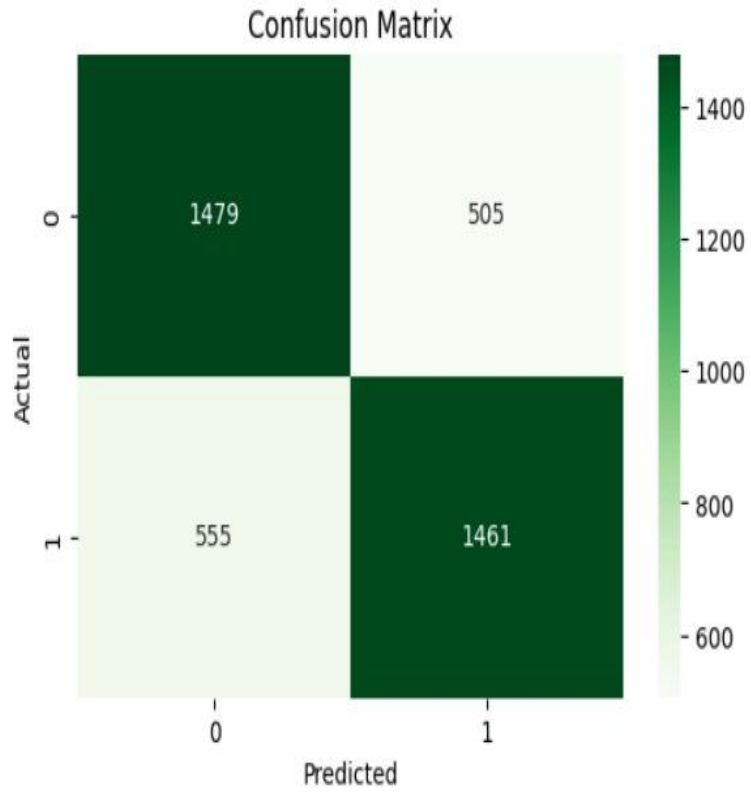
```
    X, y, test_size=0.2, random_state=42)
# (Optional) Scale data - not required for Decision Trees
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Define Decision Tree and hyperparameter search space
dt = DecisionTreeClassifier(random_state=42)
param_dist = {
    'max_depth': [5, 10, 15, None],
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 5),
    'criterion': ['gini', 'entropy']
}
# RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=dt,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=1
)
# Train the model
```

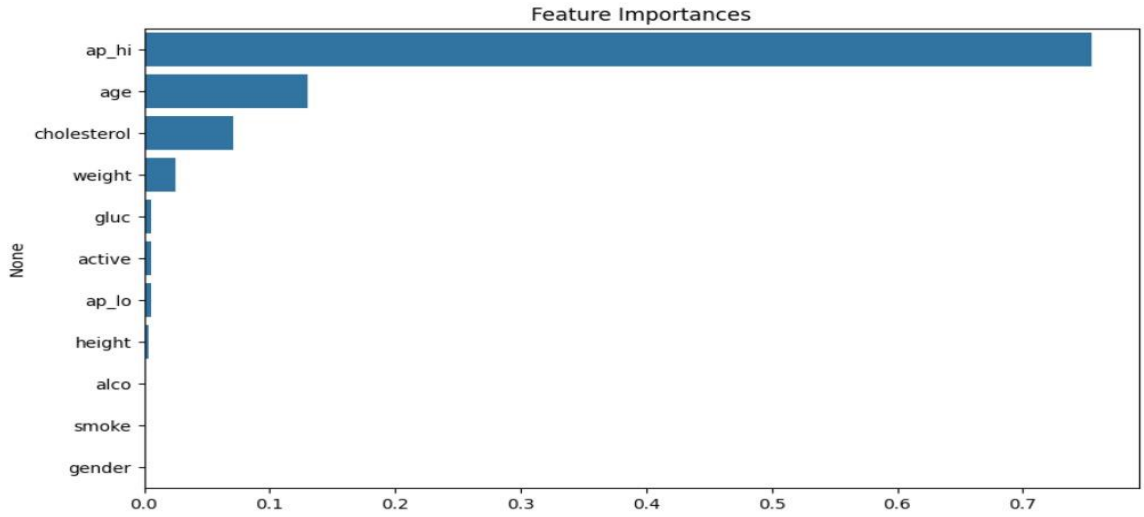
```
random_search.fit(X_train_scaled, y_train)

best_dt = random_search.best_estimator_

# Fit model

random_search.fit(X_train_scaled, y_train)

best_dt = random_search.best_estimator_

# Tree size details

print("\n🌳 Decision Tree Structure Info:")

print("Tree depth:", best_dt.get_depth())

print("Number of leaves:", best_dt.get_n_leaves())

print("Total nodes:", best_dt.tree_.node_count)

# Predict

y_pred = best_dt.predict(X_test_scaled)

# Metrics

print(f"\nBest Parameters: {random_search.best_params_}")

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")

print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens')

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()
```

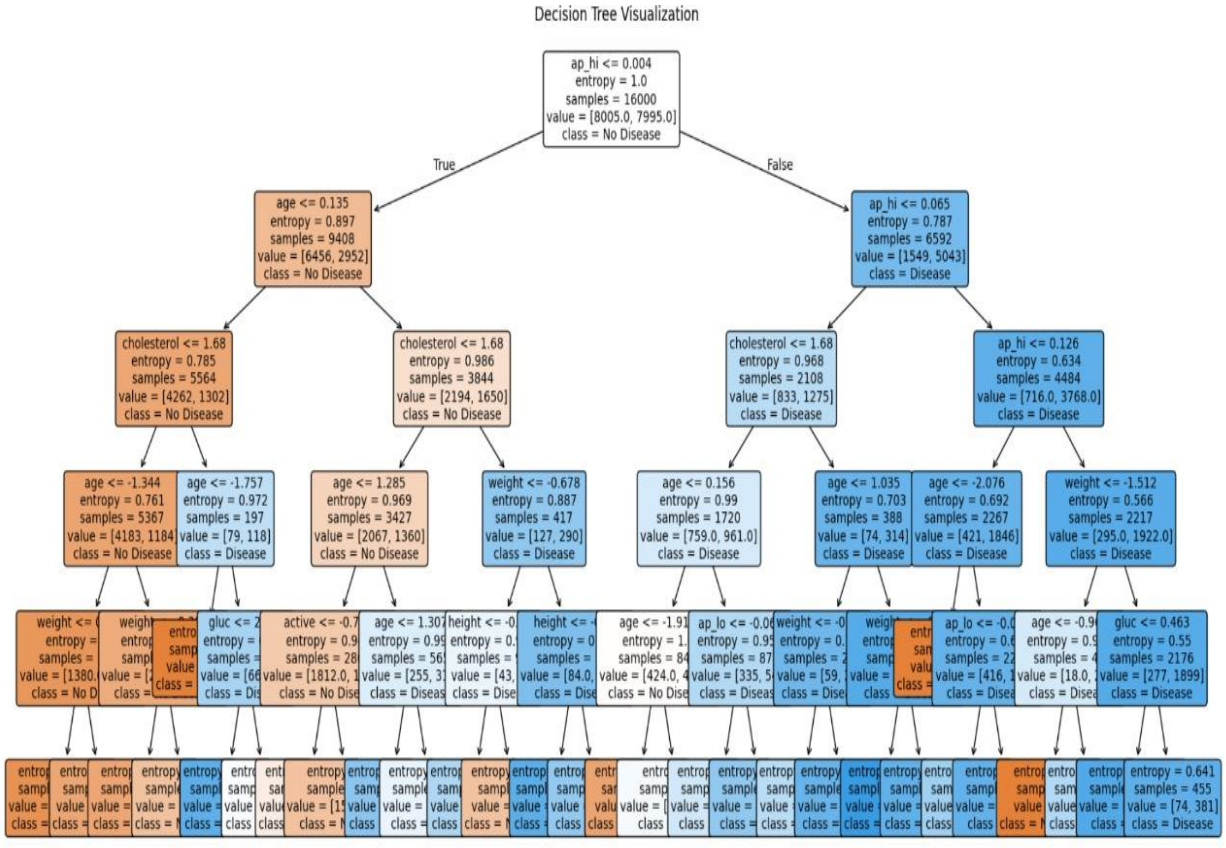
```
# Feature Importance
features = X.columns
importances = best_dt.feature_importances_
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10,6))
sns.barplot(x=importances[indices], y=features[indices])
plt.title("Feature Importances")
plt.show()
# 🌳 Visualize the Tree
plt.figure(figsize=(20,10))
plot_tree(best_dt, feature_names=X.columns, class_names= ['No Disease', 'Disease'],
          filled=True, rounded=True, fontsize=10)
plt.title("Decision Tree Visualization")
plt.show()
```

Output

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits

🌳 Decision Tree Structure Info:
Tree depth: 5
Number of leaves: 30
Total nodes: 59

Best Parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 16}
Accuracy: 0.7350

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.75      0.74      1984
           1       0.74      0.72      0.73      2016

    accuracy                           0.73      4000
   macro avg       0.74      0.74      0.73      4000
weighted avg       0.74      0.73      0.73      4000
```




Feature Importances
ap_hi
age
cholesterol
weight
gluc
active
ap_lo
height
alco
smoke
gender
None
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Decision Tree Visualization
ap_hi <= 0.004
entropy = 1.0
samples = 16000
value = [8005.0, 7995.0]
class = No Disease
True
False
age <= 0.135
entropy = 0.897
samples = 9408
value = [6456, 2952]
class = No Disease
ap_hi <= 0.065
entropy = 0.787
samples = 6592
value = [1549, 5043]
class = Disease
cholesterol <= 1.68
entropy = 0.785
samples = 5564
value = [4262, 1302]
class = No Disease
cholesterol <= 1.68
entropy = 0.986
samples = 3844
value = [2194, 1650]
class = No Disease
cholesterol <= 1.68
entropy = 0.968
samples = 2108
value = [833, 1275]
class = Disease
ap_hi <= 0.126
entropy = 0.634
samples = 4484
value = [716.0, 3768.0]
class = Disease
age <= -1.344
entropy = 0.761
samples = 5367
value = [4183, 1184]
class = No Disease
age <= -1.757
entropy = 0.972
samples = 197
value = [79, 118]
class = Disease
age <= 1.285
entropy = 0.969
samples = 3427
value = [2067, 1360]
class = No Disease
weight <= -0.678
entropy = 0.887
samples = 417
value = [127, 290]
class = Disease
age <= 0.156
entropy = 0.99
samples = 1720
value = [759.0, 961.0]
class = Disease
age <= 1.035
entropy = 0.703
samples = 388
value = [74, 314]
class = Disease
age <= -2.076
entropy = 0.692
samples = 2267
value = [421, 1846]
class = Disease
weight <= -1.512
entropy = 0.566
samples = 2217
value = [295.0, 1922.0]
class = Disease
gluc <= 0.463
entropy = 0.55
samples = 2176
value = [277, 1899]
class = Disease
entropy = 0.641
samples = 455
value = [74, 381]
class = Disease

Random Forest algorithm

Introduction

Random Forest is a powerful machine learning algorithm that belongs to the ensemble family. It combines the predictions of multiple decision trees to improve overall accuracy and robustness. Instead of relying on a single decision tree—which can easily overfit—Random Forest creates many trees on random subsets of data and features, then aggregates their predictions. This ensemble approach reduces variance and helps the model generalize better to unseen data, making it one of the most widely used algorithms in classification and regression problems.

How Random Forest Works?

The core idea behind Random Forest is to grow many decision trees during training time. For each tree:

1. A random subset of the training data is chosen (bootstrapping).
2. At each node, a random subset of features is selected to find the best split.
3. Each tree is grown to the maximum extent (or controlled by parameters like max depth).
4. Once all trees are built, predictions are made by aggregating their outputs. For classification, it uses majority voting; for regression, it averages the results.
5. This randomness ensures trees are decorrelated, improving the ensemble's predictive power and reducing overfitting compared to single trees.

Why Use Random Forest?

Random Forest offers several benefits:

1. **Robustness to Noise:** The ensemble averages out noisy predictions from individual trees.
2. **Feature Importance:** It provides an estimate of feature importance, helping interpret which variables influence predictions.
3. **Handles Non-linearity and Interactions:** It captures complex feature interactions without explicit modelling.

4. **Minimal Data Preparation**: Works well even without extensive feature scaling or normalization.
5. **Reduced Overfitting:** Due to averaging multiple trees built from random subsets.

This makes it a go-to choice for many real-world classification problems like predicting cardiovascular disease risk.

Dataset Description

The dataset is from a cardiovascular disease study, containing over 700,000 records with features such as:

1. **Demographics:** Age (in days), height, weight
2. **Clinical measurements:** Systolic blood pressure (ap_hi), diastolic blood pressure (ap_lo)
3. **Target:** cardio (binary label where 0 = no disease, 1 = presence of disease)

This dataset provides rich physiological indicators for predicting cardiovascular risk.

Data Preprocessing

Data preprocessing is critical for model quality:

1. **Sampling:** To speed up experiments, a subset of 20,000 samples was randomly selected.
2. **Filtering:** Removed unrealistic blood pressure values (ap_hi between 80 and 200, ap_lo between 50 and 150) to avoid noise.
3. **Feature Engineering:**
    - Calculated BMI as weight/(height in meters)^2 — a standard obesity indicator.
    - Converted age from days to years for interpretability.
    - Calculated pulse pressure (difference between systolic and diastolic BP), an important cardiovascular health marker.
4. **Binning:** Created categorical variables from continuous ones to reduce noise and improve tree splitting:
    - Age categories (e.g., <40, 40–50, etc.)

- BMI categories (underweight, normal, overweight, obese)
- Pulse pressure categories

5. **Scaling:** Applied standard scaling to normalize features, important for algorithms sensitive to feature magnitude.

Model Building

1. **Dataset Preparation:**
   - Loaded the dataset and sampled 20,000 rows for efficiency.
   - Removed outliers in blood pressure to clean the data.
2. **Feature Engineering:**
   - Created new features: BMI, age in years, and pulse pressure.
   - Categorized age, BMI, and pulse pressure into bins for better splitting.
3. **Feature and Target Selection:**
   - Dropped irrelevant columns (id, age, height, weight).
   - Defined X as features and y as the target (cardio).
4. **Train-Test Split:**
   - Split data into 80% training and 20% testing sets.
5. **Feature Scaling:**
   - Applied standard scaling for consistency (optional for Random Forests).
6. **Model Tuning with GridSearchCV:**
   - Used GridSearchCV to find the best parameters like number of trees, depth, and split criteria.
   - Performed 5-fold cross-validation for reliable tuning.
7. **Training and Prediction:**
   - Trained the model with the best parameters.

Model Performance and Visualization

1. **Accuracy:** The model's accuracy on the test set indicates how well it predicts unseen data.

2. **Classification report:** Displays precision, recall, and F1-score for both classes, providing detailed insight into performance.
3. **Confusion matrix:** Visualizes the counts of true positives, true negatives, false positives, and false negatives, helping identify types of errors.
4. **Feature importance:** The Random Forest assigns scores to each feature based on how much it decreases impurity across all trees, revealing which features are most predictive. In this case, pulse pressure and BMI categories often have high importance.
5. **Tree visualization:** Plotting a few individual trees helps understand the decision logic, although single trees in a forest are often less interpretable than standalone decision trees.

Conclusion and Limitations

Conclusion: The Random Forest model effectively predicts cardiovascular disease risk based on clinical and demographic features, after careful preprocessing and hyperparameter tuning. It balances bias and variance well, yielding robust results.

Limitations:

1. **Interpretability:** Random Forest models are complex ensembles, making it hard to interpret the full decision process.
2. **Computational cost:** Training many trees with GridSearch can be resource-intensive.
3. **Sampling:** The model was trained on a subset of data, which might limit capturing all patterns in the full dataset.
4. **Feature engineering:** Further feature extraction or selection might improve model performance.

Code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.tree import plot_tree+
# Load dataset
df = pd.read_csv('cardio_train.csv.zip', sep=';')
# Sample and clean
df = df.sample(n=20000, random_state=42)
df = df[(df['ap_hi'] > 80) & (df['ap_hi'] < 200)]
df = df[(df['ap_lo'] > 50) & (df['ap_lo'] < 150)]
# Feature engineering
df['bmi'] = df['weight'] / ((df['height'] / 100) ** 2)
df['age_years'] = df['age'] // 365
df['pulse_pressure'] = df['ap_hi'] - df['ap_lo']
# Categorize some features safely (fill NaN with 0 before converting)
df['age_category'] = pd.cut(df['age_years'], bins=[0, 40, 50, 60, 70], labels=[0, 1, 2, 3], include_lowest=True)
df['age_category'] = df['age_category'].fillna(0).astype(int)
df['bmi_category'] = pd.cut(df['bmi'], bins=[0, 18.5, 25, 30, 100], labels=[0, 1, 2, 3], include_lowest=True)
df['bmi_category'] = df['bmi_category'].fillna(0).astype(int)
df['pressure_category'] = pd.cut(df['pulse_pressure'], bins=[0, 30, 50, 70, 200], labels=[0, 1, 2, 3], include_lowest=True)
```

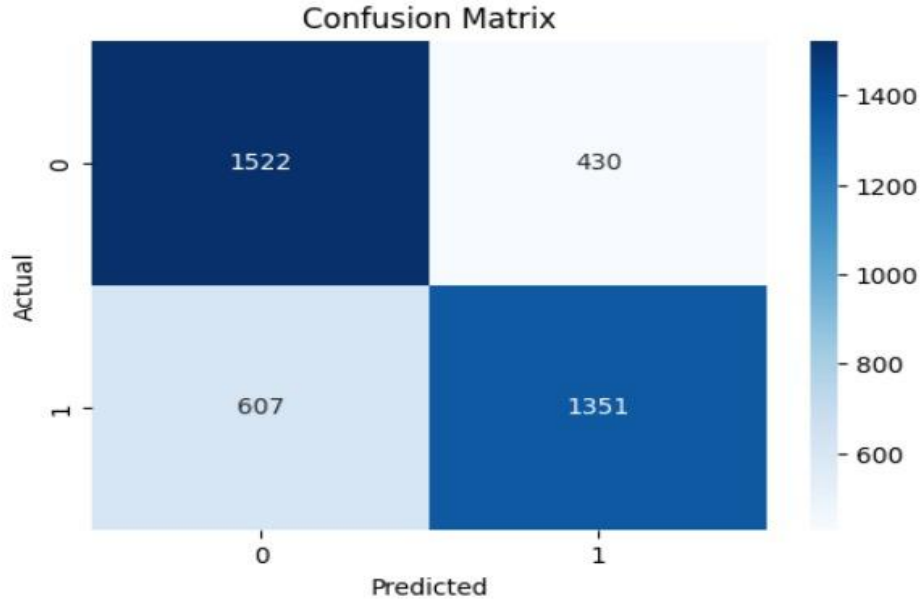
```
df['pressure_category'] = df['pressure_category'].fillna(0).astype(int)


# Prepare features and target
X = df.drop(['id', 'cardio', 'age', 'height', 'weight'], axis=1)
y = df['cardio']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)
# Grid search for hyperparameters with expanded param grid for better tuning
param_grid = {
    'n_estimators': [150, 200, 250],
    'max_depth': [15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}
grid = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    n_jobs=-1,
```

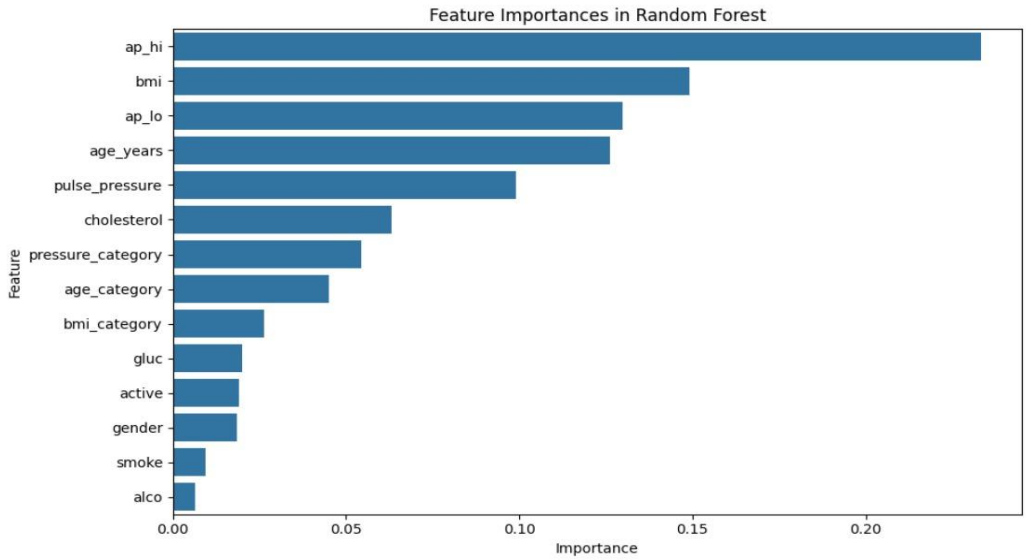
```
    verbose=2
)
grid.fit(X_train_scaled, y_train)
# Best model from grid search
best_rf = grid.best_estimator_
# Predictions
y_pred = best_rf.predict(X_test_scaled)
# Evaluation
print("\nBest Parameters:", grid.best_params_)
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Feature Importance
importances = best_rf.feature_importances_
features = X.columns
indices = np.argsort(importance s)[::-1]
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=importances[indices], y=features[indices])
plt.title("Feature Importances in Random Forest")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
# Number of trees in the forest
print(f"Number of trees in the best Random Forest: {best_rf.n_estimators}")
# Visualize two trees from the Random Forest
for i in range(2):
    plt.figure(figsize=(20, 10))
    tree = best_rf.estimators_[i]
    plot_tree(tree, feature_names=X.columns, class_names=["No Disease", "Has Disease"],
              filled=True, rounded=True, max_depth=3)
    plt.title(f"Visualization of Decision Tree {i+1} from the Random Forest")
    plt.show()
```

Output

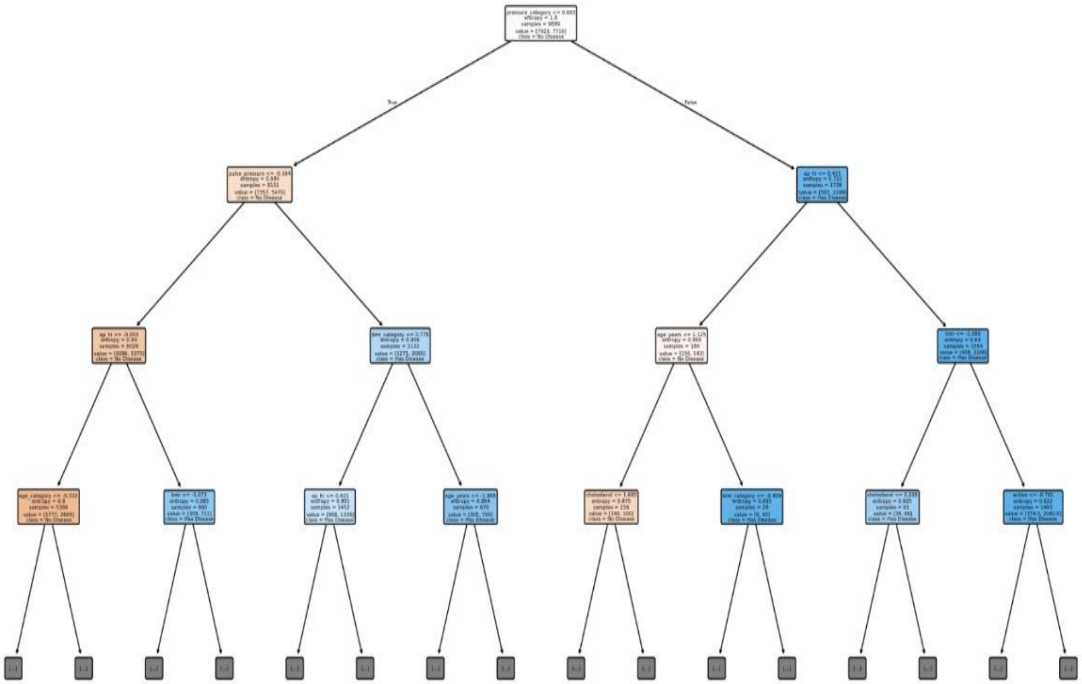
	precision	recall	f1-score	support
0	0.71	0.78	0.75	1952
1	0.76	0.69	0.72	1958
accuracy			0.73	3910
macro avg	0.74	0.73	0.73	3910
weighted avg	0.74	0.73	0.73	3910



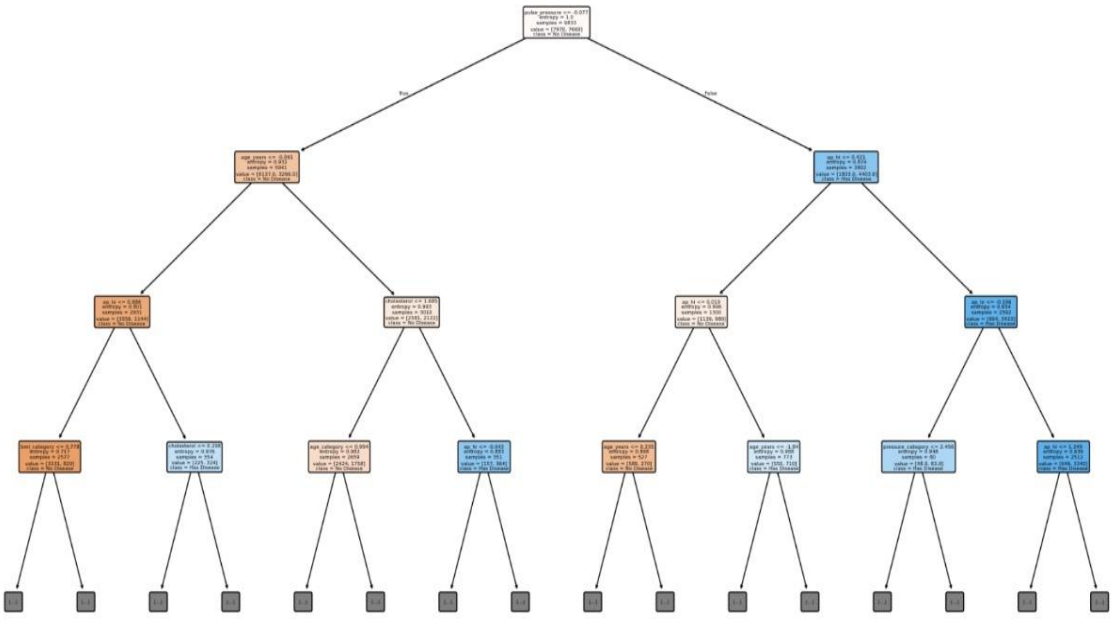


Number of trees in the best Random Forest: 250

Visualization of Decision Tree 1 from the Random Forest



Visualization of Decision Tree 2 from the Random Forest



Support Vector Machine Algorithm

Introduction

Support Vector Machine (SVM) is a supervised machine learning algorithm primarily used for classification tasks. It aims to find the optimal hyperplane that best separates classes in the feature space by maximizing the margin between data points of different classes. SVM is powerful for both linear and nonlinear classification by using kernel functions to transform data into higher dimensions.

How SVM Works?

1. SVM identifies a decision boundary (hyperplane) that maximizes the margin—the distance between the closest points (support vectors) of each class.
2. For linearly separable data, it finds a straight line (in 2D) or hyperplane (in higher dimensions).
3. For non-linearly separable data, it uses kernel functions (e.g., RBF, polynomial) to map data into a higher-dimensional space where a linear separator is possible.
4. The regularization parameter C controls the trade-off between maximizing margin and minimizing classification error.
5. The kernel parameter gamma defines how far the influence of a single training example reaches.

Why Use SVM?

1. Effective in high-dimensional spaces, especially when the number of features exceeds the number of samples.
2. Robust to overfitting with proper kernel choice and regularization.
3. Works well when classes are separable with a clear margin.
4. Can handle both linear and complex nonlinear classification problems through kernel trick.
5. Generally, achieves good accuracy and is widely used for biomedical classification problems like cardiovascular disease prediction.

Dataset Description

1. Dataset contains patient data related to cardiovascular health, including:
2. Demographics: Age (in days), height, weight
3. Clinical measurements: Systolic (ap_hi) and diastolic (ap_lo) blood pressure
4. Target variable: cardio (0 = no cardiovascular disease, 1 = disease present)

The dataset is large and diverse, but code samples 20,000 rows and filters out unrealistic blood pressure values for quality.

Data Preprocessing

1. **Sampling:** Randomly selected 20,000 records for faster experimentation.
2. **Filtering:** Removed outliers in blood pressure (ap_hi between 80-200, ap_lo between 50-150) to reduce noise.
3. **Feature Engineering:** BMI computed as weight divided by squared height in meters. Age converted from days to years. Pulse pressure calculated as the difference between systolic and diastolic BP.
4. **Scaling:** Standardized features using StandardScaler to normalize their ranges — critical for SVM performance.

Model Building

1. **Dataset Preparation:**
   - Loaded the cardiovascular dataset and randomly sampled 20,000 records to make training faster.
   - Removed outliers in blood pressure (ap_hi and ap_lo) to ensure clean and realistic data.
2. **Feature Engineering:**
   - Created new features to enrich the dataset:
     - BMI: Body Mass Index derived from height and weight.
     - Age in years: converted from days.
     - Pulse pressure: difference between systolic and diastolic pressure.
3. **Feature and Target Selection:**

- Dropped non-essential columns (id, age, height, weight) from features.
- Defined X as the input features and y as the target variable (cardio).

4. **Train-Test Split:**
   - Split the dataset into 80% training and 20% testing sets to evaluate performance.
5. **Feature Scaling:**
   - Standardized the features using StandardScaler.
   - Scaling is essential for SVM as it relies on distance-based calculations.
6. **Model Tuning with GridSearchCV:**
   - Used GridSearchCV to find the best hyperparameters:
     - C: Regularization strength.
     - kernel: Linear or RBF kernel.
     - gamma: Kernel coefficient.
   - Performed 5-fold cross-validation to select the best parameter combination.
7. **Training and Prediction:**
   - Trained the SVM model with the best-found parameters.
   - Made predictions on the scaled test data.
8. **Model Evaluation:**
   - Evaluated the model using:
     - Accuracy: percentage of correct predictions.
     - Classification Report: includes precision, recall, F1-score.
     - Confusion Matrix: to visualize correct and incorrect classifications.
9. **Visualization with PCA:**
   - Applied Principal Component Analysis (PCA) to reduce features to 2D for visualization.
   - Plotted the predicted classes in 2D space to observe how well the SVM separates the classes.

Model Performance and Visualization

1. **Accuracy:** Reports test accuracy to assess how well the model generalizes.
2. **Classification Report:** Provides precision, recall, and F1-score for each class, highlighting model strengths and weaknesses.

3. **Confusion Matrix:** Visualizes true vs predicted labels, helping identify types of errors (false positives/negatives).
4. **PCA Visualization:** Applied Principal Component Analysis (PCA) to reduce feature dimensions to 2D for visualization. Plotted test samples colored by predicted class to visualize decision boundaries and class separation in reduced space.

Conclusion and Limitations

Conclusion: The SVM model, after hyperparameter tuning, effectively classifies cardiovascular disease risk using clinical and demographic features. The model benefits from good feature engineering and proper scaling.

Limitations:

1. SVMs can be computationally intensive for very large datasets.
2. Kernel choice and hyperparameter tuning significantly affect performance.
3. PCA visualization is an approximation and may not perfectly represent complex multidimensional decision boundaries.
4. Model interpretability is limited compared to simpler models (e.g., decision trees).
5. Sampling a subset of data may exclude useful patterns present in the full dataset.

Code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

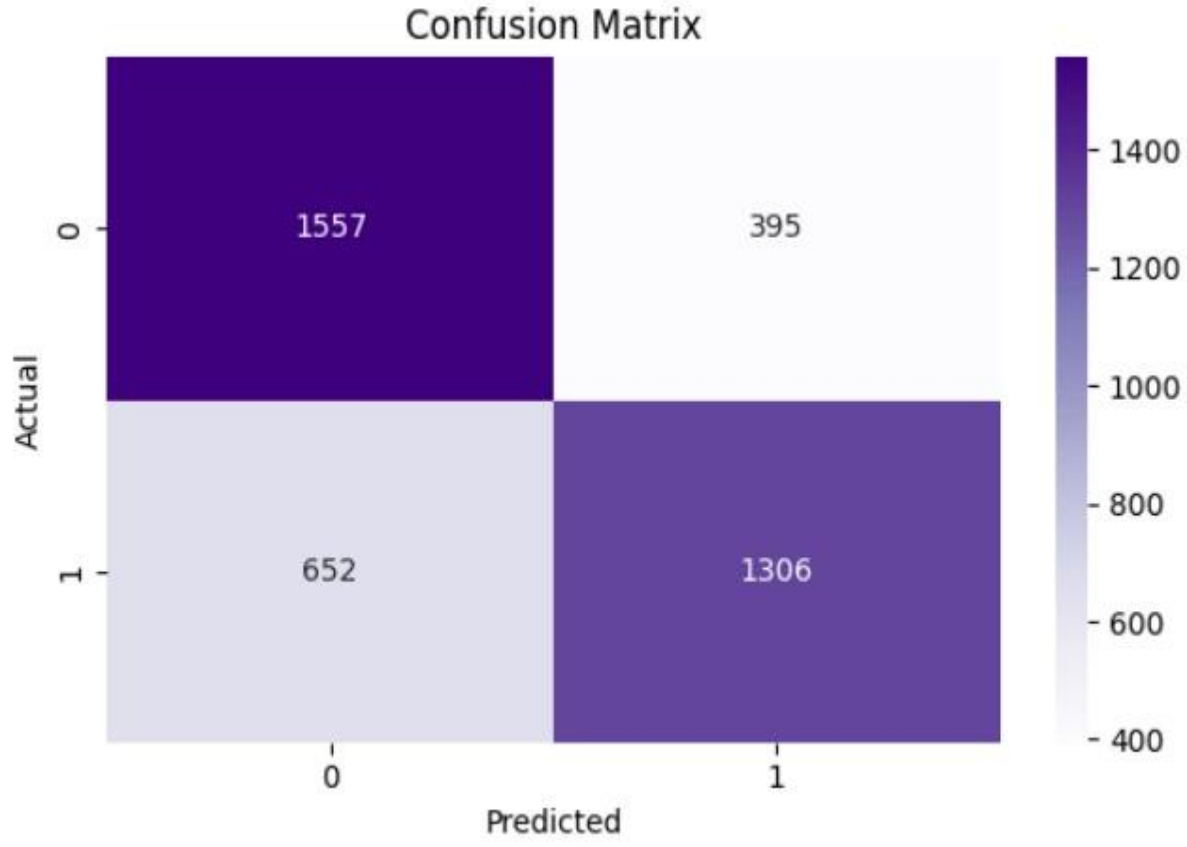
```
from sklearn.decomposition import PCA

# Load and sample data

df = pd.read_csv('cardio_train.csv.zip', sep=';')

df = df.sample(n=20000, random_state=42)

df = df[(df['ap_hi'] > 80) & (df['ap_hi'] < 200)]

df = df[(df['ap_lo'] > 50) & (df['ap_lo'] < 150)]

# Feature engineering

df['bmi'] = df['weight'] / ((df['height'] / 100) ** 2)

df['age_years'] = df['age'] // 365

df['pulse_pressure'] = df['ap_hi'] - df['ap_lo']

# Optional categorical features (if needed later)

# df['age_category'] = pd.cut(df['age_years'], bins=[0, 40, 50, 60, 70], labels=[0, 1, 2, 3]).astype(int)

# df['bmi_category'] = pd.cut(df['bmi'], bins=[0, 18.5, 25, 30, 100], labels=[0, 1, 2, 3]).astype(int)

# Define features and target

X = df.drop(['id', 'cardio', 'age', 'height', 'weight'], axis=1)

y = df['cardio']

# Train/test split

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

# Scaling

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# SVM with GridSearchCV
```

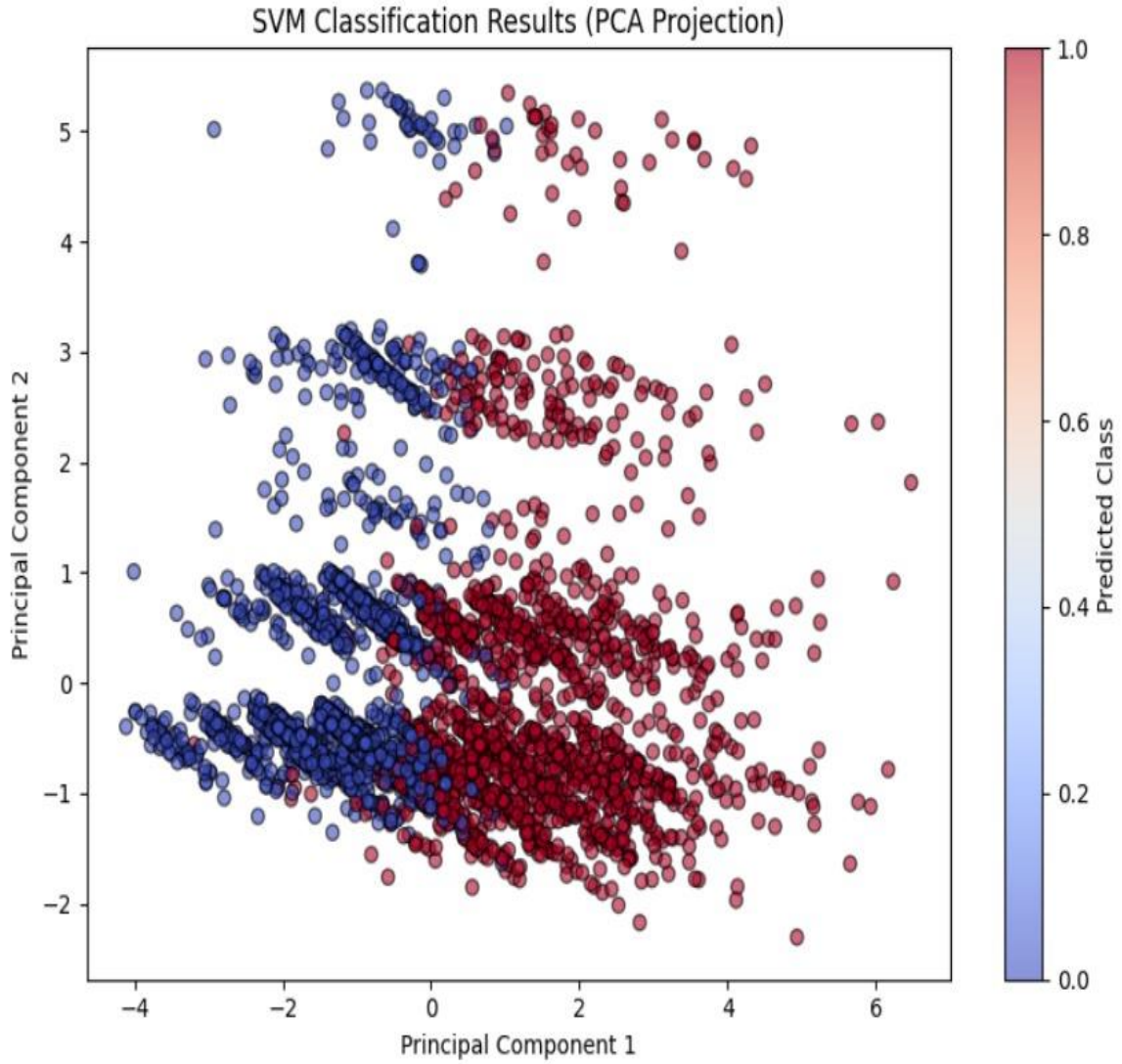
```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
grid = GridSearchCV(SVC(), param_grid, cv=5, n_jobs=-1, verbose=1)
grid.fit(X_train_scaled, y_train)
# Best model
best_svm = grid.best_estimator_
y_pred = best_svm.predict(X_test_scaled)
# Output
print("\nBest Parameters:", grid.best_params_)
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Purples')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
# PCA for visualization
pca = PCA(n_components=2)
```

```
X_vis = pca.fit_transform(X_test_scaled)

plt.figure(figsize=(8, 6))

scatter = plt.scatter(X_vis[:, 0], X_vis[:, 1], c=y_pred, cmap='coolwarm', alpha=0.6, edgecolor='k')

plt.title("SVM Classification Results (PCA Projection)")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.colorbar(scatter, label='Predicted Class')

plt.tight_layout()

plt.show()
```

Output

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

Best Parameters: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
Accuracy: 0.7322

Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.80      0.75      1952
           1       0.77      0.67      0.71      1958

    accuracy                           0.73      3910
   macro avg       0.74      0.73      0.73      3910
weighted avg       0.74      0.73      0.73      3910
```




SVM Classification Results (PCA Projection)
Principal Component 2
Principal Component 1
Predicted Class
5
4
3
2
1
0
−1
−2
−4
−2
0
2
4
6
1.0
0.8
0.6
0.4
0.2
0.0

Logistic Regression

Introduction

Logistic Regression is a supervised machine learning algorithm used mainly for binary classification tasks — that is, when you want to classify data into two groups, like "disease" vs "no disease." Unlike linear regression which predicts continuous values, logistic regression predicts the probability that a given input belongs to a particular category. It then assigns the class label based on a probability threshold (usually 0.5).

How It Works?

1. Logistic Regression applies the logistic (sigmoid) function to a linear combination of input features, converting any real-valued number into a probability between 0 and 1.
2. The model predicts class 1 if the probability exceeds a certain threshold (commonly 0.5), otherwise class 0.
3. It estimates parameters (weights) by maximizing the likelihood of observing the training data using maximum likelihood estimation.
4. The model's output is interpretable as the log-odds or probability of the positive class.

Why Use Logistic Regression?

1. **Interpretability:** Logistic regression coefficients tell you how each feature affects the odds of having the disease (positive or negative impact).
2. **Efficiency:** It's computationally cheap and fast to train on large datasets.
3. **Probabilistic outputs:** Gives you probabilities, which allow for nuanced decisions and evaluation with ROC/AUC curves.
4. **Baseline performance:** Often used as a starting point before trying more complex models.
5. **Works well with linearly separable data:** If your features can separate classes by a linear boundary in transformed space, logistic regression can perform very well.

Dataset Description

1. The dataset contains clinical records related to cardiovascular health.
2. Features include:
    - Blood pressure (systolic ap_hi and diastolic ap_lo)
    - Height and weight
    - Age (given in days, converted to years)
    - Target variable cardio: 0 for no cardiovascular disease, 1 for disease.
    - Code samples 20,000 random rows for manageable processing.
3. It filters out extreme blood pressure values that are likely errors or outliers to improve data quality.

Data Preprocessing

1. **Sampling:** To reduce computation and training time, you randomly select 20,000 rows.
2. **Filtering:** Blood pressure values outside reasonable medical ranges are excluded to avoid noise.
3. **Feature Engineering:**
    - **BMI:** Calculated using the formula weight/(height/100)2\text{weight} / (\text{height}/100)^2weight/(height/100)2, which indicates body fatness.
    - **Age in years:** Since original age is in days, dividing by 365 makes it interpretable.
    - **Pulse pressure:** Difference between systolic and diastolic BP; higher values can indicate arterial stiffness and cardiovascular risk.
4. **Scaling:** StandardScaler is applied so each feature has zero mean and unit variance, which helps logistic regression converge faster and prevents features with large scales dominating the model.

Model Building

1. **Dataset Preparation:**
   - Loaded the cardiovascular disease dataset and randomly sampled 20,000 rows to optimize runtime.
   - Cleaned the dataset by removing outliers in systolic (ap_hi) and diastolic (ap_lo) blood pressure to ensure data quality.
2. **Feature Engineering:**
   - Derived additional informative features:
     - BMI (Body Mass Index): calculated from height and weight.
     - Age in years: transformed from age in days.
     - Pulse pressure: calculated as the difference between systolic and diastolic BP.
3. **Feature and Target Selection:**
   - Dropped unnecessary features like id, age, height, and weight.
   - Defined X as the input features and y as the target variable (cardio).
4. **Train-Test Split:**
   - Split the dataset into 80% training and 20% testing sets for model training and evaluation.
5. **Feature Scaling:**
   - Standardized the features using StandardScaler.
   - Scaling is important for logistic regression as it ensures faster convergence and balanced coefficient estimation.
6. **Model Training:**
   - Initialized and trained a Logistic Regression model with increased maximum iterations (max_iter=1000) to ensure convergence.
   - Used the scaled training data to fit the model.
7. **Prediction and Evaluation:**
   - Made predictions on the test set.
   - Evaluated model performance using:
     - Accuracy: overall prediction correctness.
     - Classification Report: precision, recall, F1-score for both classes.
     - Confusion Matrix: visual breakdown of correct and incorrect predictions.

8. **Probability Prediction and ROC Curve:**
    - Generated predicted probabilities for the positive class.
    - Plotted the ROC Curve to visualize the trade-off between true positive and false positive rates.
    - Calculated AUC (Area Under Curve) to summarize model performance — higher AUC indicates better discrimination between classes.

Model Performance and Visualization

1. **Accuracy:** Percentage of correct predictions on test data. It's a simple overall performance metric but can be misleading if classes are imbalanced.
2. **Classification report:** Provides detailed metrics like:
    - **Precision:** How many predicted positives are actually positive (low false positives).
    - **Recall (Sensitivity):** How many actual positives were correctly identified (low false negatives).
    - **F1-score:** Harmonic mean of precision and recall, balances the two.
3. **Confusion Matrix:** Visualizes True Positives, True Negatives, False Positives, and False Negatives. Helpful to understand types of errors the model makes.
4. **ROC Curve:** Plots the trade-off between True Positive Rate (sensitivity) and False Positive Rate as classification threshold varies.
5. **AUC (Area Under Curve):** Scalar summary of ROC; values closer to 1 indicate strong classification ability. This helps assess the model beyond a fixed threshold.

Conclusion and Limitations

Conclusion: Logistic regression effectively classifies cardiovascular disease based on key clinical features. It's simple, fast, and produces interpretable results, making it useful in healthcare scenarios.

Limitations:

1. It assumes a linear relationship between features and log-odds, so complex nonlinear relationships might not be captured.
2. Sensitive to outliers and multicollinearity among features — requires good preprocessing and feature selection.
3. May underperform compared to more complex algorithms (SVM, Random Forest, Neural Networks) on very complex datasets.
4. Does not automatically handle interactions between features unless explicitly added.

Code

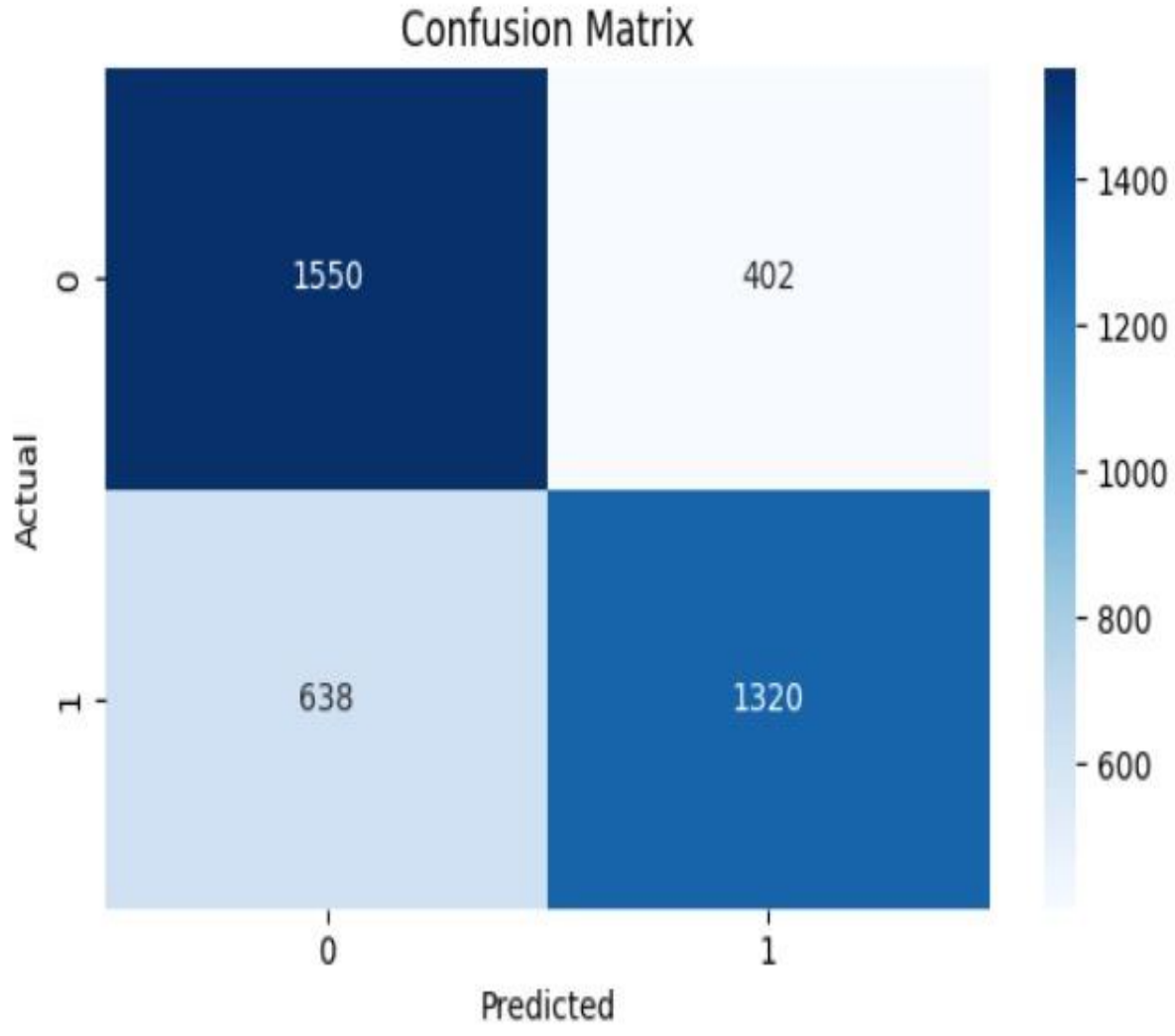
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc

# Load dataset

df = pd.read_csv('cardio_train.csv.zip', sep=';')

# Sample and clean

df = df.sample(n=20000, random_state=42)

df = df[(df['ap_hi'] > 80) & (df['ap_hi'] < 200)]

df = df[(df['ap_lo'] > 50) & (df['ap_lo'] < 150)]

# Feature engineering
```

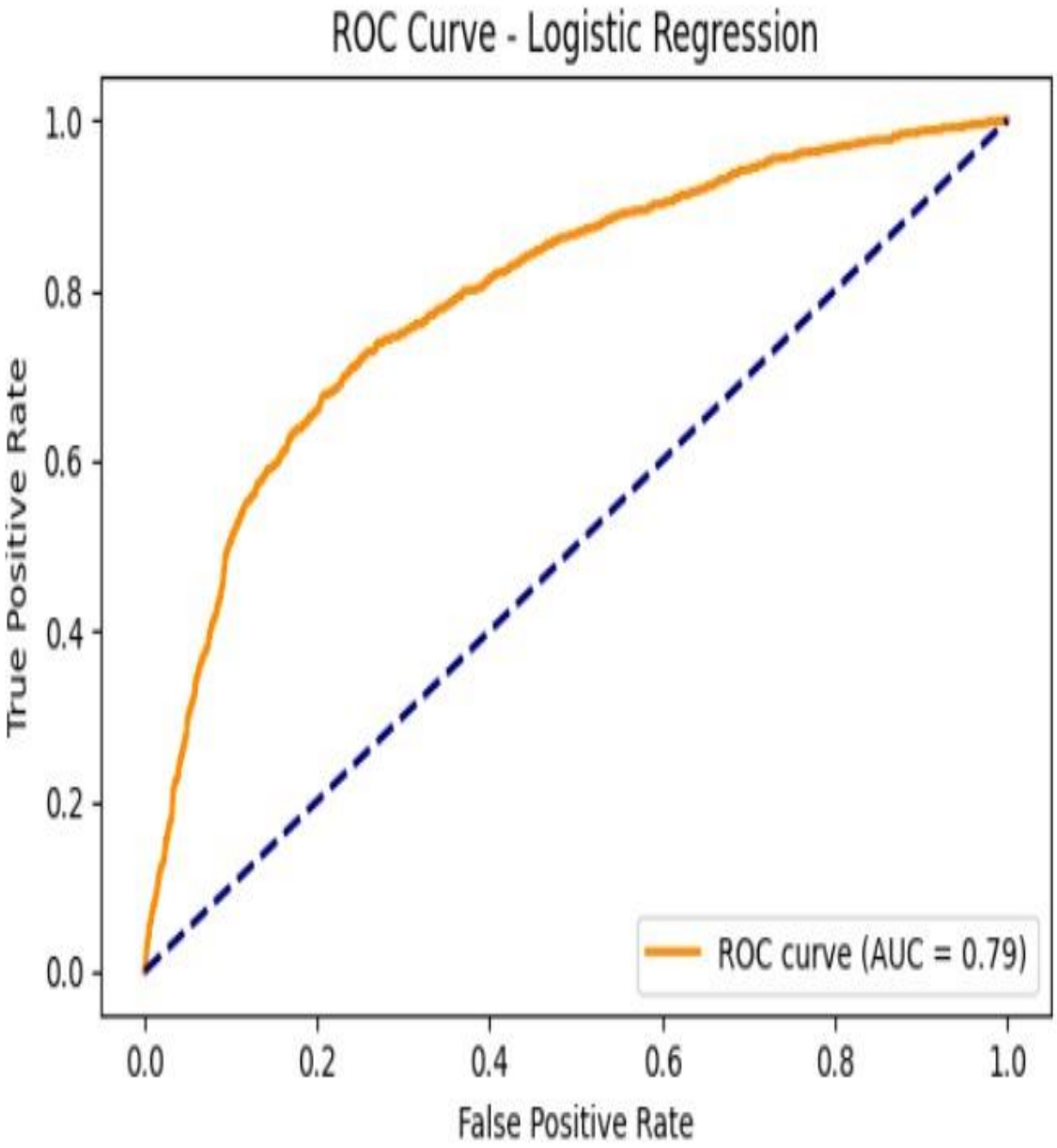
```
df['bmi'] = df['weight'] / ((df['height'] / 100) ** 2)

df['age_years'] = df['age'] // 365

df['pulse_pressure'] = df['ap_hi'] - df['ap_lo']

# Select features and target

X = df.drop(['id', 'cardio', 'age', 'height', 'weight'], axis=1)

y = df['cardio']

# Split and scale

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Train model

lr = LogisticRegression(max_iter=1000, random_state=42)

lr.fit(X_train_scaled, y_train)

# Predictions

y_pred = lr.predict(X_test_scaled)

y_proba = lr.predict_proba(X_test_scaled)[:, 1]

# Evaluation

print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
```

```
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title("ROC Curve - Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

Output

```
Accuracy: 0.734

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.79      0.75      1952
           1       0.77      0.67      0.72      1958

    accuracy                           0.73      3910
   macro avg       0.74      0.73      0.73      3910
weighted avg       0.74      0.73      0.73      3910
```




ROC Curve - Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.79)