BRAC UNIVERSITY

Heart Disease Prediction

Artificial Intelligence (CSE422)

Submitted by:

Student 1:

Name: Mir Sajin Mahmud

ID: 21301247

Student 2:

Name : Marzia Khanam

ID: 21201571

Submitted to:

Hasnat Jamil Bhuiyan

&

Farhan Faruk

Section: 06

Fall 2024

Table of contents:

Introduction

The project addresses a binary classification problem by analyzing a dataset, preprocessing it, and applying machine learning models. The primary objective is to predict outcomes based on specific features, ensuring data quality and effective model training. This study focuses on predicting the likelihood of heart disease, which has become increasingly common in recent years. By identifying the potential risk of heart disease, individuals can take proactive steps to reduce their chances of developing the condition. This motivation emphasizes the importance of our project, as it aims to contribute to the early detection and prevention of heart disease through data-driven insights.

Dataset Description

The dataset used for this project was sourced from Kaggle. As per the project requirements, a dataset with more than 30,000 entries was necessary. Consequently, this dataset was selected as it contains over 70,000 data points. Although the larger dataset size slightly reduced the model's accuracy, it provided a comprehensive foundation for analysis and training.

- **Source**: Kaggle
- **Link**: Click here for the dataset link

The dataset comprises **91,000 data points** and contains **13 features**. Excluding the `id` column and the target variable (`cardio`), there are **11 input features**. A detailed description of these features is provided below:

Feature Name	Feature Type	Data Type	Description
Age	Objective Feature	Integer (days)	represents the age of the individual in days.
Height	Objective Feature	Integer (cm)	Represents the height of the individual in centimeters.
Weight	Objective Feature	Float (kg)	Represents the weight of the individual in kilograms.
Gender	Objective Feature	Categorical (represented as binary)	Represents the gender of the individual (encoded as a

			categorical variable).
Systolic Blood Pressure	Examination Feature	Integer	Represents the systolic blood pressure (`ap_hi`).
Diastolic Blood Pressure	Examination Feature	Integer	Represents the diastolic blood pressure (`ap_lo`).
Cholesterol	Examination Feature	Object (Categorical)	Indicates cholesterol levels (normal, above normal, well above normal).
Glucose	Examination Feature	Object (Categorical)	Indicates glucose levels (normal, above normal, well above normal).
Smoking	Subjective Feature	Binary	Indicates whether the individual is a smoker (1: Yes, 0: No).
Alcohol Intake	Subjective Feature	Binary	Indicates alcohol consumption (1: Yes, 0: No).
Physical Activity	Subjective Feature	Binary	Indicates engagement in physical activity (1: Yes, 0: No).
Presence of Cardiovascular Disease	Target Variable	Binary	Represents the presence (`1`) or absence (`0`) of cardiovascular disease.

The dataset is balanced, with 34,964 instances where cardiovascular disease is absent (`cardio = 0`) and 34,932 instances where it is present (`cardio = 1`). This balance ensures fairness in model training and evaluation.

Here is the code and output of data types:

```
df.dtypes
```

	0
id	int64
age	float64
gender	float64
height	float64
weight	float64
ap_hi	float64
ap_lo	float64
cholesterol	object
gluc	object
smoke	float64
alco	float64
active	float64
cardio	float64

dtype: object

Here is the Bar chart of the balanced Dataset:

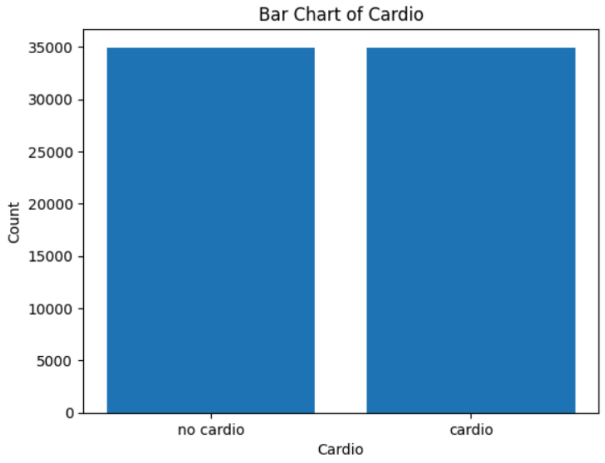
Code:

```
[11] # Count the values in the "cardio" column
     cardio_counts = df["cardio"].value_counts()

     # Define labels and values
     labels = ["no cardio", "cardio"]
     values = [cardio_counts[0], cardio_counts[1]]

     # Plot the bar chart
     plt.bar(labels, values)
     plt.xlabel("Cardio")
     plt.ylabel("Count")
     plt.title("Bar Chart of Cardio")
     plt.show()
```

Bar chart:



Data pre-processing

The dataset contained both null values and categorical data, which required preprocessing to ensure compatibility with machine learning models that primarily operate on numerical inputs. Specifically, the `cholesterol` and `gluc` columns contained categorical values (`normal`, `above normal`, and `well above normal`) that needed to be encoded into numeric representations. Below is a detailed explanation of the preprocessing steps:

1. Handling Null Values

The dataset was analyzed for missing values, revealing 1,001 rows with null entries, which accounted for 1.43% of the total data. Given the small proportion of missing data, these rows were removed to maintain data integrity while minimizing the loss of information. This approach ensured that the overall structure and distribution of the dataset remained unaffected. Here is the code of removing Null values:

```
df_cleaned = df.dropna()
# Calculate the total number of rows after removal
total_rows_after = len(df_cleaned)
# Display results
print(f"Total Rows Before Removal: {total_rows}")
print(f"Total Rows After Removal: {total_rows_after}")
print(f"Rows Removed: {total_rows - total_rows_after}")
```

```
Total Rows Before Removal: 70000
Total Rows After Removal: 68999
Rows Removed: 1001
```

2. Encoding Categorical Values

The columns cholesterol and gluc contained categorical values. To solve this we take these steps:

- Normal is represented as 1
- Above Normal is represented as 2
- Well Above Normal is represented as 3

```
# Define mappings for cholesterol and gluc
mapping = {
    'normal': 1,
    'above normal': 2,
    'well above normal': 3
}

# Apply the mapping to the 'cholesterol' column
df_cleaned['cholesterol'] = df_cleaned['cholesterol'].map(mapping)

# Apply the mapping to the 'gluc' column
df_cleaned['gluc'] = df_cleaned['gluc'].map(mapping)

# Display the updated DataFrame
print("\nUpdated DataFrame:")
print(df_cleaned.head())
```

This encoding transformed the categorical variables into numeric form, allowing the machine learning models to process these features effectively.

These preprocessing steps ensured that the dataset was clean, consistent, and suitable for subsequent analysis and model training.

Feature Scaling

The dataset exhibited high variance in certain features such as age, height, weight, ap_hi, and ap_lo, compared to other features. This discrepancy in scale could lead to a machine learning model assigning disproportionate importance to features with larger ranges, potentially biasing the predictions.

To address this issue, feature scaling was applied using the sklearn.preprocessing module. This process normalized the variance across all features, ensuring that each feature contributed almost equally to the model's training process. By standardizing the dataset, the scaled features enabled the machine learning algorithms to perform optimally without favoring any specific feature due to its scale.

Model Training and Test

For this project, six machine learning classifiers were trained and tested to evaluate their effectiveness in predicting the likelihood of cardiovascular disease. The dataset was split into training (70%) and testing (30%) sets using the train_test_split function from sklearn, with stratification applied to maintain class balance. Feature scaling was performed using StandardScaler for models that require normalized data (e.g., Logistic Regression, KNN, and SVC).

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score as acc
from sklearn.preprocessing import StandardScaler

# Split the data
x = df_n.drop(columns="cardio")
y = df_n["cardio"]
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3, random_state=1, stratify=y)

# Scale the data (for Logistic Regression, KNN, SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
# Define classifiers
classifiers = [
    LogisticRegression(),
    GaussianNB(),
    RandomForestClassifier(),
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    SVC(probability=True)  # Support Vector Classifier
]
clf_accuracies = {}

# Train and evaluate each classifier
for model in classifiers:
    if isinstance(model, GaussianNB):  # GaussianNB doesn't require scaling
        model.fit(X_train, Y_train)
        y_pred = model.predict(X_test)
    else:
        model.fit(X_train_scaled, Y_train)  # Train the model on scaled data
        y_pred = model.predict(X_test_scaled)  # Predict on scaled test data

    accuracy = acc(Y_test, y_pred)  # Calculate accuracy
    clf_accuracies[type(model).__name__] = round(accuracy, 2)  # Store the accuracy

# Print the accuracies
print("Classifier Accuracies:")
for clf, acc_value in clf_accuracies.items():
    print(f"{clf}: {acc_value}")
```

Results:

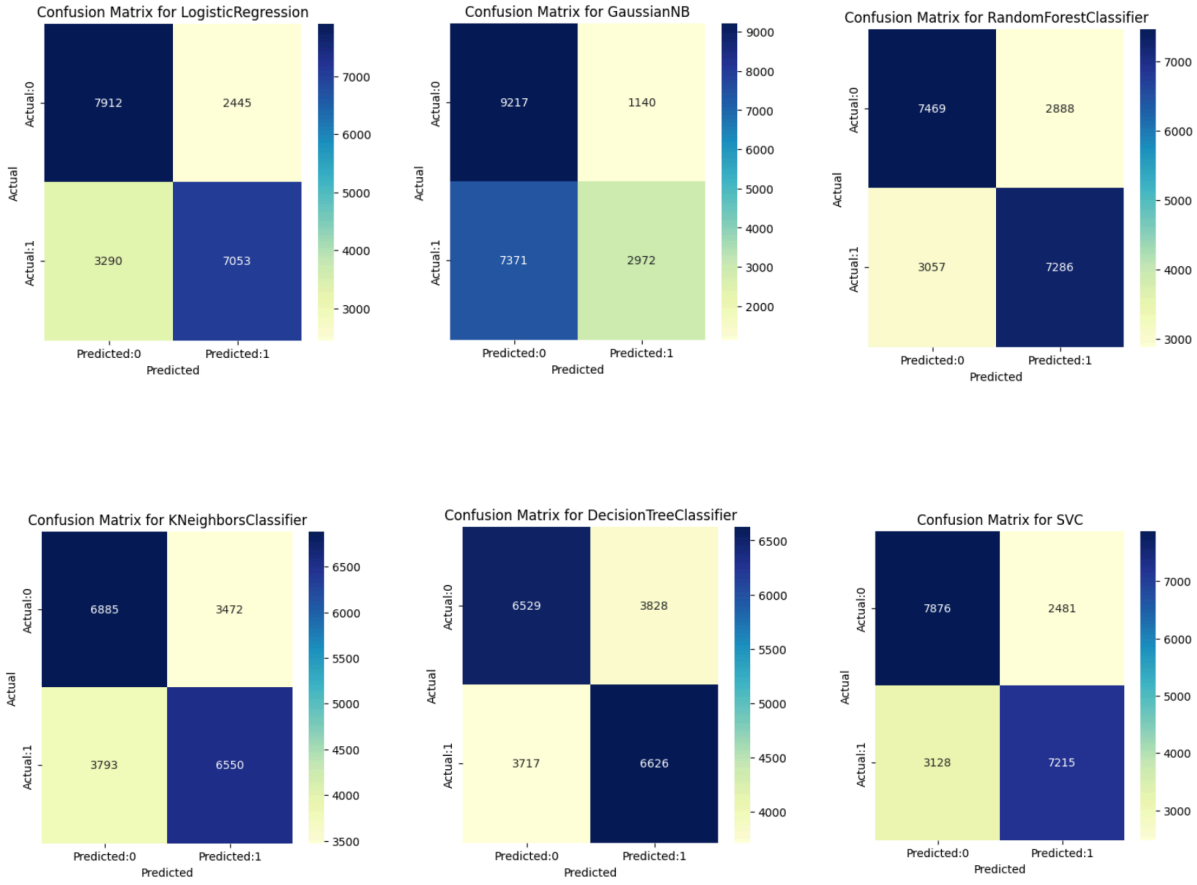
```
Classifier Accuracies:
LogisticRegression: 0.72
GaussianNB: 0.59
RandomForestClassifier: 0.71
KNeighborsClassifier: 0.65
DecisionTreeClassifier: 0.64
SVC: 0.73
```

As **Gaussian Naive Bayes** is suitable for small Datasets, it shows the lowest accuracy.

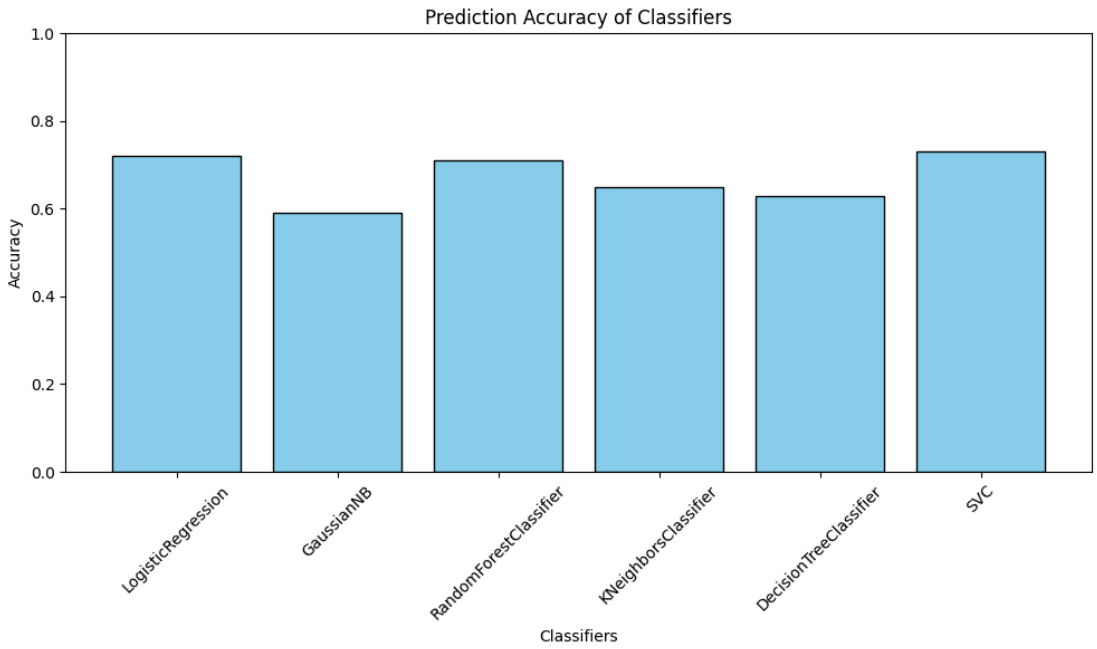
Confusion Matrix Analysis

To evaluate the predictive performance further, confusion matrices were generated for each classifier. These matrices provided insights into the number of true positives, true negatives, false positives, and false negatives, helping assess the models' precision and recall. Heatmaps were plotted for each confusion matrix for better visualization.

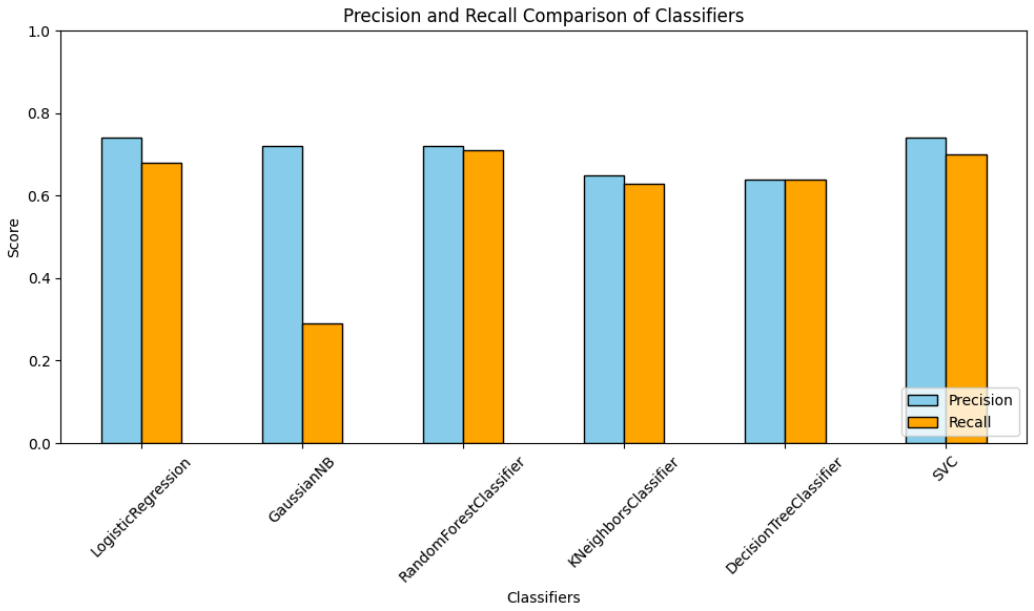
Here is the chart of the confusion matrices:



Bar chart of the prediction accuracy of each model:



Precision and recall comparison :



The evaluation of classifiers using Precision and Recall revealed distinct performance differences. Random Forest and SVC emerged as the best-performing models, achieving high and balanced Precision and Recall, indicating their ability to accurately identify positive cases while minimizing false negatives. Logistic Regression also performed well, showing consistent results. In contrast, Gaussian Naive Bayes had lower Recall, suggesting difficulty in capturing all positive cases. K-Nearest Neighbors and Decision Tree classifiers delivered balanced but slightly lower scores compared to Random Forest and SVC. Overall, Random Forest is recommended for its superior performance, making it ideal for applications requiring both exactness and completeness.

Conclusion

This project successfully applied machine learning models to predict the likelihood of heart disease using a comprehensive dataset with over 70,000 entries. Through careful preprocessing, including handling null values, encoding categorical variables, and applying feature scaling, the dataset was prepared for optimal model performance. Six classifiers were trained and evaluated, with Random Forest and SVC emerging as the most effective models, achieving high precision and recall. These results highlight the importance of robust preprocessing and model selection in achieving accurate predictions. The insights from this project underscore the potential of machine learning in the early detection and prevention of heart disease, offering valuable contributions to the healthcare domain. Future work could focus on integrating additional features, refining hyperparameters, and exploring ensemble techniques to further enhance model performance.