Faculty of Engineering & Technology

Electrical & Computer Engineering Department

MACHINE LEARNING AND DATA SCIENCE

ENCS5341

Assigment_3

"Predicting Heart Disease Using Classification Techniques"

Prepared by: Salwa Fayyad 1200430

Sahar Fayyad 1190119

Instructor: Dr. Yazan Abu Farha

Section: 2

Date: 25.1.2024

Table of Contents

Tables of figures:

Introduction

In this project, we aim to explore and analyze a comprehensive dataset that predicts heart disease using classification techniques. The dataset involves various clinical and demographic features: Age, Sex, Chest Pain Type, Resting Blood Pressure (RestingBP), Cholesterol level, etc, and the presence of Heart Disease (target).

The dataset comprises diverse patient profiles, ranging from a 36-year-old male with atypical angina (heart attack) and a normal cholesterol level to a 60-year-old with asymptomatic chest pain and high cholesterol. These attributes are essential in understanding the patterns and risk factors associated with heart disease.

In the initial phase of our analysis, we implemented a nearest-neighbor baseline model to understand the basic patterns within the data. This model, chosen for its simplicity and effectiveness in capturing the nearest similarities in multidimensional data, was evaluated with both k=1 and k=3 to observe the variation in performance.

In the second phase, we used two additional machine learning methods that uses in classification technique: Support Vector Machine (SVM) and Random Forest. These methods are good for sorting data into categories, like figuring out if someone has heart disease or not. SVM is great for working with complex data and finding patterns that aren't obvious. Random Forest is good for dealing with a lot of data without getting confused. We used both methods to see which is better at identifying the signs of heart disease. This helps doctors figure out who is at risk.

To evaluate the performance of these models, we used specific evaluation metrics like accuracy and recall. These metrics are essential for understanding how well our models work in predicting heart disease. In medical datasets, recall is an essential metric because it focuses on minimizing false negatives (FN), which are particularly dangerous in healthcare. High recall ensures that serious conditions like cancer or heart disease are not overlooked. Also, Accuracy tells us how often the models make correct predictions. Together, these metrics provide a comprehensive assessment of our models' performance in predicting heart disease.

Preprocessing dataset

The EDA (Exploratory Data Analysis) function provided is designed to perform a comprehensive analysis of a dataset, with a focus on both quantitative and qualitative factors such as mean, median for numeric and mode for non-numeric columns respectively. As shown in the figures below:

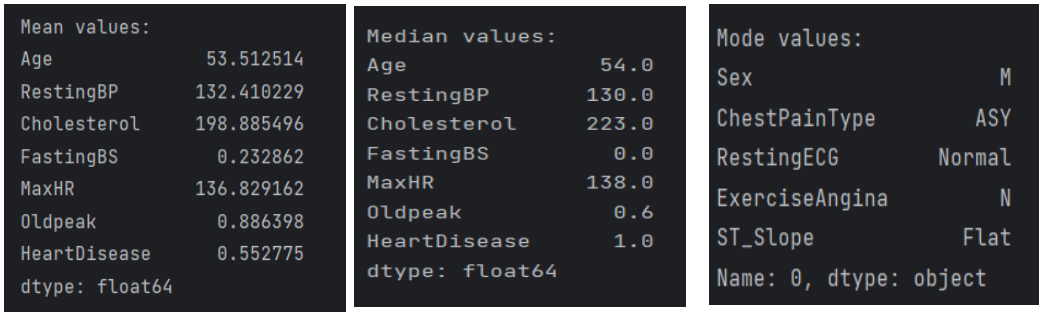


Figure 1: Quantitative measure

The function primarily targets datasets with a mix of numerical and categorical features related to heart disease and visualize those using histograms for numerical features and bar plots for categorical features. The first set of histograms explores the distribution of numeric features, and overlaying mean and median lines for additional insights into central tendency.

Figure 2: Histogram for each numeric values

The second set of bar plots focuses on non-numeric, particularly categorical, features with mode line:

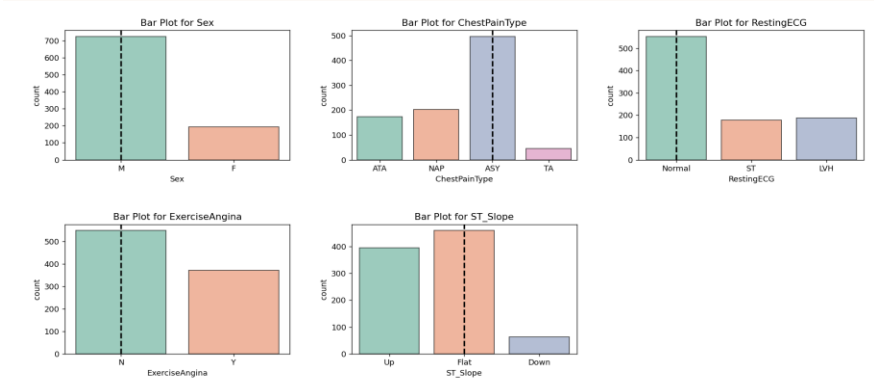


Figure 3: Bar plots for categorical values

After that, we check if there is a missing data or duplicate data, then we removing the duplicated data, and handling the missing data, if the data was non-numeric we replaced it with mode, and if the data was numeric data, then we check if the data have outliers or not, the distribution skewed or normal to know what is the best quantitative measure.

```
Before handling the data
isnull
 Age              0
Sex               2
ChestPainType     0
RestingBP         0
Cholesterol       2
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64

duplicated 1
```

```
After handling the data
isnull
 Age              0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64

duplicated 0
```

Figure 4: Before and after handling data

Typically, the mean is less susceptible to the presence of outliers (compared to the median), so we opted to fill the missing data using the median method in the numerical missing values in cholesterol.

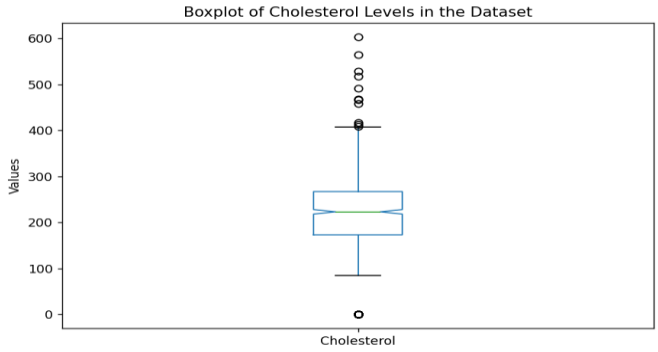


Figure 5: Boxplot of cholesterol levels in the dataset

Also, we dealt with the outliers in the data, after finding it, we remove it from the data, because it is consider as a noise.

```
outliers: (array([ 76, 109, 149, 166, 241, 324, 365, 390, 399, 449, 592, 616, 702,
       732, 759, 771, 791, 850, 900], dtype=int64), array([2, 1, 2, 5, 1, 5, 1, 4, 1, 1, 1, 2, 5, 1, 1, 5, 5, 5, 5],
      dtype=int64))
```

Figure 6: outliers

From the above figure, the first array represents the values that have been identified as outliers, furthermore the second array represent the count of occurrences for each corresponding value in the first array.

Together, these visualizations provide a thorough understanding of the dataset's characteristics, making the EDA process more insightful and interpretable.

Then we apply One-hot encoding which is typically used to handle categorical variables in machine learning. Consequently, each observation is marked with a 0 or 1 in these columns, indicating the presence or absence of a specific category.

Experiments and Results

In our project, we conducted a series of experiments to evaluate the performance of various models, including baseline models, model evaluation procedures, and selection of hyper-parameters.

In the main function and after preprocessing the data and handle it, we divide the data to training and testing data 20% testing and 80% taring, and random_state=42 for reproducibility and making the results consistent across different runs. This divided data is then used uniformly across the K-NN, Random Forest and SVM models.

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

Figure 7: split the data into training and testing in the main

➢ **Baseline Model** (K-Nearest Neighbors)

The KNN function facilitates k-Nearest Neighbors classification with flexible parameters such as the number of neighbors (k =1, k=3), and by using the Manhattan distance. After initializing and training the KNN classifier, it predicts labels for the testing data and calculates accuracy and recall scores.The result in both $k = 1$ and $k = 3$ for Manhattan distances is represented in the figure below:

```
-------------------------KNN---------------------------------
Results for k=1 using manhattan distance:
Accuracy: 0.7333
Recall: 0.8125
Results for k=3 using manhattan distance:
Accuracy: 0.7333
Recall: 0.8229
```

Figure 8: Results for KNN in Manhattan distance

Then, we found the confusion metrics for each $K = 1, and\ K = 3$ in k-NN model, as shown in the figures below:

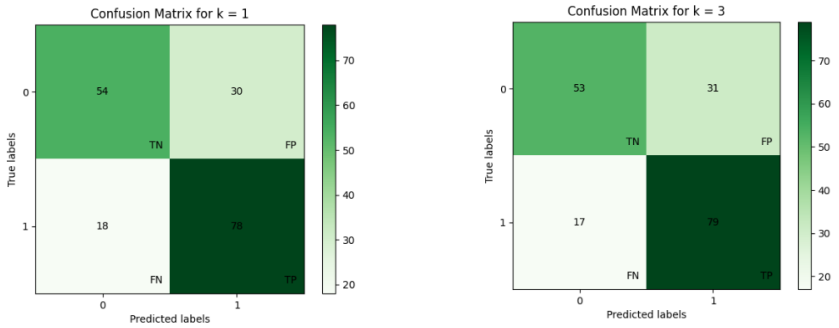


Figure 9: Confusion metrics for both k=1 and k=3 in KNN in Manhattan distance

➢ **Random Forest**

In this model, first, we design a function for hyper parameter tuning of a Random Forest Classifier. It takes the training data (X_train, y_trian) as input, then we splits the training data into new training data of 80% and validation data of 20%. Then, it iterates through different combinations of hyper parameter [n estimators: number of trees in the random forest, max depth: maximum depth of each decision tree in the random forest, min samples split: minimum number of samples required to split an internal node, min samples leaf: minimum number of samples required to be at a leaf node] each one of them have 4 values, specified in the "parameter grid", then creating a RandomForestClassifier for each combination. The model is then trained on a training subset (X_train_new, y_train_new) and evaluated on a validation set (X_val). Accuracy and recall scores are computed for each combination, and the results are stored in a list called reault. The best-performing set of hyper parameters is determined by the highest accuracy score and the corresponding values are returned.

```
-------------------------RF-----------------------------------
Best Hyperparameters:
Best n_estimators: 100
Best max_depth: 10
Best min_samples_split: 7
Best min_samples_leaf: 2
Best Accuracy: 0.8958
Best Recall: 0.9277
```

Figure 10: Best Hyper parameter in RF_testing in validation set

Then RF function is designed to implement a Random Forest classifier, it takes as input the training data features and testing data features. The Random Forest classifier is initialized with the best hyper parameter values return from the RF_testing function, the model is trained using the training data. Subsequently, predictions are made on the testing data, and the function computes and prints the accuracy and recall scores as performance metrics.

```
-------------------------RF-----------------------------------
Results of test data for RF :
Best Accuracy: 0.8833
Best Recall: 0.9583
Confusion Matrix For Random Forest Model
[[67 17]
 [ 4 92]]
```

Figure 11: Best Hyper parameter in RF in testing set

Then, we found the confusion metrics for this model, as shown in the figure below. And we found that our model effectively succeed in reducing false negatives (FN) therefore increasing the recall, a crucial success given their high risk.

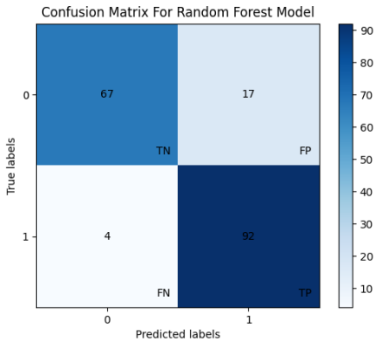


Figure 12: Confusion Matrix for Random Forest model

And we notice that the performance in Random forest is better from the K-NN, because Random Forest's ensemble nature, ability to handle non-linear relationships, and robustness to noise contribute to its improved performance compared to the simpler KNN algorithm. The combination of these factors makes Random Forest a powerful and versatile choice for a wide range of classification tasks, and we can notice that the random forest have less FP, and FN.

➢ Support vector Machine (SVM)

In this model, we first design a function to tune the hyper parameter of the SVM classifier with the same steps applied in RF. We passed the training data as input and split it into new training data and validation tests. Next, it iterates through different combinations of hyper parameters [kernel: kernel parameter and C: Regularization Parameter] each one of them has 4 values defined in the

parameter grid, then in each iterate the model is trained on a new training subset, and predict on the validation set, and find the accuracy and recall for each one, then it return the hyper parameter values for the best accuracy, to be the best hyper parameters values to use it in the testing data.

```
Best Hyperparameters for SVM:
{'C': 0.1, 'kernel': 'linear'}
Best Accuracy: 0.8426
Best Recall: 0.8720
```

Figure 13: Best Hyper parameter in SVM_testing in validation set

Then we use the best hyper parameter values to train the data in the training set, and use the testing data to predict, the values shown in the below figure.

```
-------------------------SVM-----------------------------------
Results of test data for SVM
Accuracy: 0.8778
Recall: 0.9583
Confusion Matrix For SVM Model
[[66 18]
 [ 4 92]]
```

Figure 14: Best Hyper parameter in SVM in testing set

Then, we found the confusion metrics for this model, as shown in the figure below.

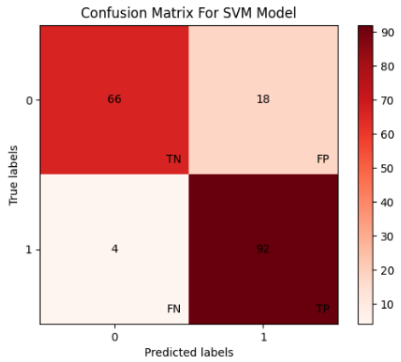
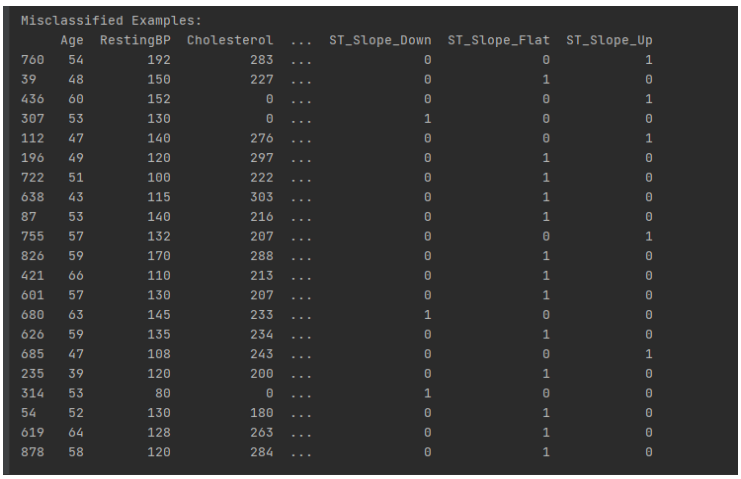


Figure 15: Confusion Matrix for Random Forest model

We can notice that the improved performance of SVM over KNN can be attributed to SVM's ability to handle non-linear relationships, its robustness to irrelevant features and outliers, and the incorporation of a global optimization objective with regularization. These factors collectively contribute to SVM's versatility and effectiveness in diverse classification scenarios.

Analysis the model

In this analysis section, the performance of the best model was thoroughly examined on Random Forest model, with a particular focus on misclassified examples. These instances offered crucial insights into the model's limitations and pointed towards potential areas for enhancement. Then prints out the examples of misclassified instances, offering a tangible representation of where the model struggled.

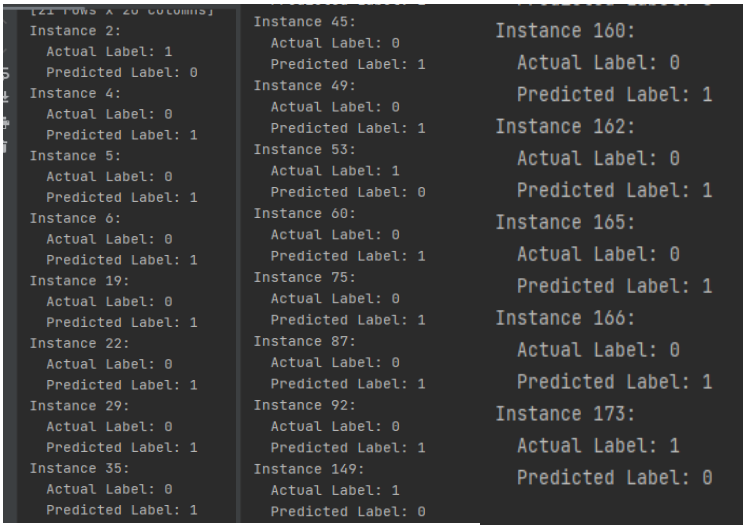


Misclassified Examples:

	Age	RestingBP	Cholesterol	...	ST_Slope_Down	ST_Slope_Flat	ST_Slope_Up
760	54	192	283	...	0	0	1
39	48	150	227	...	0	1	0
436	60	152	0	...	0	0	1
307	53	130	0	...	1	0	0
112	47	140	276	...	0	0	1
196	49	120	297	...	0	1	0
722	51	100	222	...	0	1	0
638	43	115	303	...	0	1	0
87	53	140	216	...	0	1	0
755	57	132	207	...	0	0	1
826	59	170	288	...	0	1	0
421	66	110	213	...	0	1	0
601	57	130	207	...	0	1	0
680	63	145	233	...	1	0	0
626	59	135	234	...	0	1	0
685	47	108	243	...	0	0	1
235	39	120	200	...	0	1	0
314	53	80	0	...	1	0	0
54	52	130	180	...	0	1	0
619	64	128	263	...	0	1	0
878	58	120	284	...	0	1	0

Figure 16: Miss-classified table

Actual labels, a comprehensive understanding of the model's challenges in specific scenarios was obtained. For example, the examination of Instance 45 exposed a notable misclassification where the predicted label ('1') did not align with the actual label ('0').



Instance 2:
  Actual Label: 1
  Predicted Label: 0
Instance 4:
  Actual Label: 0
  Predicted Label: 1
Instance 5:
  Actual Label: 0
  Predicted Label: 1
Instance 6:
  Actual Label: 0
  Predicted Label: 1
Instance 19:
  Actual Label: 0
  Predicted Label: 1
Instance 22:
  Actual Label: 0
  Predicted Label: 1
Instance 29:
  Actual Label: 0
  Predicted Label: 1
Instance 35:
  Actual Label: 0
  Predicted Label: 1
Instance 45:
  Actual Label: 0
  Predicted Label: 1
Instance 49:
  Actual Label: 0
  Predicted Label: 1
Instance 53:
  Actual Label: 1
  Predicted Label: 0
Instance 60:
  Actual Label: 0
  Predicted Label: 1
Instance 75:
  Actual Label: 0
  Predicted Label: 1
Instance 87:
  Actual Label: 0
  Predicted Label: 1
Instance 92:
  Actual Label: 0
  Predicted Label: 1
Instance 149:
  Actual Label: 1
  Predicted Label: 0
Instance 160:
  Actual Label: 0
  Predicted Label: 1
Instance 162:
  Actual Label: 0
  Predicted Label: 1
Instance 165:
  Actual Label: 0
  Predicted Label: 1
Instance 166:
  Actual Label: 0
  Predicted Label: 1
Instance 173:
  Actual Label: 1
  Predicted Label: 0

Figure 17: miss-classification instances

The subsequent visualization of feature distributions for both correctly and misclassified examples corroborated this observation. The visualizations corresponding each feature provide an insightful representation of misclassified examples, where the orange color denotes instances that the model inaccurately classified, and the blue color signifies correctly classified examples.

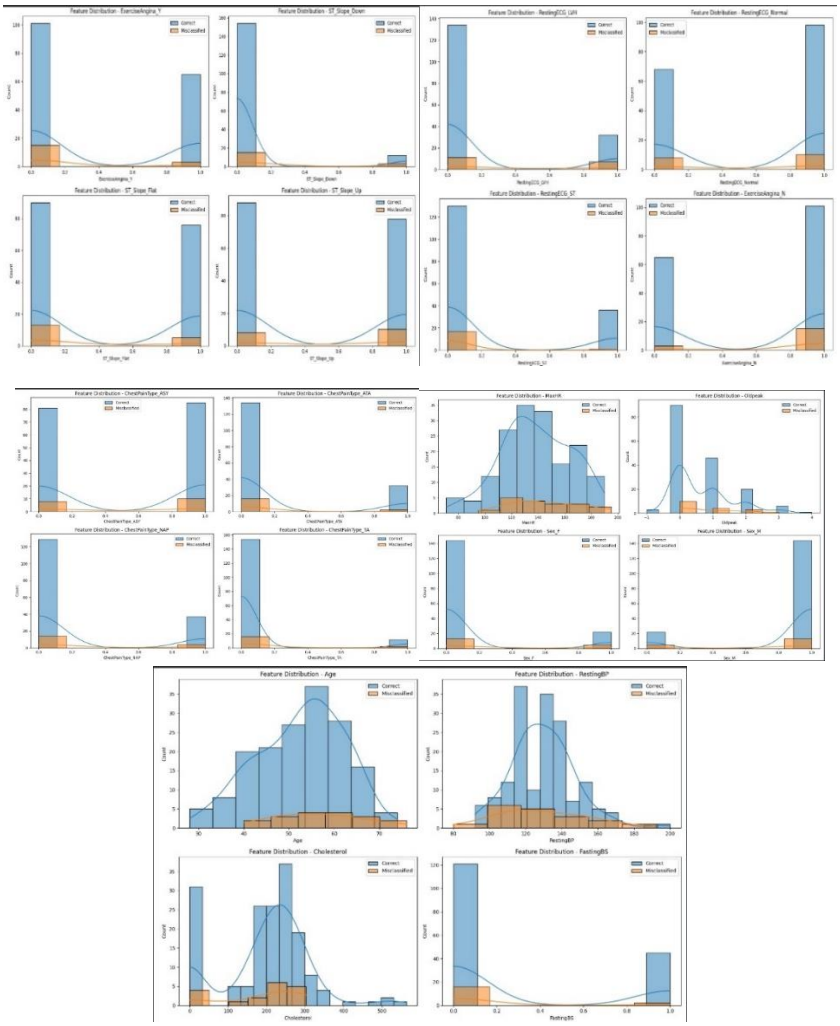


Figure 18: Visualization for a miss-classification examples

Conclusion:

In our project predicting heart disease with k-NN, Random Forest (RF), and SVM models, we observed superior accuracy and recall for RF and SVM compared to k-NN. Notably, RF exhibited the best performance, particularly in recall and accuracy. Our focus on enhancing recall aims to minimize false negatives, crucial for correctly identifying heart disease cases. The analysis of misclassified examples highlighted challenges, such as subtle patterns and overlapping feature distributions, indicating the need for ongoing model optimization through feature engineering and fine-tuning. Whatever the wonderful performance, it's essential to acknowledge the limitations, including dataset representativeness and the trade-off between precision and recall. Continuous improvement remains pivotal for addressing model shortcomings and optimizing heart disease prediction.