INTERNATIONAL UNIVERSITY
VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
School of Computer Science and Engineering

PROJECT REPORT

HEART FAILURE PREDICTION

ARTIFICIAL INTELLIGENCE (IT159IU)

Course by Br. Le Thanh Sach and Dr. Ly Tu Nga

TEAM MEMBERS:	**STUDENTS' ID:**
Tran Dinh An	ITDSIU20113
Tran Ngoc Dang Khoi	ITCSIU21197
Nguyen Hai Ngoc	ITDSIU21057

HCMC, May 2024

TABLE OF CONTENTS

CONTRIBUTION TABLE

No.	Full Name	Student ID	Task	Contribution
1	Trần Đình An	ITDSIU20113	Team Leader with building ANN model and optimizing all models	33.3%
2	Trần Ngọc Đăng Khôi	ITCSIU21197	EDA and pre-processing data	33.3%
3	Nguyễn Hải Ngọc	ITDSIU21057	Building and doing hyperparameter tuning with Naïve Bayes and SVM	33.3%

CHAPTER 1. INTRODUCTION

In the era of AI, the ability to effectively preprocess and analyze datasets has become increasingly vital. This project focuses on leveraging the dataset 'Heart Failure Prediction'

from Kaggle to explore the capabilities of three distinct machine learning models: Naive Bayes, Support Vector Machine (SVM), and Artificial Neural Network (ANN). The goal is to evaluate and compare their performance in a structured manner.

By the end of this project, we aim to identify which model performs best under the given conditions and provide recommendations for similar classification tasks. This report details the methodology, findings, and conclusions derived from our analysis.

1. Methodology

The first phase of the project involves preprocessing and Exploratory Data Analysis (EDA) of the 'Heart Failure Prediction' dataset. Preprocessing is a critical step that ensures the data is clean, consistent, and suitable for analysis. Exploratory Data Analysis (EDA) is conducted to uncover initial insights and detect patterns. This phase includes handling data inconsistencies, scaling numerical features, and splitting the data into training and testing sets. Statistical analysis and data visualization techniques are also implemented to provide a foundation for selecting appropriate features, inform the choice of algorithms, and understand the underlying structure of the data.

The core of the project is the deployment of three machine learning models: Naive Bayes, SVM, and ANN. Each model is trained on 2 sets of preprocessed data – one with all the features and one with only the topmost important features. These models are chosen for their diverse approaches to classification problems:

- **Naive Bayes:** A probabilistic classifier based on Bayes' theorem, particularly effective for high-dimensional data.
- **Support Vector Machine (SVM):** A powerful classifier that finds the optimal hyperplane for separating classes in the feature space.
- **Artificial Neural Network (ANN):** A flexible model that mimics the human brain's neural networks, capable of capturing complex patterns.

The final phase involves comparing the performance of these models using standard evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. This comparative analysis provides insights into the strengths and weaknesses of each model in relation to the specific dataset used.

2. About the dataset

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Heart failure is a common event caused by CVDs, and this dataset contains 12 features that can predict mortality by heart failure.

Most cardiovascular diseases can be prevented by addressing behavioral risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity, and harmful use of alcohol using population-wide strategies. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

3. Productive Tools

- Google Colab: a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs.
- Library used:
  - NumPy: For array operations and mathematical functions. ○ pandas: For data manipulation and analysis. ○ Matplotlib and Seaborn: For data visualization. ○ scikit-learn: For machine learning tasks.
  - Keras: For building and training neural network models.

CHAPTER 2. EDA AND DATA PRE-PROCESSING

This part details the exploratory data analysis (EDA) and preprocessing steps undertaken for a project on predicting heart failure events. The dataset used contains clinical records of patients with various features that may help predict heart failure outcomes.

1. Exploratory Data Analusis (EDA)

Exploratory Data Analysis is the initial step aimed at understanding the dataset and uncovering patterns. Especially in this case, we explore the structure, distribution, and relationships within the data to figure out the risk of dataset.

1.1. Initial Inspection:

- In general, the 'Heart Failure Prediction' dataset contains clinical records of patients, with various features that may contribute to predicting heart failure:
- It includes 299 rows and 13 columns with no missing value by using `df.shape()` and `df.info()`
- It includes 13 features with patient demographics, medical history and clinical measurement information.
- Main datatype: Float and Integer, both numerical and binary representation included.
- There is no duplicated row existing in the dataset. However, the 'Float' datatype is not appropriate with 'age' attribute.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  DEATH_EVENT               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

Figure 2.1. Description of dataset

1.2. Correlation matrix

To check the correlations between the key attribute, `DEATH_EVENT`, with the other attributes, we use correlation heatmap to check.

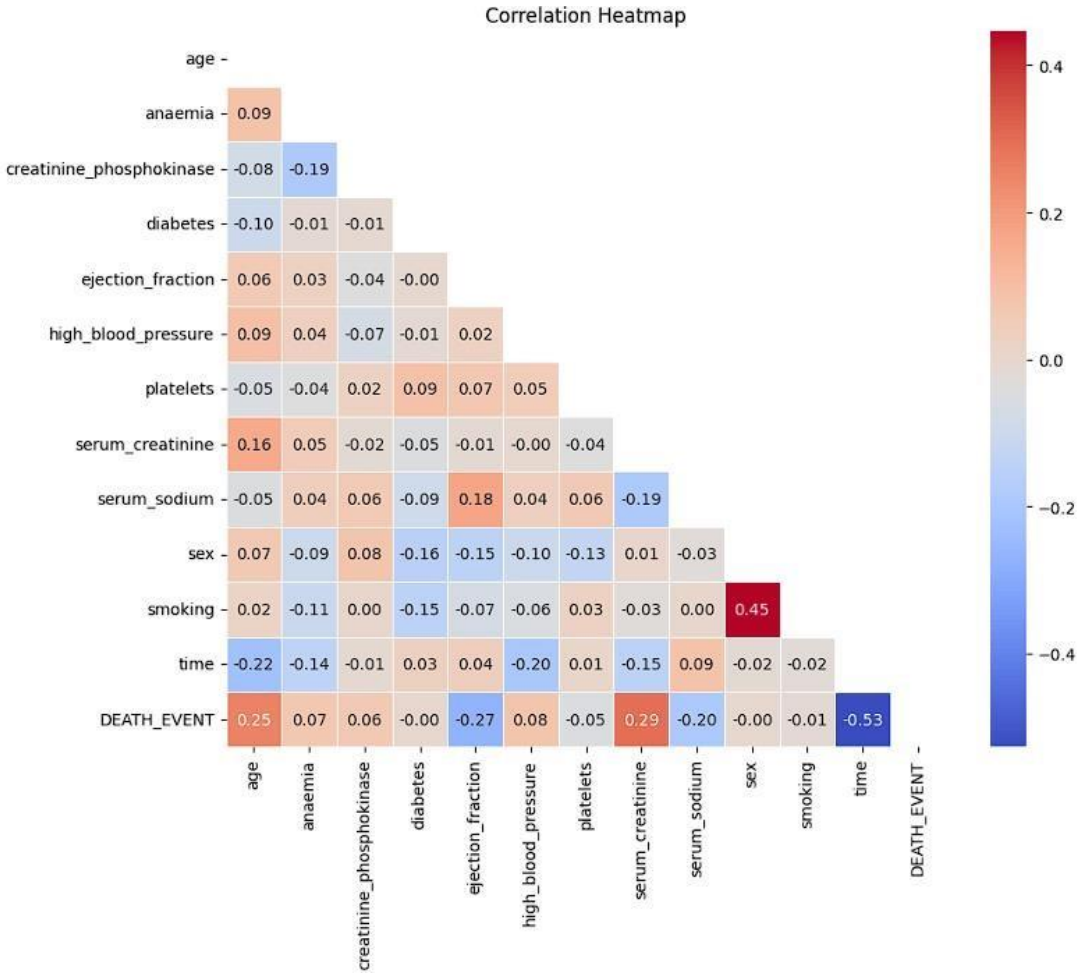


Figure 2.2. Correlation Heatmap of "Heart Failure Prediction" dataset

Here, we easily see that there are four attributes that has the typical correlation to the `DEATH_EVENT` attributes:

- `time` attributes as the most significant negative correlation with -0.53.
- `ejection_fraction ` as the negative correlation with -0.27.
- `serum_creatinine` as the most significant positive correlation values with 0.29. - `age ` as the positive correlation values with 0.29.

In order to make sense about the trend of data, we create `pair plot` graph.

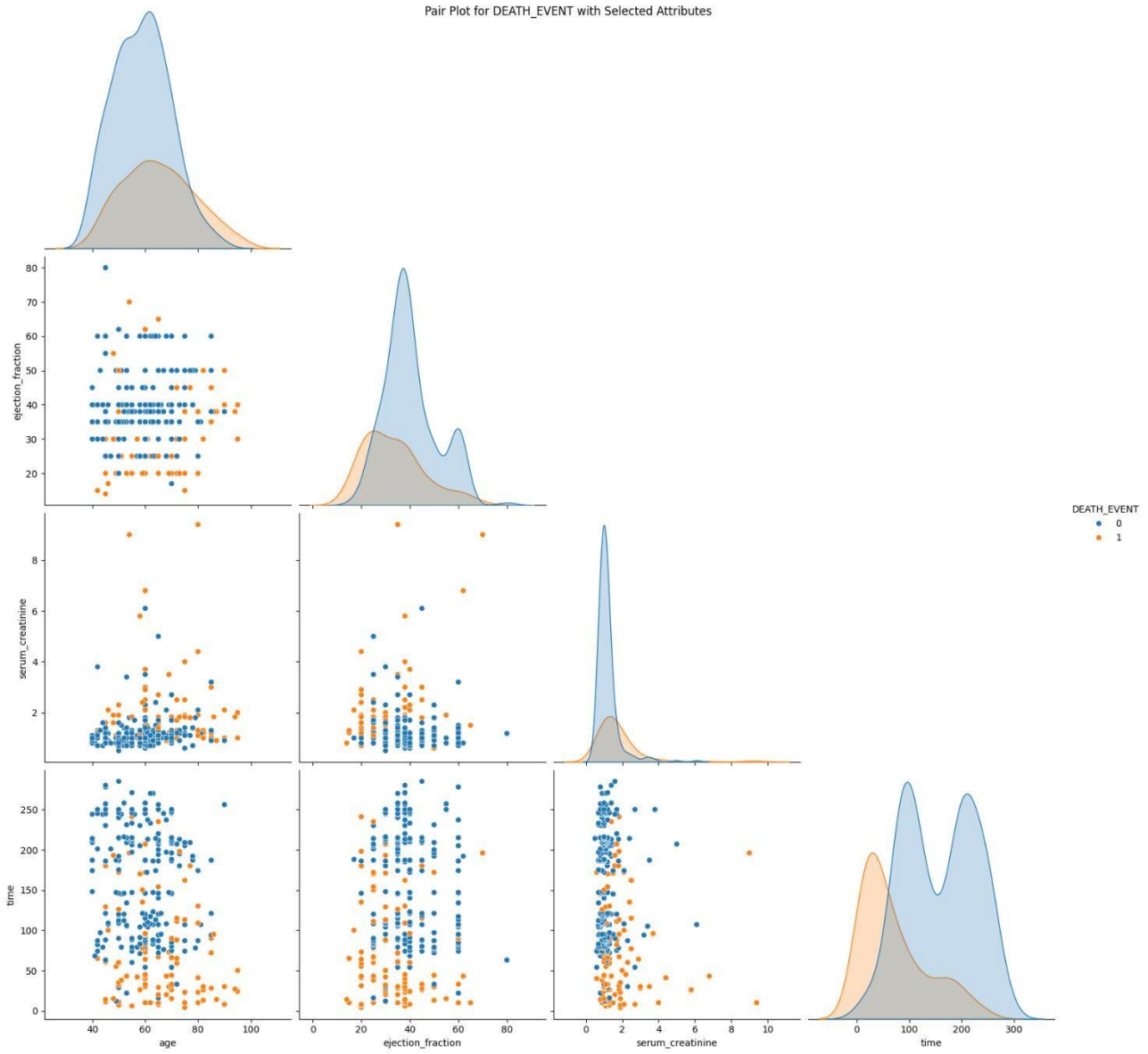


Figure 2.3. 'Pair Plot' graph

Overall, there is no significant trend between these four typical attributes. It may come from the unconsiderable correlation between them, which is testified by the correlation heatmap.

1.3. Outliers

To check the outliers of the dataset, we implement `Box plot` to visualize all the numerical attributes of dataset. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables.The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution.The

box plot (a.k.a. box and whisker diagram) is a standardized way of displaying the distribution of data based on the seven numerical attributes:

- age: age of the patient.
- creatinine_phosphokinase: Level of the CPK enzyme in the blood (mcg/L).
- ejection_fraction: Percentage of blood leaving the heart at each contraction.
- platelets: Platelets in the blood (kiloplatelets/mL)
- serum_creatinine: Level of the serum creatinine in blood (mg/dL)
- serum_sodium: Level of serum sodium in the blood (mEq/L) +

time: Follow-up period (days)

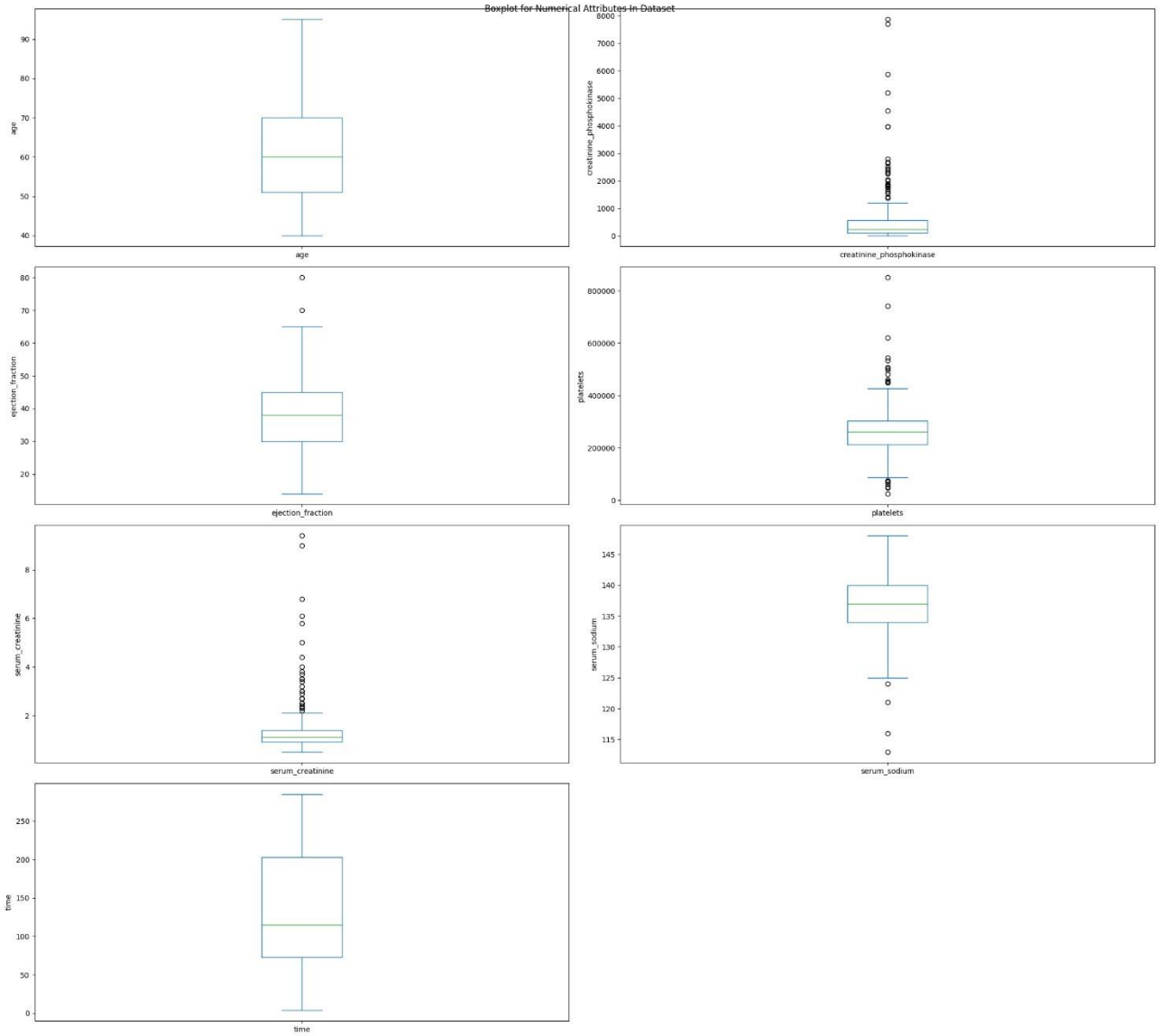


Figure 2.4. 'Box Plot'graph of numerical attributes in dataset

Overall, almost outliers exist in numerical clinical measurement attributes, except for `time` and `age` attribute. It could come from the unknown clinical measurement, which possibly impacts the final result.

1.4. Imbalance

Imbalance in the dataset is a common issue, especially in medical data, where one class is often underrepresented. In this section, we analyze the imbalance in the `DEATH_EVENT` attribute and other categorical variables in the "Heart Failure Prediction" dataset.

Firstly, we used 'Pie chart' to visualize the distribution of two main classes in `DEATH_EVENT` column.

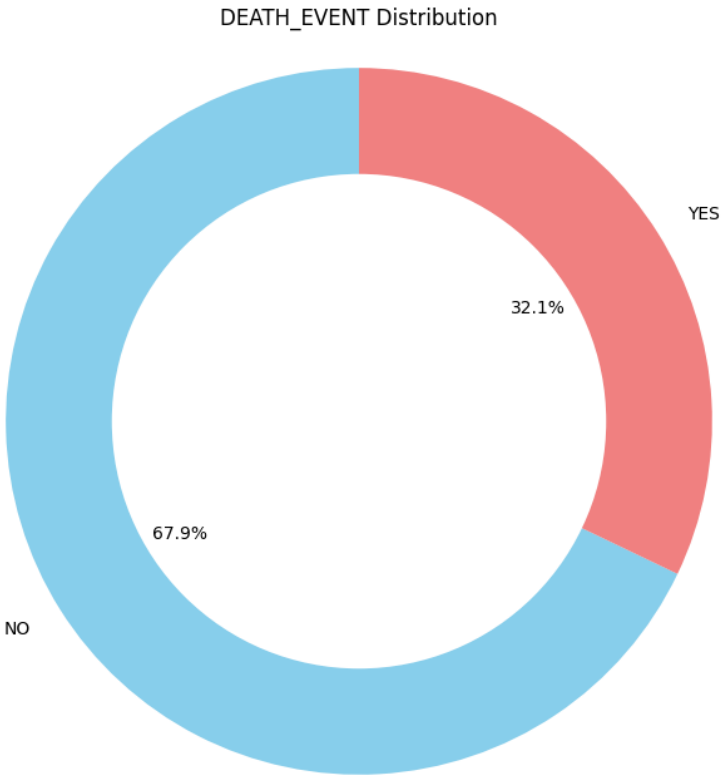


Figure 2.5. Distribution of DEATH_EVENT

From the pie chart, it is evident that there is an imbalance between two classes. Specifically, a significant portion of the data belongs to the class where `DEATH_EVENT` is 0 (indicating the patient did not experience a heart failure event), while a smaller portion belongs to the class where `DEATH_EVENT` is 1 (indicating the patient did experience a heart failure event). This imbalance can lead to a model that is biased towards the majority class, potentially ignoring the minority class.

We further analyzed the distribution of other categorical attributes, noting considerable biases between classes. For example, attributes such as `sex` and `high_blood_pressure` exhibit noticeable skewness in their distributions.

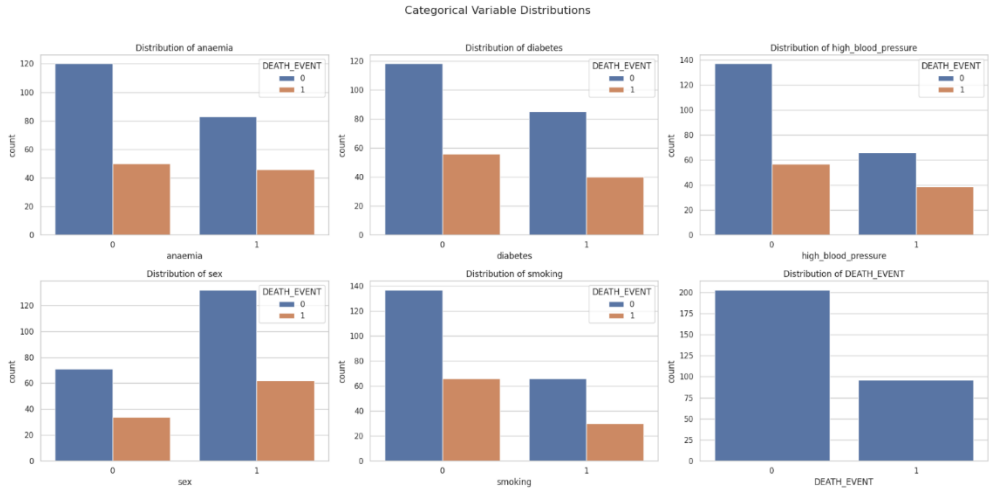


Figure 2.6. Categorical Variable Distribution's Bar Charts

In continuous variable distribution, we examined the spread of values across different categories. The continuous variables were visualized to understand their distribution concerning the target variable, `DEATH_EVENT`. These distributions can provide insights into the underlying patterns and help in addressing the imbalance during model training.

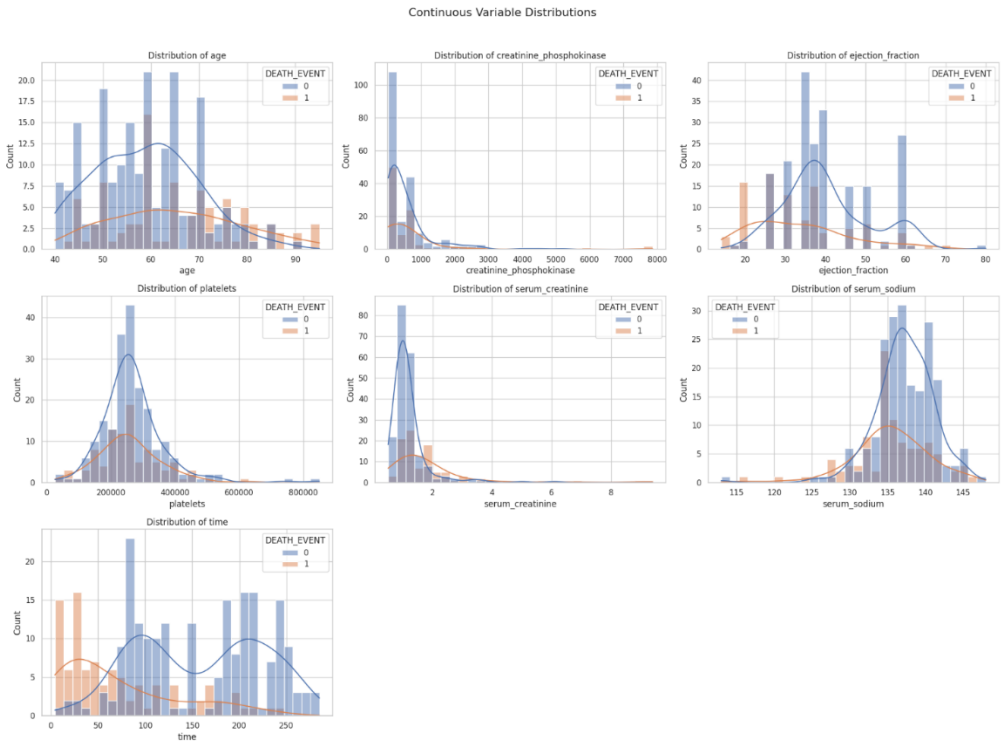


Figure 2.7. Continuous Variable Distribution between the target and the considerable attributes

The imbalance in both categorical and continuous variables highlights the need for careful handling during the preprocessing stage to ensure that the model does not become biased towards the majority class.

1.5. Conclusion

In conclusion, we approach the properties of the dataset with clear description:

- There are 299 non-null values in all the attributes, thus no missing value and no duplicated rows.
- Datatype is also either `Float` or `Integer`, all works well except age, whose datatype `Float` is not suitable for that attribute.
- There are many outliers existing in the dataset, which are mainly consisted in clinical attributes with unknown reasons.
- Imbalance has existed in this dataset, particularly in the target variable and a few categorical variables like `high_blood_pressure` and `sex`.
- There are no trends between the target and the other considerable attributes in this dataset due to low correlation.

2. Data Proprocessing

2.1. Handling inapproriated datatype

To handle the inpproriated datatype mainly happening in `age` attribute, we use `.astype()` method provided by Pandas to convert the `age` attribute's datatype into the `int64` (Integer). This ensures consistency and allows for proper numerical analysis.

```
[ ]  df['age'] = df['age'].apply(np.ceil)
     df['age'] = df['age'].astype('int64')
     df.info()
     df.head()
     df.to_csv('out.csv') # export to check
```

Figure 2.7. Convert 'float64' of 'age' attribute into 'int64'

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    int64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(2), int64(11)
memory usage: 30.5 KB
```

Figure 2.8. Result of conversion

2.2. Handling outliers by Robust Scaler

Feature Scaling is one of the most important steps of Data Preprocessing. Due to the presence of outliers with unknown reasoning in the clinical attributes of the dataset, we opted to use the "Robust Scaler" to scale the dataset's values.

The Robust Scaler is a scaling technique that is particularly effective in handling outliers. Unlike standard scaling methods that use the mean and standard deviation, the Robust Scaler uses the median and the interquartile range (IQR) for scaling. This makes it more resilient to outliers, as these statistics are less influenced by extreme values. It works by:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

Figure 2.9. Robust Scaler formula

- Median Centering: Each feature is centered around the median, ensuring that the scaling is not skewed by outliers.
- Interquartile Range Scaling: Features are scaled according to the IQR, which is the range between the 25th and 75th percentiles. This method reduces the impact of outliers by focusing on the central 50% of the data.

By using the Robust Scaler, we ensure that the scaled dataset maintains robustness against outliers, leading to more reliable and stable models.

```
[ ]  # Columns to scale
     columns_to_scale = ['age', 'creatinine_phosphokinase', 'ejection_fraction',
                         'platelets', 'serum_creatinine', 'serum_sodium', 'time']

     # Select the columns to be scaled
     df_to_scale = df[columns_to_scale]

     # Initialize the RobustScaler
     scaler = RobustScaler()

     # Fit the scaler to the selected data and transform it
     scaled_data = scaler.fit_transform(df_to_scale)

     # Create a DataFrame with the scaled data
     df_scaled = pd.DataFrame(scaled_data, columns=columns_to_scale)

     # Combine the scaled data with the rest of the DataFrame
     df_scaled = pd.concat([df_scaled, df.drop(columns=columns_to_scale)], axis=1)
     df = df_scaled
     df
```

Figure 2.10. Implementation of 'Robust Scaler'

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium	time	anaemia	diabetes	high_blood_pressure	sex	smoking	DEATH_EVENT
0	0.789474	0.713212	-1.200000	0.032967	1.6	-1.166667	-0.853846	0	0	1	1	0	1
1	-0.263158	16.350161	0.000000	0.014923	0.0	-0.166667	-0.838462	0	0	0	1	0	1
2	0.263158	-0.223416	-1.200000	-1.098901	0.4	-1.333333	-0.830769	0	0	0	1	1	1
3	-0.526316	-0.298604	-1.200000	-0.571429	1.6	0.000000	-0.830769	1	0	0	1	0	1
4	0.263158	-0.193340	-1.200000	0.714286	3.2	-3.500000	-0.823077	1	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
294	0.105263	-0.406015	0.000000	-1.175824	0.0	1.000000	1.192308	0	1	1	1	1	0
295	-0.263158	3.372718	0.000000	0.087912	0.2	0.333333	1.200000	0	0	0	0	0	0
296	-0.789474	3.888292	1.466667	5.274725	-0.6	0.166667	1.253846	0	1	0	0	0	0
297	-0.789474	4.646617	0.000000	-1.340659	0.6	0.500000	1.269231	0	0	0	1	1	0
298	-0.526316	-0.116004	0.466667	1.461538	1.0	-0.166667	1.307692	0	0	0	1	1	0

299 rows × 13 columns

Figure 2.11. Result

2.3. Splitting data

To prepare our dataset for modeling, we first separate the features from the target variable (`DEATH_EVENT`) and then split the data into training, validation, and testing sets. This ensures that our model can be trained and validated effectively before being tested on unseen data.

```
X = df.drop(['DEATH_EVENT'], axis=1)
y = df['DEATH_EVENT']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42)
```

Figure 2.12. Splitting data for preparation.

To do that:

- We split dataset into training and testing sets, with 80% of the data used for training and 20% reserved for testing. The `random_state=42` ensures the reproducible split.
- Furtherly, the training data is split into training and validation sets, with 75% of the data used for training and 25% for validation. This allows us to tune our model and validate its performance during the training process.

By structuring the data splits in this manner, we ensure that we have a robust training set, a validation set to tune and evaluate our model during development, and a testing set to asess the model's performance on unseen data.

2.4. Handle imbalance by using SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a popular technique used in machine learning to address the issue of class imbalance in datasets. This is achieved by selecting two or more similar instances in the minority class and creating new synthetic instances that lie along the line segments connecting them.

The main advantage of SMOTE is to enhance the performance of machine learning models by providing a more balanced training set, which can lead to better generalization and improved predictive accuracy for the minority class. This technique is particularly useful for applications such as fraud detection, medical diagnosis, and any domain where imbalanced data is a common challenge.

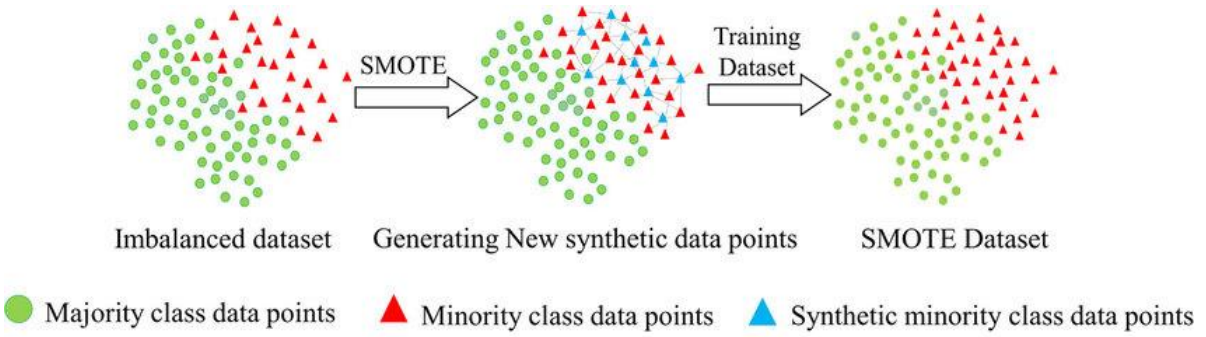
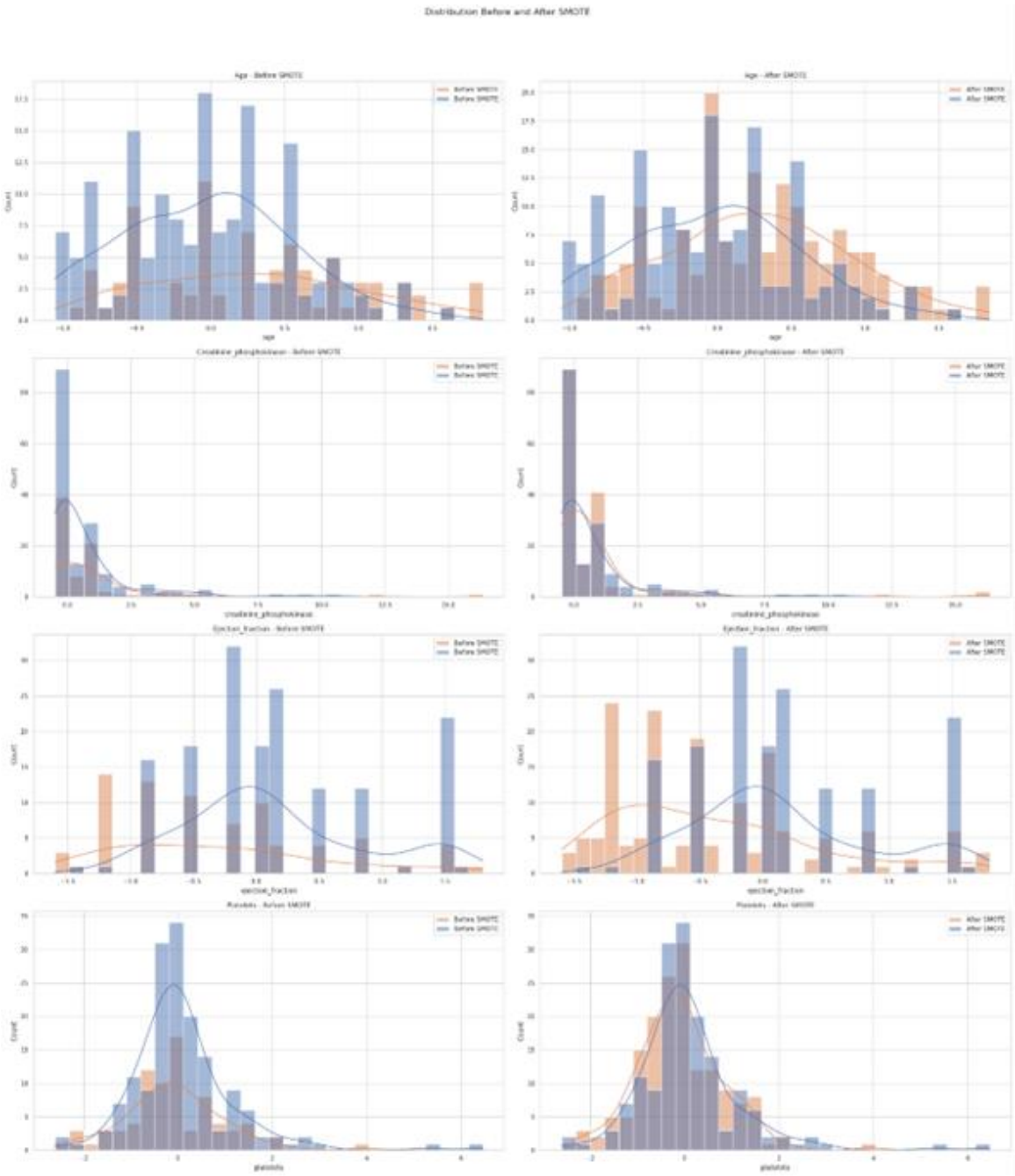


Figure 2.13. SMOTE operation

By implementing SMOTE, we aim to achieve a balance training dataset that can improve the model's ability to generalize and accurately predict both classes, thereby enhancing overall performance.

❖ **Result:**

The application of SMOTE ideally shows an improved balance in the class distributions, as illustrated in the following figure, which compares the dataset before and after the implementation of SMOTE.


Distribution Before and After SMOTE

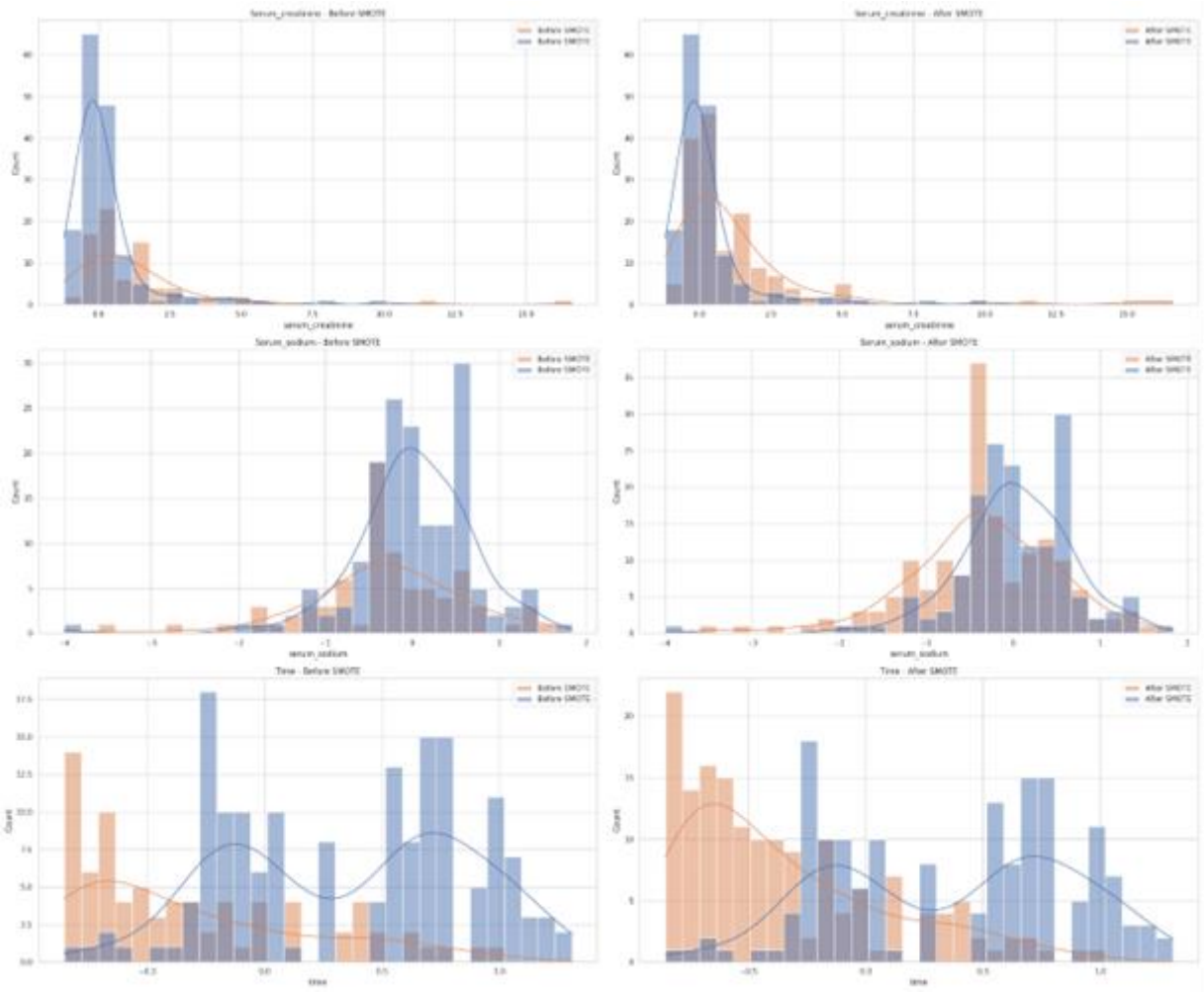


Figure 2.14.. Before and after implemetation of SMOTE

CHAPTER 3. MODELING AND EVALUATION

This chapter outlines our approach to modeling mortality prediction in heart failure cases and evaluating model performance. We present details of the design and architecture of Naive Bayes, Support Vector Machine, and Artificial Neural Network models. We then discuss evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, alongside visual aids like confusion matrices and ROC curves.

1. Naïve Bayesian

Naive Bayes algorithm is a probabilistic classifier based on Bayesian's Theorem, with the "naive" assumption of independence between every pair of features. Despite its simplicity, it often performs well in various applications.

Bayesian's Theorem provides a way to update the probability estimate for a hypothesis as more evidence or information becomes available. It is mathematically expressed as:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Figure 3.1. Bayesian's theorem

Where:

- P(A|B) is the posterior probability of the hypothesis A given the evidence of B
- P(B|A) is the likelihood of the evidence given that A is true.
- P(A) is the prior probability of the hypothesis before seeing the evidence.
- P(B) is the total probability of the evidence.

Naive Bayes assumption simplifies the calculation by assuming the features are conditionally independent given the class. Thus, for a set of features $x = (x_1, x_2, ..., x_n)$, the probability P(x|C) can be written as:

$$\mathbf{P(x|C) = P(x_1|C) \,.\, P(x_2|C) \,.\, \ldots \,.\, P(x_n|C)}$$

In Python, Naive Bayes model includes a parameter call *var_smoothing*. This parameter is used to solve the Zero – Probability problem when computing the probability of each part. In this dataset, *var_smoothing* is assigned to be default. It will be optimized in the future by the hyperparameter tuning process.

In the project, the model is trained twice. First, it is trained with all features of the dataset, which produces an accuracy of 90%.

```
accuracy_score(y_test, model.predict(X_test))
```

```
0.9
```

Figure 3.2. Accuracy score of NB full attributes

Then, the model is trained with 4 attributes that have the highest correlation to the target variable: *age, ejection_fraction, serum_creatinine,* and *time*. Obviously, the accuracy score is still around 90%, as we fix the imbalanced features of data.

```
accuracy_score(y_test, model.predict(X_test))
```

```
0.9
```

Figure 3.3. Accuracy score of NB 4-attributes

After training the model, the testing dataset is used to display the classification report, confusion matrix, and ROC AUC score.

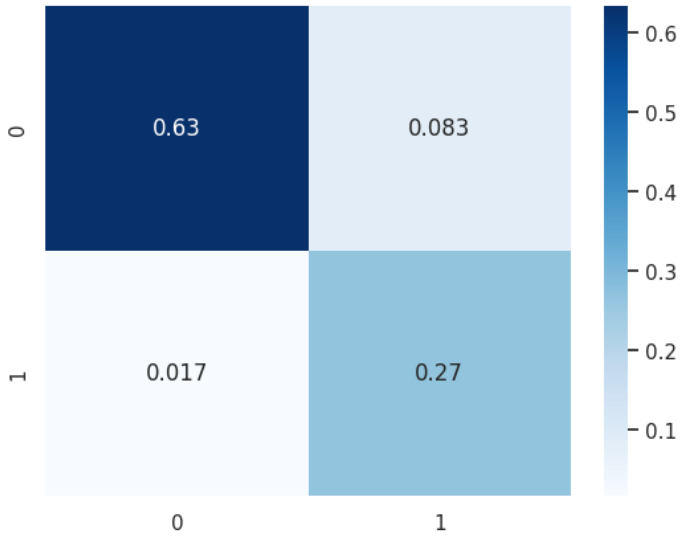


Figure 3.4. Confusion matrix of NB

By the confusion matrix, there is 63% of data that is true positive, 27% true negative, 8.3% false positive, and 1.7% false negative.

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.88	0.93	43
1	0.76	0.94	0.84	17
accuracy			0.90	60
macro avg	0.87	0.91	0.88	60
weighted avg	0.91	0.90	0.90	60

Figure 3.5. Classification report

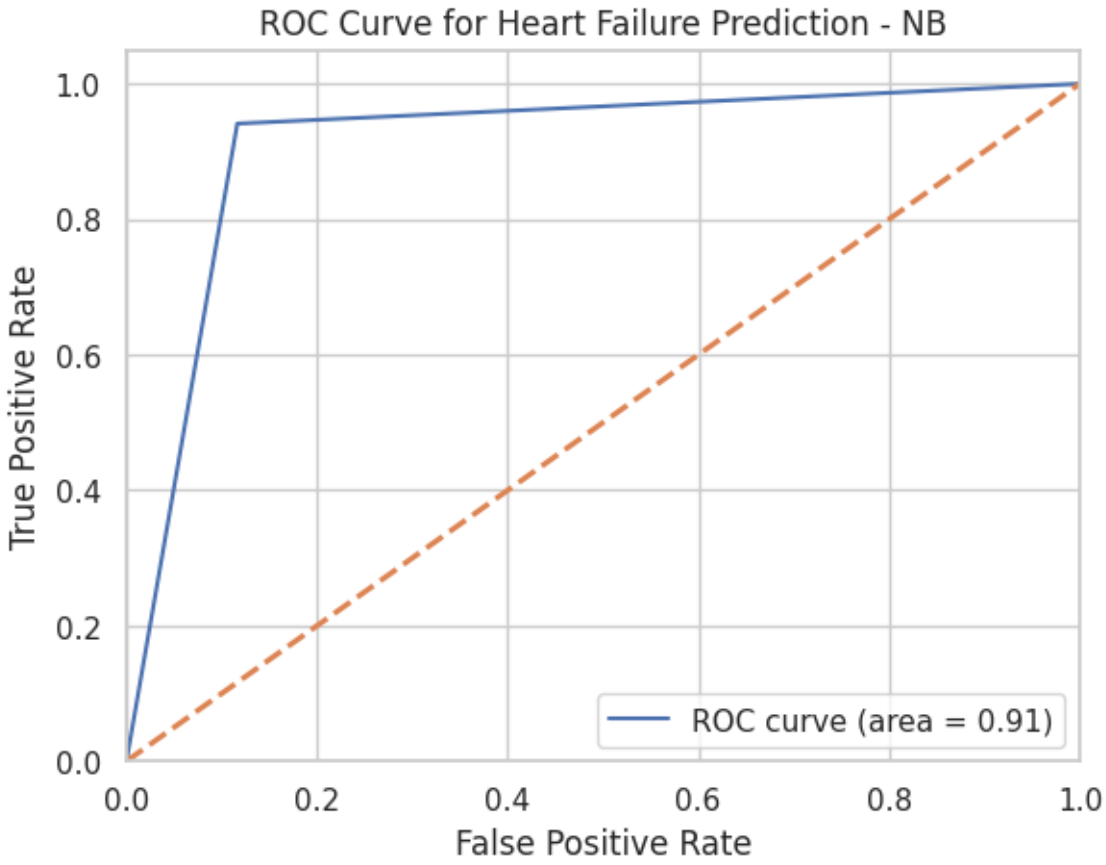


Figure 3.6. ROC Curve

We can easily observe that the precision, recall, and F1-score of class 0 is higher than the result in class 1. The recall of class 1 is slightly higher than class 0 (0.94 > 0.88), which means that the number of correct samples that the model predicted over the total samples that is labeled as class 1, is 94%. However, the precision (76% that the sample predicted correctly by the model over the total of samples being predicted) of class 1 is smaller than class 0. That is the reason for the large range of difference between F1-score (stands for the combination of precision and the recall) of class 0 and class 1. The total accuracy states that the model correctly predicts the class labels for 90% of the samples. Finally, the ROC AUC score of 91% represents that the model has a high ability to separate between the two classes, and we can conclude that Naive Bayes did a better job of classifying the positive class in this dataset.

In this project, we import the **optuna** module for optimizing the models. In Naïve Bayes, *var_smoothing* is the parameter used for fine-tuning. We use call 100 trials in the range of 1e-12 to 1e-6 to find the best parameter.

```
def optimizing(trial):
  var_smoothing = trial.suggest_float('var_smoothing', 1e-12, 1e-6, log=True)
  model = GaussianNB(var_smoothing=var_smoothing)

  skf = StratifiedKFold(n_splits=10)
  scores = cross_val_score(model, X_train1, y_train_smote, cv=skf, scoring='accuracy')

  return scores.mean()
```

```
study = optuna.create_study(direction='maximize')
study.optimize(optimizing, n_trials=100)
```

Figure 3.7. NB's optimizing script

After 100 trials, we find the best parameter is var_smoothing = 4.36e-09.

```
best_params = study.best_params

best_model = GaussianNB(var_smoothing=best_params['var_smoothing'])
best_model.fit(X_train1, y_train_smote)
```

```
GaussianNB
GaussianNB(var_smoothing=4.363460167844285e-09)
```

Figure 3.8. NB's optimizing trials

Then, we use the best parameter to retrain the model, and find the best performance of Naïve Bayes.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.88      0.93        43
           1       0.76      0.94      0.84        17

    accuracy                           0.90        60
   macro avg       0.87      0.91      0.88        60
weighted avg       0.91      0.90      0.90        60


ROC AUC Score:
0.9124487004103967
```

Figure 3.9. NB's optimizing performance

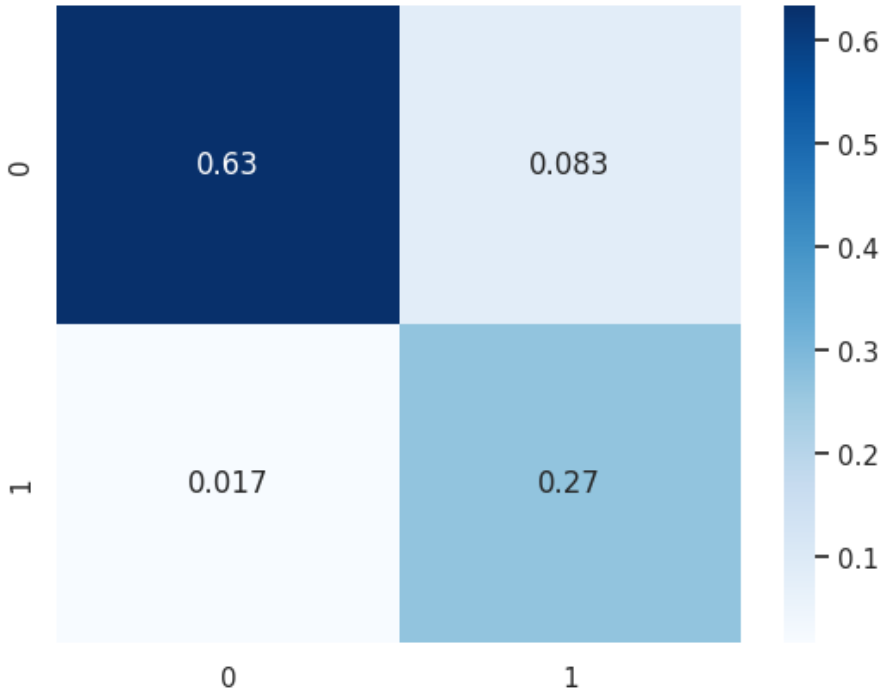


Figure 3.10. NB's optimizing confusion matrix

By the confusion matrix, there is 63% of data that is true positive, 27% true negative, 8.3% false positive, and 1.7% false negative.

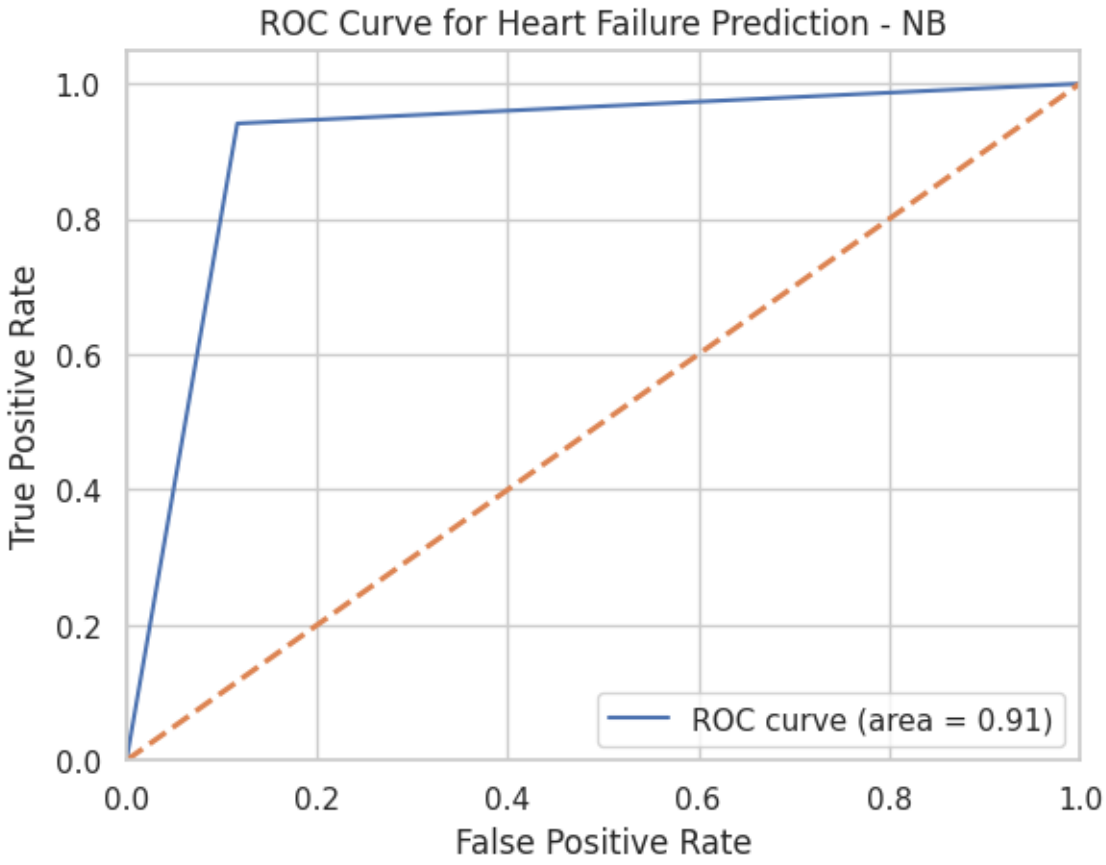


Figure 3.11. NB's optimizing ROC Curve

We can easily observe that the precision, recall, and F1-score of class 0 is higher than the result in class 1. The recall of class 1 is slightly higher than class 0 (0.94 > 0.88), which means that the number of correct samples that the model predicted over the total samples that is labeled as class 1, is 94%. However, the precision (76% that the sample predicted correctly by the model over the total of samples being predicted) of class 1 is smaller than class 0. That is the reason for the large range of difference between F1-score (stands for the combination of precision and the recall) of class 0 and class 1. The total accuracy states that the model correctly predicts the class labels for 90% of the samples. Finally, the ROC AUC score of 91% represents that the model has a high ability to separate between the two classes, and we can conclude that Naive Bayes did a better job of classifying the positive class in this dataset.

Fortunately, the result before tuning and after tuning is the same. Maybe in the pretune model, we call the default parameter in the same range with the best parameter, so it provides with the same performance.

2. Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm used for both classification and regression tasks. SVM is particularly well – suited for classification of complex datasets with a clear margin of separation.

Some key concepts of SVM include:

- **Hyperplane:** In an n-dimensional space, a hyperplane is an (n-1)-dimensional subspace that divides the space into two half-spaces. In SVM, the hyperplane is used to separate different classes.
- **Support Vectors:** These are the data points that are closest to the hyperplane. The position of the support vectors is crucial because they define the hyperplane's orientation and position.
- **Margin:** This is the distance between the hyperplane and the nearest data point from either class. SVM aims to maximize this margin, which helps in achieving better generalization.
- **Kernel Trick**: When data is not linearly separable in the original feature space, SVM uses a kernel function to transform the data into a higher-dimensional space where a hyperplane can be used to separate the classes. Common kernels include: Linear Kernel, Polynomial Kernel, RBF Kernel, and Sigmoid Kernel.

In Python, SVM model is defined as SVC. Parameters of SVC are:

- **C:** Regularization parameter. The strength of the regularization is inversely proportional to **C.** This parameter must be strictly positive.
- **kernel:** Specifies the kernel type to be used in the algorithm. It can be “linear”, “poly”, “rbf”, “sigmoid”, or callable.
- **degree:** Degree of the polynomial kernel function.
- **gamma:** kernel coefficient for “rbf”, “poly”, and “sigmoid”.
- **coef0:** Independent term in kernel function. It is only significant in “poly” and “sigmoid”.

The above information is the fundamental concept of SVM. However, the model is assigned to be default in this current code. In the future, we will optimize those parameters in the process called hyperparameter-tuning.

In the project, the model is trained twice. First, it is trained with all features of the dataset, which produces an accuracy of 86.7%, which means the model correctly predicts the class labels for 86.7% of the samples.

Then, the model is trained with 4 attributes that have the highest correlation to the target variable: *age, ejection_fraction, serum_creatinine,* and *time*. Obviously, the accuracy score increased to around 88%, which means the model correctly predicts the class labels for 88% of the samples after applying features selection.

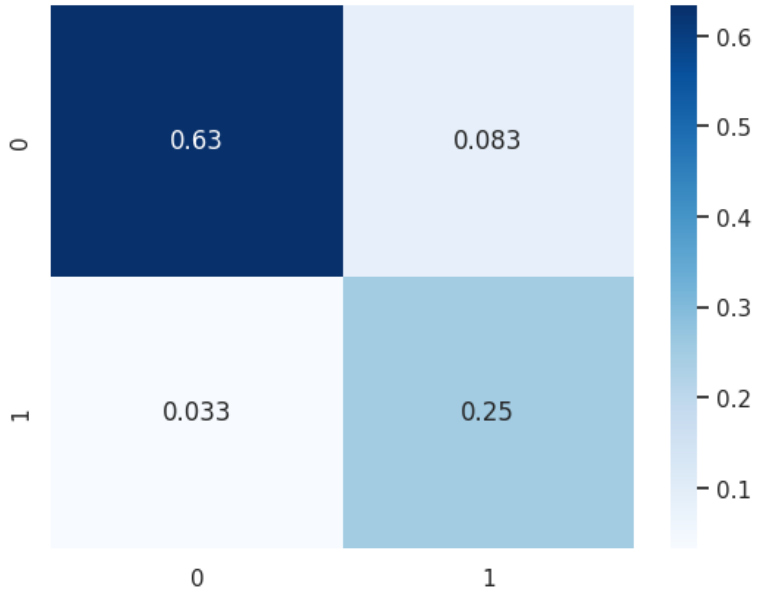


Figure 3.12. Confusion matrix

By the confusion matrix, there is 63% of data that is true positive, 25% true negative, 8.3% false positive, and 3.3% false negative.

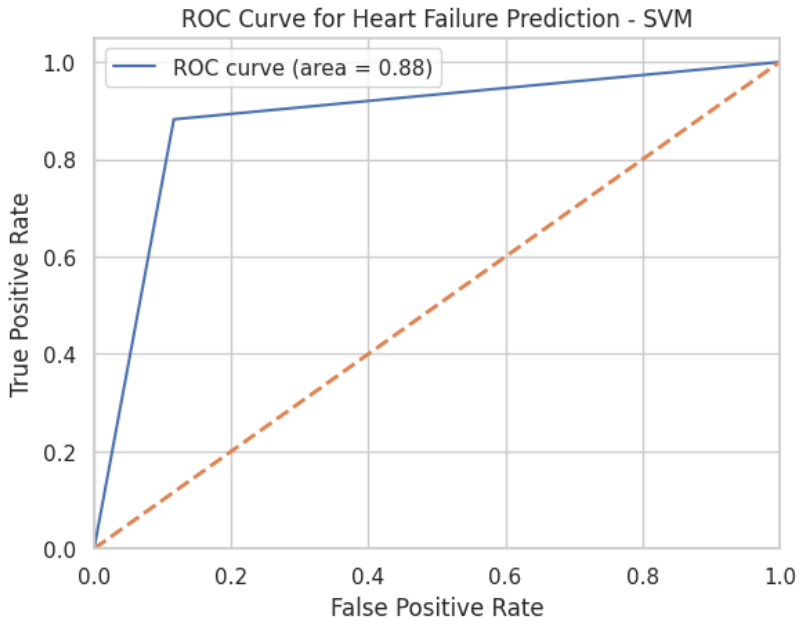


Figure 3.13. ROC Curve

Classification Report:	precision	recall	f1-score	support
0	0.95	0.88	0.92	43
1	0.75	0.88	0.81	17
accuracy			0.88	60
macro avg	0.85	0.88	0.86	60
weighted avg	0.89	0.88	0.89	60

Figure 3.14. Classification report

We can easily observe that the precision, recall, and F1-score of class 0 is slightly higher than the result in class 1. The recall of class 0 is high (0.95), which means that the number of correctly samples that the model predicted over the total samples that is labeled as class 0, is 93%. Recall of class 0 is the same as recall of class 1, and the precision (95% that the sample predicted correctly by the model over the total of samples being predicted) is higher. That is the reason for the large range of difference between F1-score (stands for the combination of precision and the recall) of class 0 and class 1. The total accuracy states that the model correctly predicts the class labels for 88% of the samples. Finally, the ROC AUC score of 88% represents that the model has a high ability to separate between the two classes, and we can conclude that Support Vector Machine did a better job of classifying the positive class in this dataset.

When optimizing SVM, we decided to choose 4 parameters: C, gamma, kernel, and degree.

- C: Float number in range (1e-5, 1e5).
- Gamma: Categorical with “scale” and “auto”.
- Kernel: Categorical with “linear”, “poly”, “rbf”, and “sigmoid”.
- Degree: Integer in the range (1, 5).

```
def optimizing(trial):
  # Suggest values for the hyperparameters
  C = trial.suggest_float('C', 1e-5, 1e5, log=True)
  gamma = trial.suggest_categorical('gamma', ['scale', 'auto'])
  kernel = trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])
  degree = trial.suggest_int('degree', 1, 5) if kernel == 'poly' else 3

  model = SVC(C=C, gamma=gamma, kernel=kernel, degree=degree)
  skf = StratifiedKFold(n_splits=10)
  scores = cross_val_score(model, X_train1, y_train_smote, cv=skf, scoring='accuracy')

  return scores.mean()
```

```
study = optuna.create_study(direction='maximize')
study.optimize(optimizing, n_trials=100)
```

Figure 3.15. SVM’s optimizing script

After 100 trials, the best parameter is printed as followed:

```
best_trial = study.best_trial

print('Best trial:')
print('  Value: {:.4f}'.format(best_trial.value))
print('  Params: ')
for key, value in best_trial.params.items():
    print('    {}: {}'.format(key, value))
```

```
Best trial:
  Value: 0.8594
  Params:
    C: 0.05695330754047806
    gamma: scale
    kernel: rbf
```

Figure 3.16. SVM’’s optimizing trials

```
best_params = study.best_params

best_model = SVC(C=best_params['C'], gamma=best_params['gamma'], kernel=best_params['kernel'])
best_model.fit(X_train1, y_train_smote)
```

```
SVC
SVC(C=0.05695330754047806)
```

Figure 3.17. SVM''s optimizing parameters

Then, we use the best parameter to retrain the model, and find the best performance of SVM.

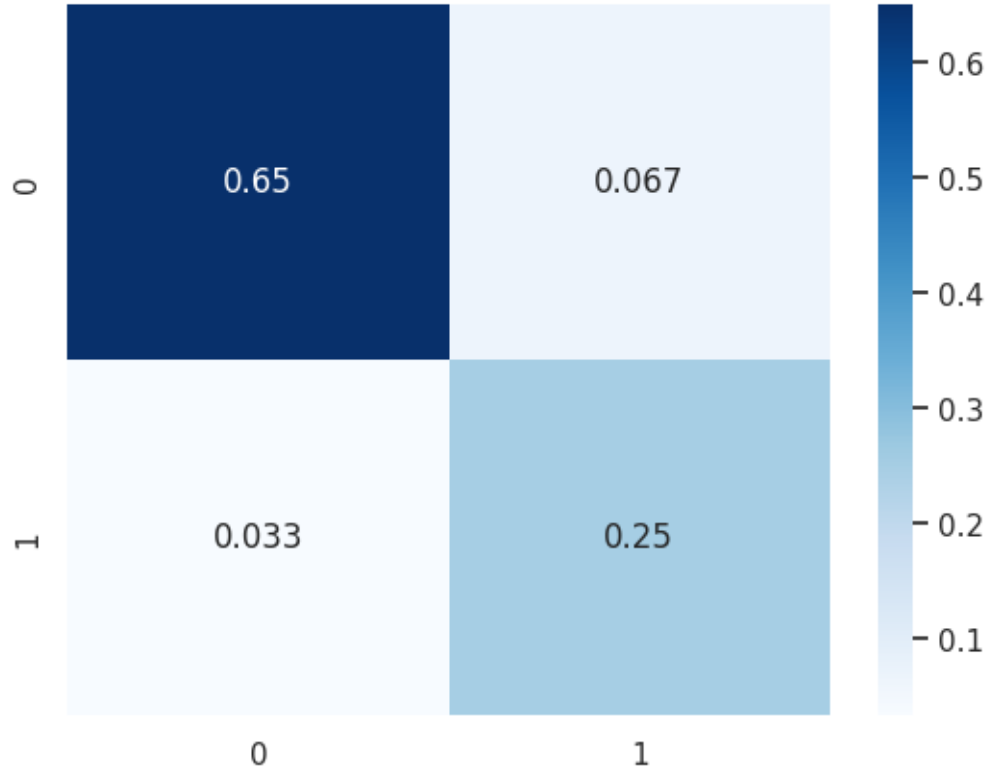


Figure 3.18. SVM''s optimizing confusion matrix

By the confusion matrix, there is 65% of data that is true positive, 25% true negative, 6.7% false positive, and 3.3% false negative.

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.91	0.93	43
1	0.79	0.88	0.83	17
accuracy			0.90	60
macro avg	0.87	0.89	0.88	60
weighted avg	0.91	0.90	0.90	60

Figure 3.19. SVM''s optimizing performance

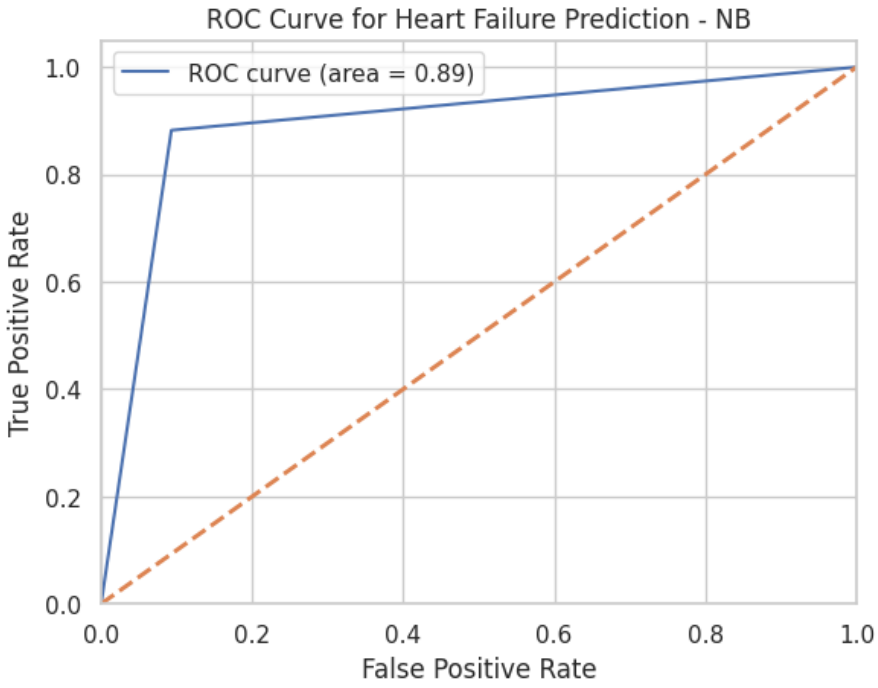


Figure 3.20. SVM''s optimizing ROC Curve

We can easily observe that the precision, recall, and F1-score of class 0 is slightly higher than the result in class 1. The recall of class 0 is high (0.91), which means that the number of correctly samples that the model predicted over the total samples that is labeled as class 0, is 93%. Recall of class 0 is slightly higher recall of class 1, and the precision (95% that the sample predicted correctly by the model over the total of samples being predicted) is higher. That is the reason for the large range of difference between F1-score (stands for the combination of precision and the recall) of class 0 and class 1. The total accuracy states that the model correctly predicts the class labels for 90% of the samples. Finally, the ROC AUC score of 90% represents that the model has a high ability to separate between the two classes,

and we can conclude that Support Vector Machine did a better job of classifying the positive class in this dataset.

3. Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model inspired by the biological neural networks of the human brain. It consists of interconnected nodes, or neurons, organized into layers. ANN is a type of deep learning algorithm capable of learning complex patterns and relationships from data.

Structure of an ANN:

- Input Layer: Neurons in this layer receive input data features.
- Hidden Layers: Intermediate layers between the input and output layers. These layers perform transformations on the input data through weighted connections and activation functions.
- Output Layer: Neurons in this layer provide the final output, often representing predictions or classifications.

```
model = Sequential()
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim=12))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])


early_stop = EarlyStopping(min_delta=0.001, patience=20, restore_best_weights=True, monitor="accuracy")
history = model.fit(X_train_smote, y_train_smote, batch_size = 25, epochs = 500,callbacks=[early_stop])
```

Figure 3.21. ANN's architecture and implementation of the model with all attributes.

A neural network model is defined with an inpur layer, two hidden layers, each using ReLU activation, and one output layer using sigmoid activation for binary classification. A Sequential model, which is a linear stack of layers, is created first. For the input layer, a Dense layer is used. Dense layers are fully connected layers, meaning that each neuron in the layer is connected to every neuron in the preceding layer. We specified the number of neurons in this layer to be 16, the weights of the neurons are initialized uniformly, the activation function used in the neurons to be ReLU, and the number of input features to the model to be 12. The hidden layers are created using two another Dense layers with 8 neurons and ReLU activation.

After each hidden layers, a dropout layer is added with fraction of 25% and 50%, respectively. Dropout is a regularization technique used to prevent overfitting in neural networks. It randomly drops a fraction of the connections between neurons during training to reduce interdependence between neurons. Finally, an output Dense layer with a single neuron and sigmoid activation is added. Sigmoid activation is commonly used in binary classification problems as it squashes the output between 0 and 1, representing the probability of the input belonging to the positive class.

After defining its architecture, the model is complied with Adam optimizer, binary cross-entropy loss function and specified the metric to evaluate the performance of the model during training and testing to accuracy. Adam is a popular optimization algorithm that combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Binary cross-entropy is the loss function used for binary classification problems. It measures the performance of a classification model whose output is a probability value between 0 and 1. An EarlyStopping instance from Keras is created to stop training when a monitored metric – in this case is accuracy - has stopped improving. We specifies the minimum change in the monitored metric must be greater than 0.001 to qualify as an improvement. The number of epochs with no improvement after which training will be stopped is set at 20. And once training stops, the model weights from the epoch with the best value of the monitored metric will be restored. Finally, we train the model with a batch size of 25, run for up to 500 epochs, and include the 'early_stop' instance we created earlier.

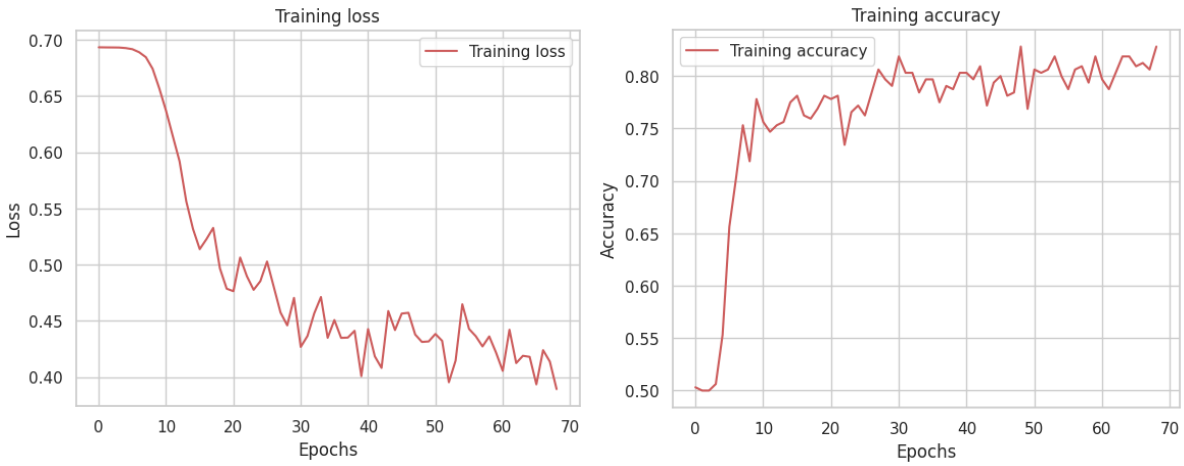


Figure 3.22. Training loss & accuracy of the model with all attributes.

The training process shows a clear improvement in both loss and accuracy metrics over the course of 69 epochs. Initially, the model starts with a loss of 0.6932 and an accuracy of

0.5031, which is close to a random guess for a binary classification problem. However, as the training progresses, there is a noticeable decline in the loss value, indicating that the model is learning and improving its ability to make predictions. By epoch 69, the loss has significantly decreased to 0.3894, suggesting that the model has become much better at minimizing the error between its predictions and the actual labels.

Correspondingly, the accuracy metric also improves markedly throughout the training process. From an initial accuracy of around 50%, the model quickly starts to gain traction, reaching over 80% accuracy by epoch 69. This improvement in accuracy indicates that the model's predictions are becoming increasingly correct, as it is correctly classifying a higher proportion of the instances. The rapid increase in accuracy during the early epochs, particularly from epoch 5 to epoch 10, signifies that the model is effectively learning from the training data. Afterward, while the accuracy continues to improve, the increments become smaller, suggesting that the model is approaching its optimal performance

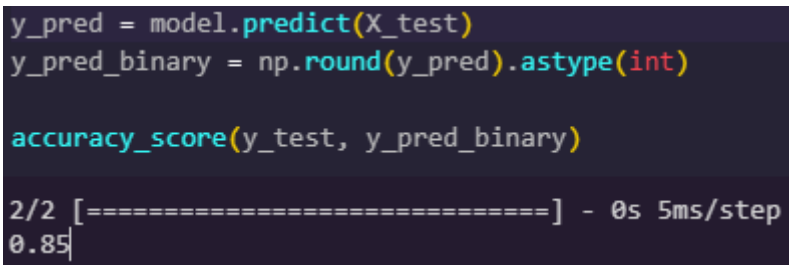


Figure 3.23. Test set performance of the model with all attributes

The evaluation of the model on the test data yields an accuracy score of 0.85, which indicates that the model correctly predicts 85% of the test instances. This performance is quite promising, especially considering that the training accuracy was around 82% by epoch 69. The high test accuracy suggests that the model has effectively generalized from the training data to new, unseen data, which is a crucial aspect of a robust machine learning model.

We then create another model with the same architecture of the previous model. The only difference from the first model is the 'input_dim' parameter of the input layer to be 4, since this time we will only train the model using 4 most important attributes found in the 'Preprocessing & EDA' part.

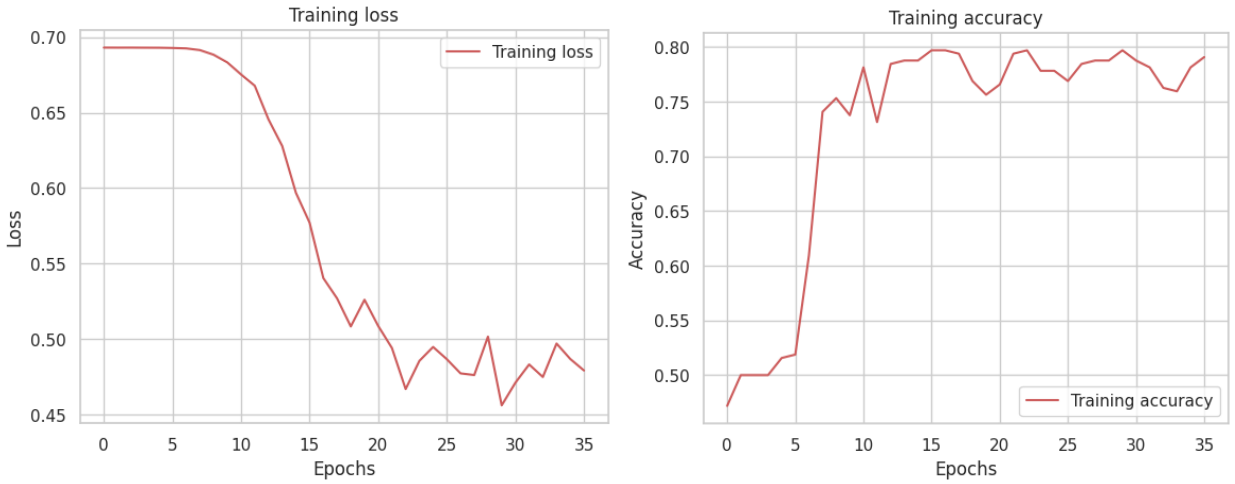
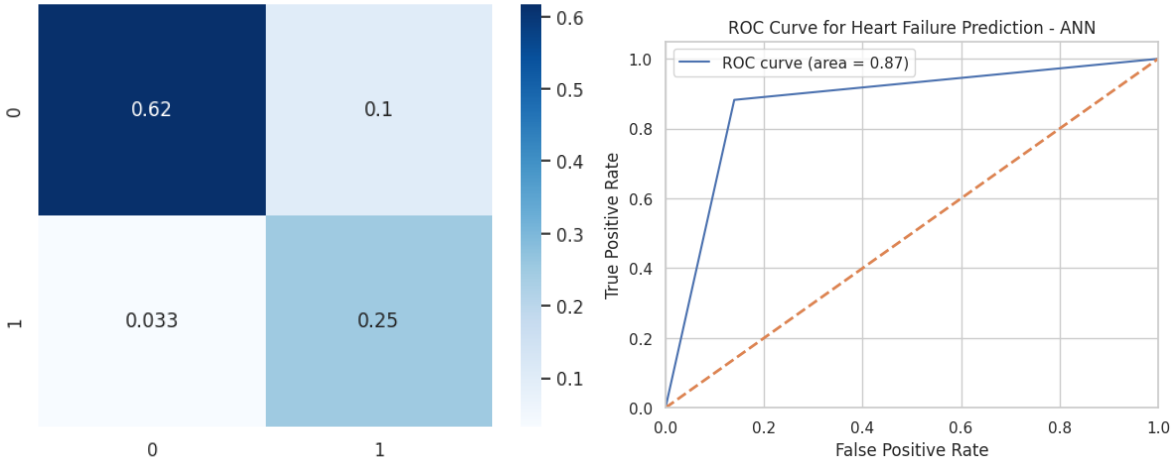


Figure 3.24. Training loss & accuracy of the model with 4 most important attributes.

Initially, both models start with a similar loss around 0.6932, reflecting the inherent difficulty in distinguishing between classes at the start of training. However, the model using all attributes shows a more pronounced improvement in accuracy from the very early epochs, whereas the model using only 4 attributes shows a slower initial improvement. For instance, the accuracy of the reduced attribute model starts at 47.19% and reaches around 75% by epoch 10, whereas the full attribute model shows a sharper increase in accuracy and reduction in loss during the same period.

By epoch 36, the model using all attributes achieves a higher overall accuracy and a lower loss, indicating a better fit and learning efficiency. Specifically, the accuracy for the model with all attributes was already above 80% around epoch 20, whereas the model with only 4 attributes hovers around the 75-80% mark, suggesting that the additional features provided valuable information that the model could leverage to improve its performance.



Classification Report:	precision	recall	f1-score	support
0	0.95	0.86	0.90	43
1	0.71	0.88	0.79	17
accuracy			0.87	60
macro avg	0.83	0.87	0.85	60
weighted avg	0.88	0.87	0.87	60

Figure 3.25. Confusion matrix, ROC Curve and Classification Report of the ANN model with 4 attributes.

The classification report shows that the model achieves a high overall accuracy of 0.87, correctly classifying 87% of the test instances. For class 0, the model has a precision of 0.95, recall of 0.86, and an F1-score of 0.90, indicating strong performance in correctly identifying class 0 instances. For class 1, the precision is lower at 0.71, but the recall is high at 0.88, resulting in an F1-score of 0.79. This suggests that the model is good at identifying class 1 instances but less confident compared to class 0.

The confusion matrix shows that 62% of class 0 predictions are correct, with a small fraction (0.1) misclassified as class 1. For class 1, 0.033 indicates a low false positive rate, and 0.25 shows a reasonable true positive rate. The ROC AUC score of 0.8714 confirms the model's strong ability to distinguish between the two classes, indicating a high true positive rate while maintaining a low false positive rate.

```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=94)

def optimizing(trial):
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-1, log=True)
    batch_size = trial.suggest_int("batch_size", 10, 50)
    dropout_1st = trial.suggest_float("dropout_1st", 0.2, 0.5)
    dropout_2nd = trial.suggest_float("dropout_2nd", 0.5, 0.8)

    model = create_model(learning_rate, dropout_1st, dropout_2nd)

    cv_scores = []
    for train_index, val_index in skf.split(X_train1, y_train_smote):
        X_train_fold, X_val_fold = X_train1.to_numpy()[train_index], X_train1.to_numpy()[val_index]
        y_train_fold, y_val_fold = y_train_smote.to_numpy()[train_index], y_train_smote.to_numpy()[val_index]

        early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='min', restore_best_weights=True)

        history = model.fit(X_train_fold, y_train_fold, epochs=200, batch_size=batch_size,
                            validation_data=(X_val_fold, y_val_fold), callbacks=[early_stop], verbose=0)

        val_loss, val_acc = model.evaluate(X_val_fold, y_val_fold, verbose=0)
        cv_scores.append(val_acc)

    return np.mean(cv_scores)
```

Figure 3.26. ANN's optimizing script

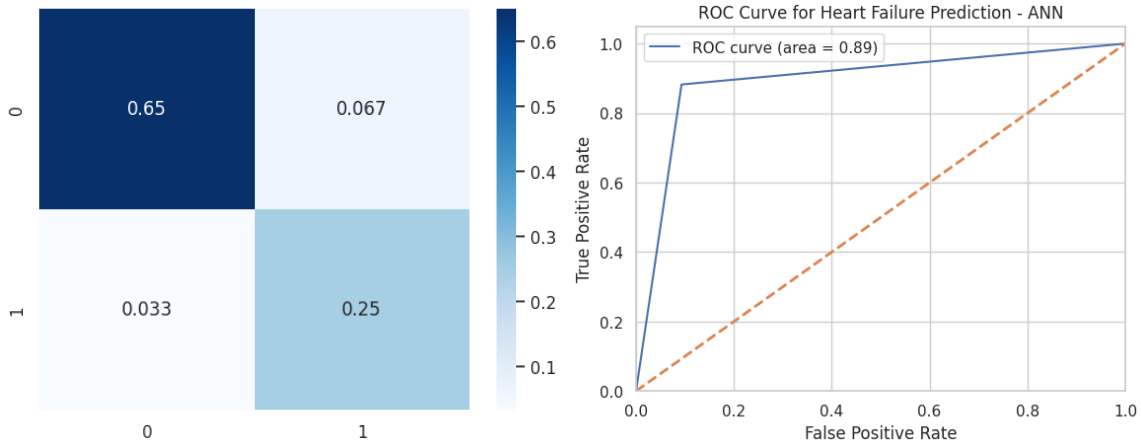
The optimizing function is designed for fine-tuning neural network hyperparameters using Optuna and StratifiedKFold for cross-validation. It suggests hyperparameters such as learning rate, batch size, and dropout rates to enhance model performance. The search space for these parameters ranges from 10^{-5} to 10^{-1} for learning rate, 10 to 50 for batch size, and 0.2 to 0.5 for the first layer dropout rate and 0.5 to 0.8 for the second layer dropout rate.

The model is created based on these suggestions, then trained and evaluated using StratifiedKFold with 5 splits to ensure representative class distribution in both training and validation sets. Early stopping is employed to prevent overfitting during training. The function returns the mean validation accuracy across all folds, serving as the objective for the Optuna optimizer.

Figure 3.27. Training loss & accuracy of the optimized model.

The training history of the optimized model provides valuable insights into its performance over the course of training. Looking at the loss values, we observe a general decreasing trend over epochs, indicating that the model is effectively minimizing its error. This decline in loss signifies that the model is learning to better fit the training data with each epoch. However, fluctuations in the loss curve might suggest that the learning process encounters some challenges, such as noisy data or model instability.

In terms of accuracy, we see fluctuations as well, but with an overall increasing trend. This suggests that the model's performance on the training data improves gradually over epochs. The fluctuations could be due to the stochastic nature of the optimization process, as well as the model's adaptation to different patterns present in the data. Despite these fluctuations, the increasing trend indicates that the model is effectively learning and making progress in capturing the underlying patterns in the training data.



Classification Report:	precision	recall	f1-score	support
0	0.95	0.91	0.93	43
1	0.79	0.88	0.83	17
accuracy			0.90	60
macro avg	0.87	0.89	0.88	60
weighted avg	0.91	0.90	0.90	60

Figure 3.28. Confusion matrix, ROC Curve and Classification Report of the optimized ANN model.

The optimized ANN model shows strong performance on the test set across various evaluation metrics. Precision scores for both classes are high, with class 0 achieving 0.95 and class 1 achieving 0.79. Similarly, recall scores are impressive, with class 0 at 0.91 and class 1

at 0.88, indicating the model's effectiveness in identifying instances of both classes. The F1-scores further emphasize the model's capability, with class 0 achieving 0.93 and class 1 achieving 0.83.

Overall accuracy on the test set is commendable at 0.90, reflecting the model's ability to correctly predict class labels for the majority of instances. The confusion matrix illustrates a high number of true positives and true negatives, with some instances of false positives and false negatives, yet maintaining strong overall performance. Additionally, the ROC AUC score, measuring the model's ability to discriminate between classes, is noteworthy at 0.89, indicating excellent discriminatory power.

CHAPTER 4. COMPARISION

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.88	0.93	43
1	0.76	0.94	0.84	17
accuracy			0.90	60
macro avg	0.87	0.91	0.88	60
weighted avg	0.91	0.90	0.90	60

ROC AUC Score:
0.9124487004103967

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.91	0.93	43
1	0.79	0.88	0.83	17
accuracy			0.90	60
macro avg	0.87	0.89	0.88	60
weighted avg	0.91	0.90	0.90	60

ROC AUC Score:
0.8946648426812586

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.91	0.93	43
1	0.79	0.88	0.83	17
accuracy			0.90	60
macro avg	0.87	0.89	0.88	60
weighted avg	0.91	0.90	0.90	60

ROC AUC Score:
0.8946648426812586

Figure 4.1. Classification Reports of optimized model

The optimized models—Naïve Bayes, SVM, and ANN—show robust performance on the test set, with each achieving high accuracy and balanced classification metrics.

The Naïve Bayes model demonstrates a slight edge in terms of ROC AUC score at 0.91, indicating its strong ability to differentiate between classes. Its precision for class 0 is exceptionally high at 0.97, but it shows a lower precision for class 1 at 0.76. However, the recall for class 1 is very high at 0.94, suggesting it effectively identifies positive instances.

Both the SVM and ANN models exhibit identical performance metrics. They both achieve an accuracy of 0.90 and share the same precision, recall, and F1-scores across both classes. Class 0 has a precision of 0.95 and recall of 0.91, while class 1 shows a precision of 0.79 and recall of 0.88. The ROC AUC score for both models is 0.89, indicating strong discriminative capabilities.

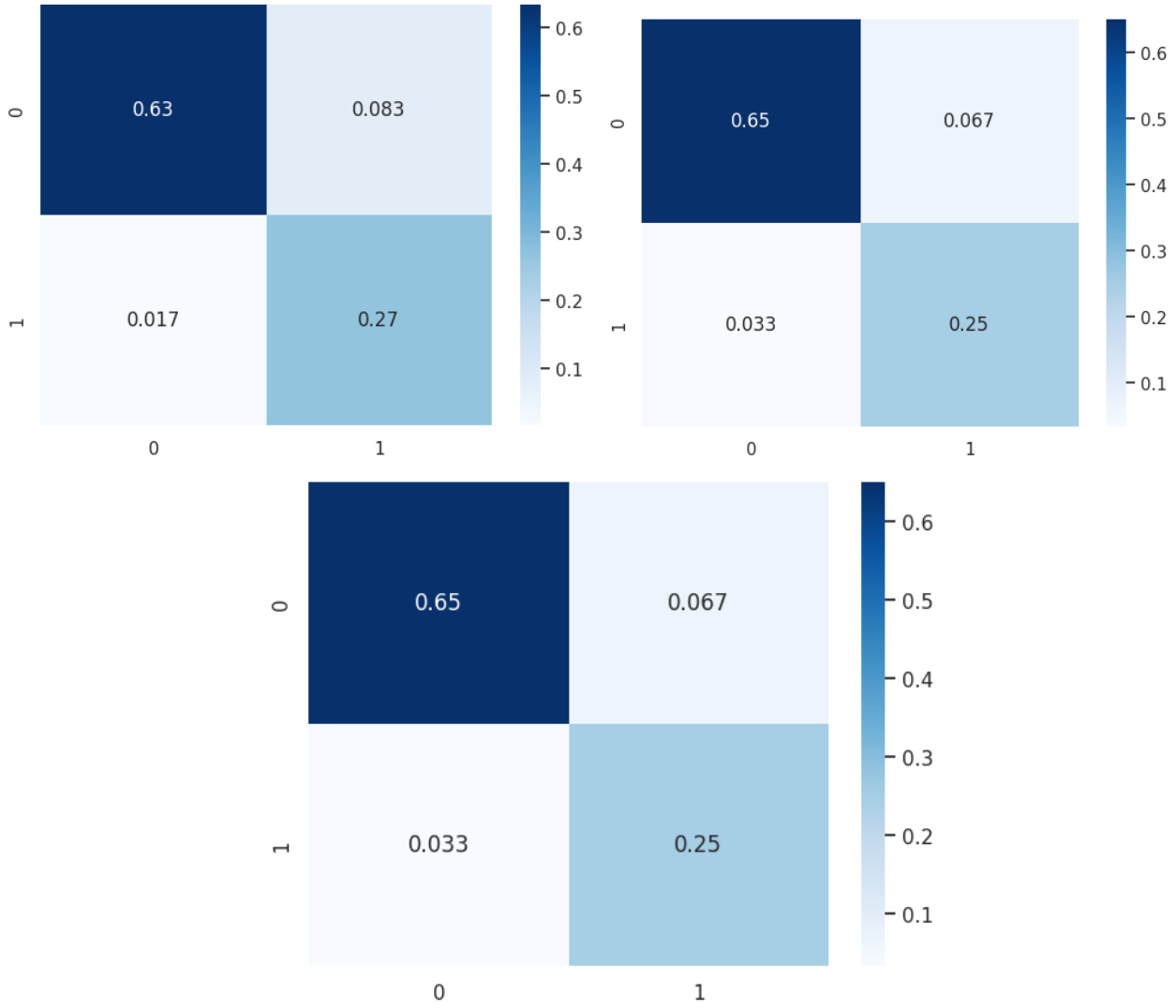


Figure 4.2. Confusion Matrices of optimized model

The confusion matrices for the Naive Bayes, SVM, and ANN models provide additional insights into their performance.

The Naive Bayes model shows a higher rate of correctly identifying true negatives (0.63) and a slightly lower false positive rate (0.083) compared to the SVM and ANN models. It also exhibits a higher true positive rate (0.27) and the lowest false negative rate (0.017). This indicates that Naive Bayes is particularly effective in correctly classifying positive instances, which aligns with its high recall for class 1.

The SVM and ANN models have identical confusion matrices, with true negative rates of 0.65 and false positive rates of 0.067. Their true positive rates are 0.25, with a false negative rate of 0.033. These models slightly outperform Naive Bayes in correctly identifying true

negatives but have a marginally higher false negative rate, indicating a minor trade-off between identifying positive and negative instances.

CHAPTER 5. CONCLUSION AND FUTURE WORKS

1. Conclusions

In this project, we developed and evaluated three different machine learning models: Naive Bayes, Support Vector Machine (SVM), and Artificial Neural Network (ANN) for a

binary classification task. Each model was optimized to improve performance, and their results were compared based on classification reports, confusion matrices, and ROC AUC scores.

The Naive Bayes model demonstrated high recall for the positive class and an overall accuracy of 90%. Its performance, reflected in the confusion matrix, showed a strong ability to correctly classify positive instances with a low false-negative rate. The SVM and ANN models achieved identical performance metrics, with an accuracy of 90%, balanced precision, and recall for both classes. Both models showed slightly better precision for the negative class compared to Naive Bayes, with similar false positive and false negative rates.

The ROC AUC scores for Naive Bayes (0.912) were marginally higher than those for SVM and ANN (both 0.895), indicating that while all models perform well, Naive Bayes has a slight edge in distinguishing between the two classes.

2. Future Works

- Feature Engineering: Further refining feature selection and engineering could improve model performance. Techniques such as polynomial features, interaction terms, or domain-specific feature creation could be explored.
- Feature Engineering: Further refining feature selection and engineering could improve model performance. Techniques such as polynomial features, interaction terms, or domain-specific feature creation could be explored.
- Feature Engineering: Further refining feature selection and engineering could improve model performance. Techniques such as polynomial features, interaction terms, or domain-specific feature creation could be explored.
- Feature Engineering: Further refining feature selection and engineering could improve model performance. Techniques such as polynomial features, interaction terms, or domain-specific feature creation could be explored.
- Feature Engineering: Further refining feature selection and engineering could improve model performance. Techniques such as polynomial features, interaction terms, or domain-specific feature creation could be explored.

REFERENCES

[1]: Davide C., Giuseppe J. (2020, 03 February): *Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone.* BMC Medical Informatics and

Decision Making. https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-0201023-5