UNIVERSITY OF HERTFORDSHIRE

School of Physics, Engineering, and Computer Science

MSc Data Science and Analytics with Advanced Research

7COM1039-0109-2022 - Advanced Computer Science Masters Project

28 August 2023

PROJECT TITLE

Diabetes Prediction using Different Machine Learning Algorithm

Fatema Tuj Johura

20015673

Supervisor: Uchenna Ojiako

MSc Final Project Declaration

I hereby present this report as a partial fulfillment of the requirements for attaining the Master's degree in Data Science and Analytics with Advanced Research at the University of Hertfordshire (UH).

This work is an outcome of my effort unless otherwise stated within the document. No human participants were involved in the research project.

I grant permission for the report to be accessible on the university's official platform, with due credit to the source.

Acknowledgments

I would like to begin by expressing my heartfelt gratitude for the divine blessings of the Almighty, which have enabled me to complete the Final Project Report (FPR).

My sincere appreciation goes to **Uchenna Ojiako, Lecturer at the Department of SPECS, University of Hertfordshire**. **Her deep knowledge and keen interest in the field of "Android Security" have been instrumental in guiding this project**. Her unwavering patience, scholarly guidance, consistent encouragement, and energetic supervision have played a pivotal role in bringing this thesis to fruition.

I extend my gratitude **to Gani Nashi, Programme Leader, Department of SPECS, and Bente Riegler, Module Leader, Department of SPECS,** for their invaluable assistance in completing this thesis. I also acknowledge the support of other faculty members and the staff of the SPECS department at the University of Hertfordshire.

Lastly, I extend my heartfelt thanks to my loving parents for their unwavering support and patience throughout this journey.

Table of Contents

1. Abstract:

Diabetes is a chronic and potentially life-threatening disease characterized by high blood sugar levels due to the body's inability to produce or effectively use insulin. It has millions of victims worldwide and has emerged as a significant public health issue. The gruesomeness of diabetes lies in its potential to result in complications that could be fatal if not treated properly. Its far-reaching and terrible consequences, such as heart disease, kidney problems, nerve damage, and blindness, underscore the urgency of early detection and management. To deal with this pressing issue, the purpose of this research is to develop an accurate and efficient diabetes prediction system. By leveraging advanced machine learning techniques, the objective is to extract informative patterns and risk factors related to diabetes from diabetic healthcare datasets through analysis. The ultimate goal is to create a reliable and automated diagnostic system that may aid medical professionals identify diabetes early, leading to timely interventions and improved patient outcomes.

A variety of conventional machine learning algorithms are used in the research, such as Logistic Regression, Support Vector Machines, K-Nearest Neighbors, Decision Tree, Random Forest, and boosting classifiers, such as Xgboost, LightGBM, and CatBoost etc. Finally, the best-performing classifiers are taken for Voting Classifier and Stacking Classifier to merge the best meaningful

patterns captured by individual models. These algorithms are trained on a famous diabetes dataset, PIMA Indian Diabetes Dataset containing essential clinical information related to diabetes cases.

Furthermore, the insights gathered from this research will pave the way to establish data-driven decision support systems and contribute to advancing medical study and patient care in the field of diabetes diagnosis. Ultimately, the proposed machine learning approach possesses the potential to lessen the burden of diabetes and improve the general health and well-being of those who are suffering from this condition.

2. Introduction:

Diabetes is a long-term metabolic disorder that affects how our body regulates blood sugar levels. It is one of the most rapidly growing health issues of the 21st century. According to the report of IDF Diabetes Atlas 10th edition (IDF, 2021), Diabetes affects 537 million people (20 to 79 years old). By 2030, this number is expected to reach 643 million, and by 2045, it will reach 783 million. In 2021, 6.7 million people die from it, each one every five seconds. Around 90–95 percent of individuals globally have Type 2 diabetes. (Panwar et al., 2016)
In medical diagnosis (WebMD, 2023), Diabetic predictions through doctors involve evaluating a combination of the disease's symptoms, medical history, and laboratory tests. After an individual has gone at least 8 hours without eating, a fasting plasma glucose test analyzes their blood glucose levels. However, those methods are time-intensive, costly, and risk introducing human errors. Here, Machine learning algorithms offer an easier and more trustworthy approach to diagnosing diabetes by reducing processing time and enhancing prediction accuracy.
Traditional machine learning techniques including logistic regression, SVM, KNN (Patra et al., 2021), decision trees, and random forests (Yuvaraj et al., 2019) have been widely applied in the field of diabetes prediction. They made an initial breakthrough in medical diagnosis from the expertise of doctors in performing human detections. However, these algorithms may not fully capture the complexity of the disease and suffer from limitations such as feature selection bias and lack of adaptability to changing data patterns.
To address these limitations, boosting algorithms have shown promising results by capturing a wider range of patterns and improving prediction accuracy. However, such boosting algorithms as Xgboost, LightGBM, and CatBoost can perform better than the previous traditional ones. They can use ensemble approaches to capture correlations between features and the target variable. They can uncover intricate patterns in the data and deliver better predictive performance compared to simpler algorithms.
In this research, a stacking algorithm and voting classifier are introduced that combines the predictive capabilities of multiple boosting algorithms, including AdaBoost, Gradient Boosting, LightGBM, XGBoost and CatBoost. By leveraging the strengths of each algorithm, the stacking approach aims to capture diverse patterns and enhance overall prediction accuracy. NuSVM classifier was used as the final estimator of this stacking algorithm. The voting classifier aggregates each base model's predictions and chooses the majority vote as the final prediction. This can help to mitigate the biases or errors that may be present in individual models and result in more accurate and reliable predictions. Each of these boosting algorithms may have strengths and weaknesses in catching different patterns in features. By combining them through voting and stacking

approaches, the overall predictive power can be enhanced, leading to better performance in diabetes prediction.
A comprehensive evaluation of the proposed method on a famous dataset of diabetic patients Pima Indians Diabetes Dataset which includes features such as age, glucose level, number of pregnancies, blood pressure, body mass index, and skin thickness. The performance evaluation of the diabetes prediction model encompassed a thorough analysis. GridSearchCV was employed to fine-tune hyperparameters, optimizing the model's predictive capabilities. Confusion matrices were used to visualize true positive, true negative, false positive, and false negative results. Essential evaluation metrics like accuracy, recall, precision, and F1-score were calculated to gauge the model's overall effectiveness. The learning curve illustrated the model's convergence and assessed its potential for further improvement. The ROC curve and area under the curve (AUC) quantified the model's ability to distinguish between classes. Furthermore, the importance of features was determined, shedding light on variables impacting predictions. The model's interpretation was deepened through SHapley Additive exPlanations (SHap) plots, enhancing the understanding of individual predictions. These evaluations collectively demonstrated the model's robustness and its potential to accurately predict diabetes outcomes.

The research aim and main contributions of this study are listed below:

- To address the urgency of early diabetes prediction in the face of its escalating global impact.

- To advance the field of medical diagnosis through the application of efficient machine learning algorithms.

- To enhance the predictive capabilities of boosting algorithms, particularly XGBoost, LightGBM, and CatBoost.

- To introduce an ensemble learning strategy, utilizing stacking and voting classifiers, for robust and improved prediction accuracy.

- To validate the performance of the proposed approach using the Pima Indians Diabetes Dataset, demonstrating superior results over traditional methods.

This highlights the effectiveness of my approach in making robust diabetes predictions and boosting the overall predictive performance in this critical healthcare domain.

3. Related Works:

Many researchers have extensively utilized the PIMA dataset for conducting studies on diabetic prediction.

In paper, Kalpana and Kumar (Kalpana et al., 2011) developed a fuzzy expert system that evaluated the PIDD (PIMA Indian Diabetes Dataset) for diabetes prediction. Their strategy involved transforming data into fuzzy values, utilizing fuzzy associations, and employing a rule-based system. The method functioned more accurately and efficiently than earlier techniques.

Dewangan and Agrawal (Dewangan et al., 2015) proposed a diabetes diagnosis system that relies on multilayer perceptron and Bayesian classification. There were three phases to the study: feature selection techniques, individual classification models, and ensemble models for improved accuracy. The system achieved an accuracy of 81.89% using the PIDD dataset. The researchers concluded that their model obtained high accuracy with fewer features which is much more beneficial.

Daghistani and Alshammari (Daghistani et al., 2020) compared random forest and logistic regression for diabetes prediction using a dataset taken from a Saudi Arabian hospital. Random forest achieved 88% accuracy, outperforming logistic regression (70.3%).

In this study (Tan et al., 2022), to increase the accuracy of diabetes risk prediction, a GA-stacking ensemble learning model is suggested. Genetic algorithms select a subset of attributes suitable for prediction, and optimized CNN and SVM models are performed as primary learners. The results show that GA-stacking outperforms other models in terms of prediction accuracy and efficiency in early-stage diabetes risk prediction.

This paper (Yahyaoui et al., 2019) introduced a Decision Support System for diabetes prediction using machine learning techniques. The study compares conventional machine learning methods such as SVM and Random Forest with deep learning approaches (CNN) on the PIMA dataset. The results show that RF performs better in diabetes prediction compared to deep learning and SVM methods, achieving an overall accuracy of 83.67%.

In the paper, Big data in medical diagnosis aids disease prediction using Grey Wolf optimization (GWO) and Recurrent Neural Network (RNN) (Babu et al., 2018). It selects relevant features with GWO and achieves better performance than existing methods, with higher accuracy, sensitivity, and specificity on the PIMA dataset.

The paper (Chen et al., 2021) presents Task-wise Split Gradient Boosting Trees (TSGB), a new method for multi-center diabetes prediction. By combining gradient-boosting decision trees and multi-task learning, TSGB adopts challenges of data insufficiency and heterogeneity. It achieves superior performance and has been deployed also as an online diabetes risk assessment software tool for early diagnosis.

The study (Madan et al., 2022) presents a real-time monitoring and hybrid deep learning-based model for detecting and predicting Type 2 diabetes mellitus. A comparative study of different deep learning models is carried out, and a CNN-Bi-LSTM model is proposed as the best-performing method than renowned others with high accuracy, sensitivity, and specificity. The model enables healthcare professionals to obtain comprehensive real-time information about patients and obtain real-time statistics on their vital signs.

The research (Panwar et al., 2016) demonstrates a new way for accurate Type 2 diabetes detection using novel preprocessing techniques and the K-nearest neighbor classifier. The methodology achieves good classification accuracy by reducing features from eight to two on the PIDD dataset, surpassing existing methods.

The paper (Ramesh et al., 2017) proposed an intelligent predictive model using deep learning to predict the risk factor and severity of diabetes. It consists of a deep neural network with a Restricted Boltzmann Machine for diabetic data analysis. Comparative studies are conducted, showing that the model's improved accuracy, recall, and precision in diabetes prediction compared to existing methods.

The paper (Srivastava et al., 2019) used machine learning, specifically artificial neural networks (ANN), to develop a diabetes prediction model. It focuses on a specific female patient group in India (PIDD) and achieves a decent accuracy in predicting the possibility of diabetes which brings a sense of trust and dependence on their model for accurate prediction of diabetes.

The paper (Jakhmola et al., 2015) introduced a new data smoothening technique for noise removal in diabetes data. The method allows to control the level of smoothening based on the acceptable loss percentage. The algorithm provides interactive data preprocessing and enhances output quality. Correlation and multiple regression are then applied to the preprocessed dataset which has a significant impact on the prediction of diabetes, allowing for more reliable and accurate results.

The study (Osman et al., 2017) proposed a combined SVM and K-means clustering approach to improve the accuracy of diabetes diagnosis. The experimental results demonstrate high accuracy in differentiating between diabetic and non-diabetic patients with their clustering techniques. The T-test statistical method also shows significant improvements compared to modern diagnosis methods on the UCI Pima Indian dataset.

The paper (Soliman et al., 2014) proposed a hybrid algorithm combining Modified-Particle Swarm Optimization and Least Squares-Support Vector Machine (LS-SVM) for the classification of type II diabetes mellitus patients. The algorithm optimizes LS-SVM parameters to improve classification accuracy.

The paper (Sridar et al., 2014) addressed the importance of early detection of diabetes and proposed a system that uses effective data mining and machine learning techniques to diagnose diabetes based on clinical data. Glucometer readings and other parameters are used to feed their system and employed BP Algorithm (ANN) and Apriori Algorithm (ARM) for diagnosis. The proposed system aims to provide online access for patients to monitor their diabetes risk without the need for doctors.

Olaniyi and Adnan (Olaniyi et al., 2014) proposed a system for diabetes prediction using Artificial Neural Network (ANN) on the PIDD dataset for training. A multilayer feed-forward network was created and trained using the backpropagation algorithm. The system achieved a recognition rate of 82% on the test, outperforming other algorithms such as ADAP, C4.5, backward sequential selection, and EM.

This paper (Maniruzzaman et al., 2017) introduced Gaussian process classification (GPC) as a technique for diabetes classification in machine learning. GPC utilizes three types of kernels: linear, polynomial, and radial basis kernel. The paper compared GPC with existing classification techniques such as LDA, QDA, and NB, and demonstrates that the GPC-based model achieves the highest accuracy, sensitivity, specificity, and other performance parameters. The study highlights the usefulness of machine learning systems for diabetes data classification, considering the significant impact of diabetes as a global health issue.

The paper (Bansal et al., 2020) proposed an ensemble of non-linear support vector machines (nl-SVM) and a partial least square classifier for improved diabetes prediction. It outperforms other classifiers and ensembles in terms of accuracy and performance as getting benefits from nl-SVM kernel transformations and PLS-reduced dimensionality.

This paper (NirmalaDevi et al., 2013) presented an amalgam model combining k-means clustering and k-Nearest Neighbor (KNN) algorithm for classifying PIDD. The model improves data quality, removes noise, and enhances the accuracy and efficiency of KNN. Experimental results show high classification accuracy with larger k values. The model beats basic KNN, as well as cascaded K-MEANS and KNN.

This article (Sainte et al., 2019) reviewed and compared various Machine Learning (ML) and Deep Learning (DL) techniques used for predicting diabetes. It demonstrates the superior accuracy of a combined CNN-LSTM model. Additionally, the study explores rarely used ML classifiers on the Pima Indian Dataset and suggests their application in diabetes prediction, along with the development of combined models for further improvement.

This paper (Ashiquzzaman et al., 2018) presented a reliable diabetes prediction system using a dropout method to address data overfitting. A deep learning neural network with these dropout layers exceeds existing approaches in properly predicting diabetes.

This paper (Rakshit et al., 2017) focused on predicting diabetes using a Two-class Neural Network. The model was run on the PIDD, and its performance was compared to that of other outcomes. The predictive model can help medical professionals predicted a patient's risk of getting diabetes.

4. Pre-Processing:

4.1 Dataset Description and Analysis:

Pima Indians Diabetes dataset description:

Feature Name	Feature Description
Pregnancies	No. of times a woman has been pregnant
Glucose	Blood sugar level after an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	measure of body fat(mm)
Insulin	serum insulin (mu U/ml)
BMI	measure of body fat based on height and weight (weight in kg/(height in m)^2)
DiabetesPedigreeFunction	Diabetes pedigree function
Age	person's age (years)
Outcome	Whether the person has diabetes (1) or not (0)

The distribution and boxplot of features in the dataset are shown below:

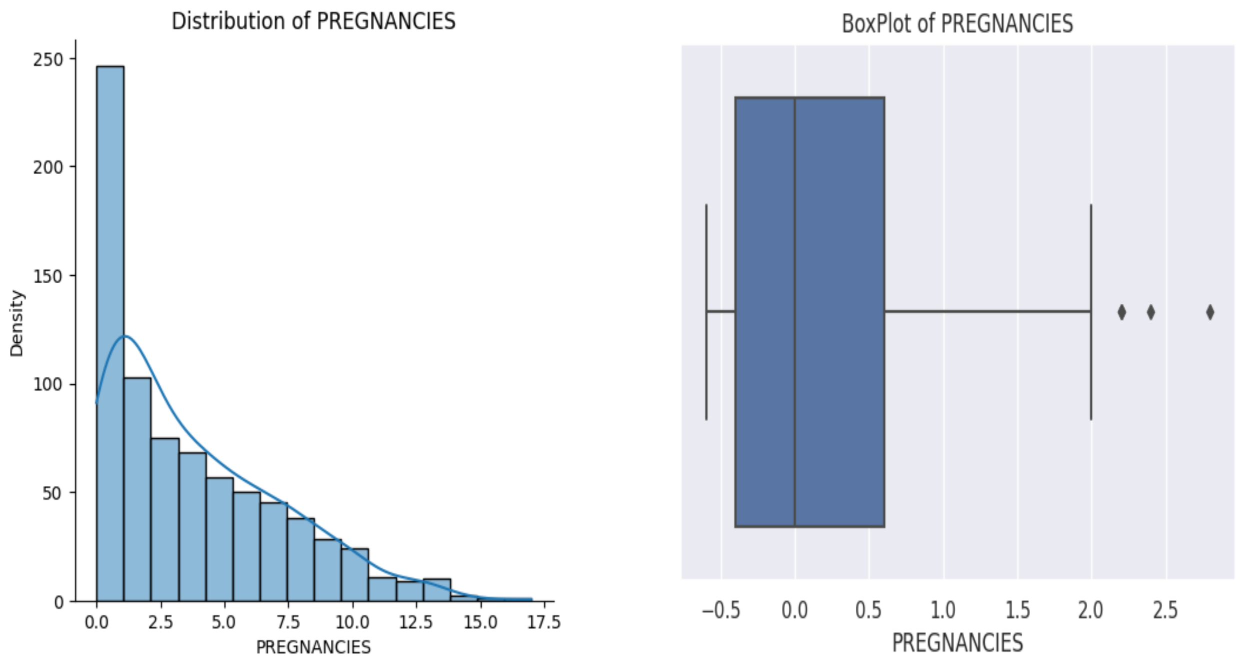

Figure 1: Distributions and Box Plot of Pregnancies Feature

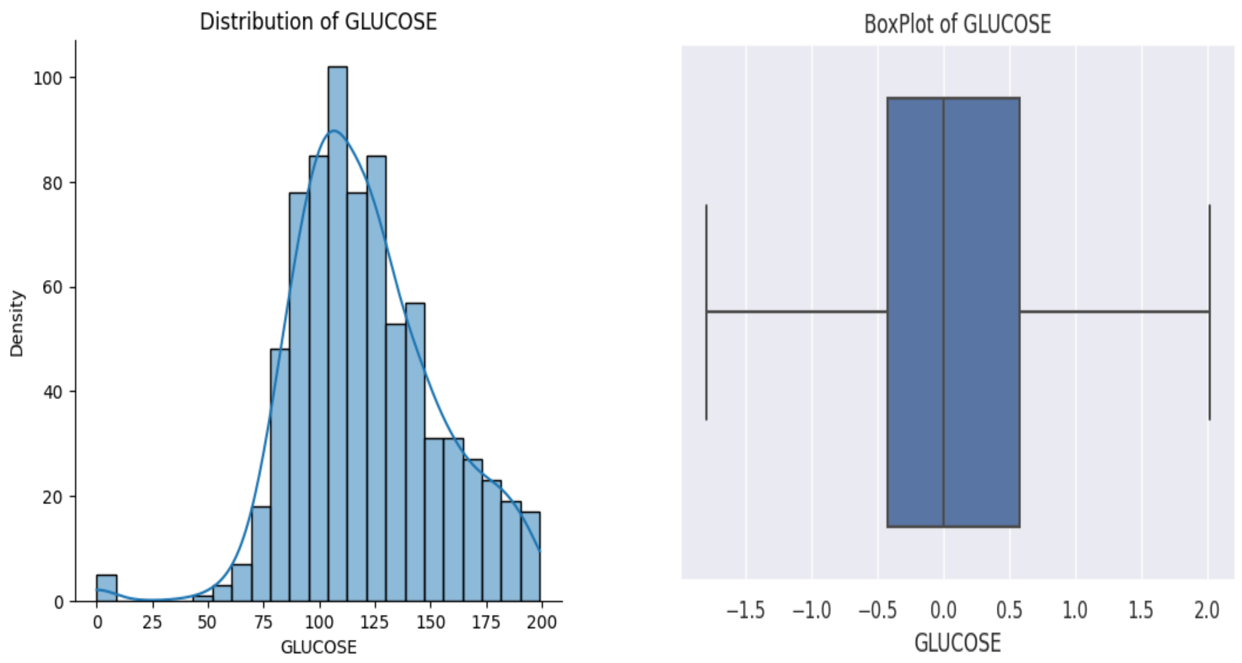

Figure 2: Distributions and Box Plot of Glucose Feature

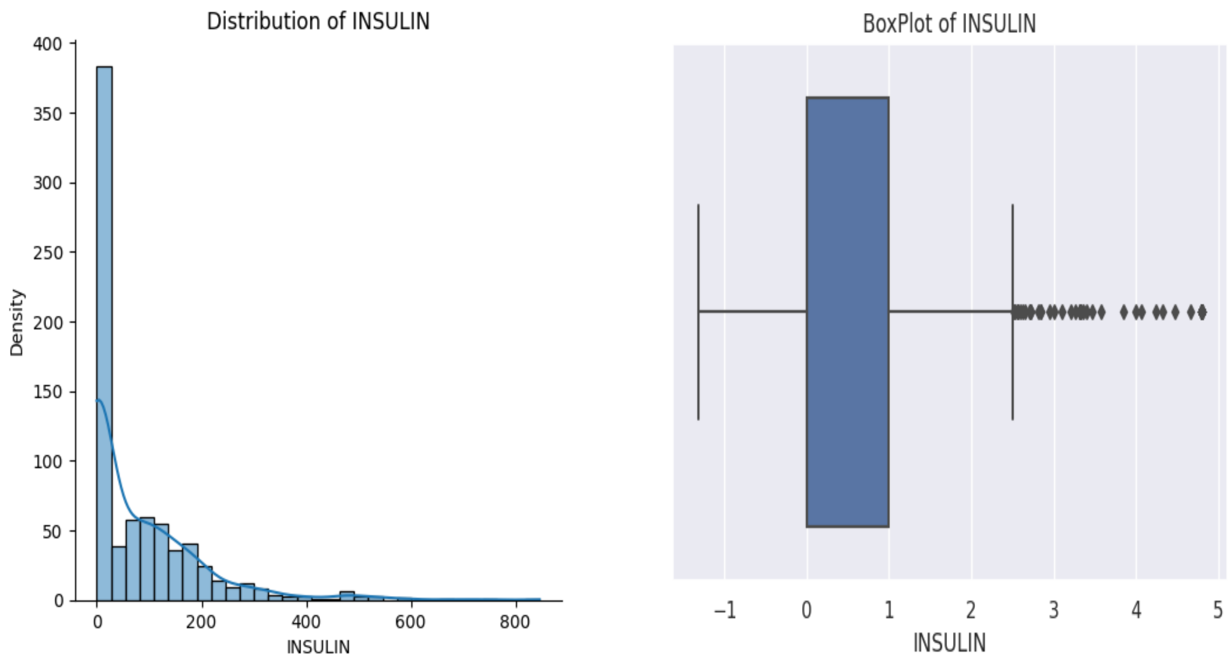

Figure 3: Distributions and Box Plot of Insulin Feature

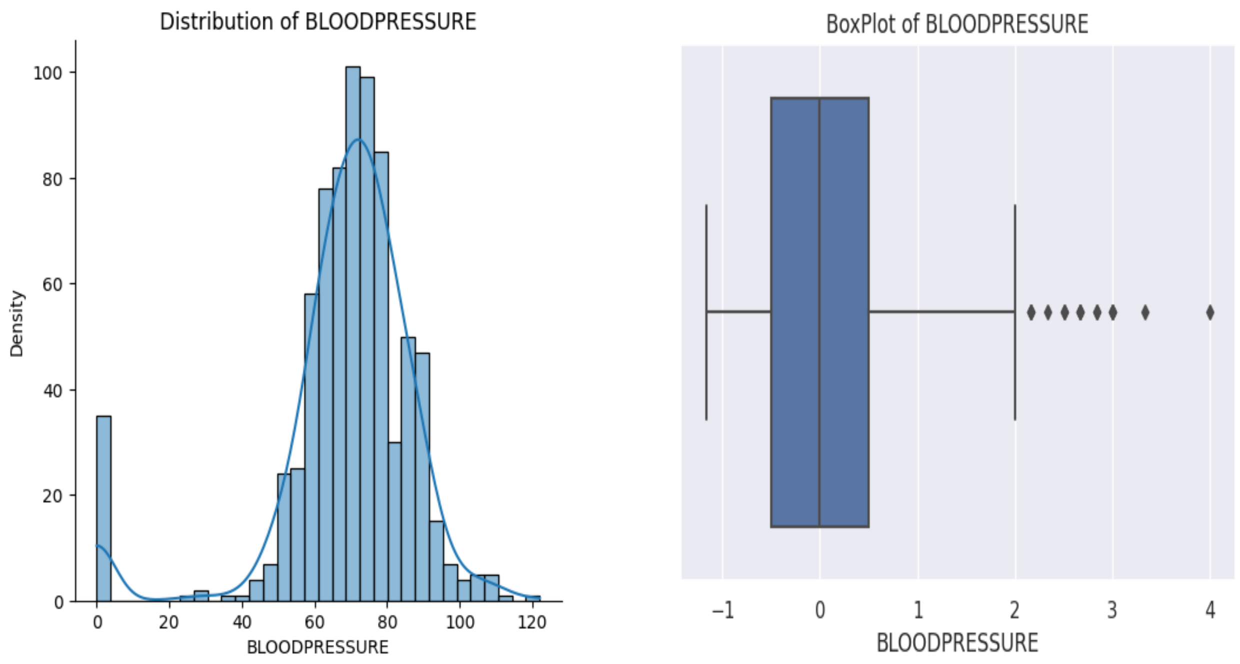

Figure 4: Distributions and Box Plot of Blood Pressure Feature

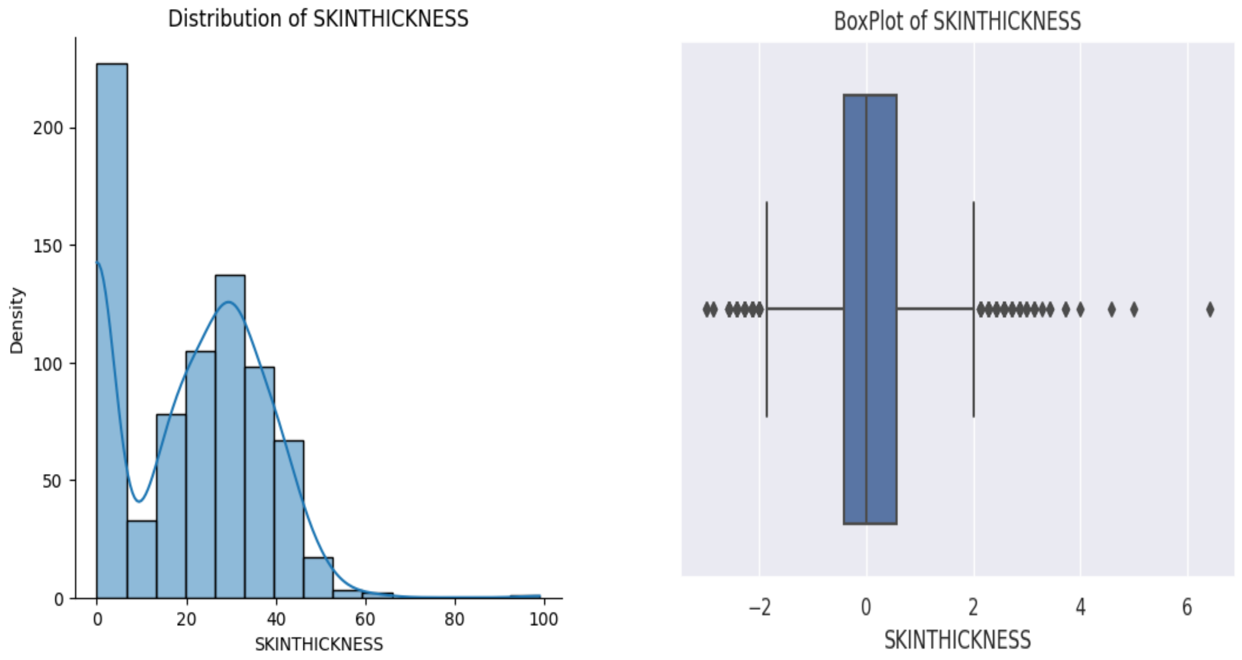

Figure 5: Distributions and Box Plot of Skin Thickness Feature

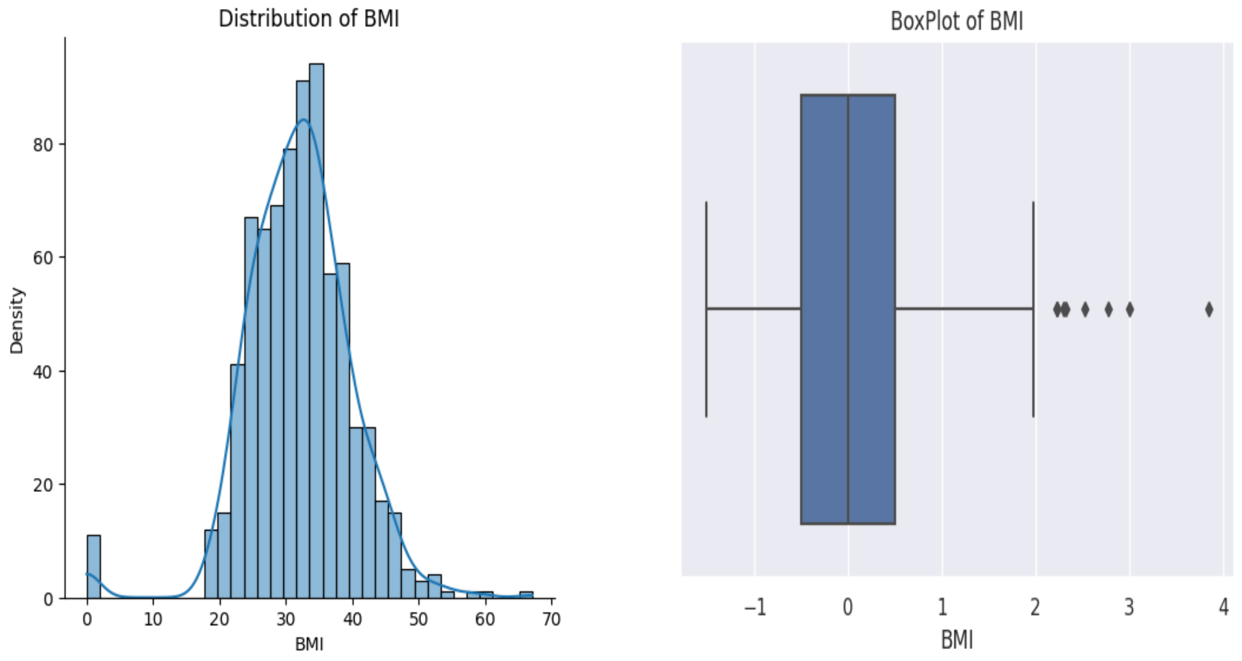

Figure 6: Distributions and Box Plot of BMI Feature

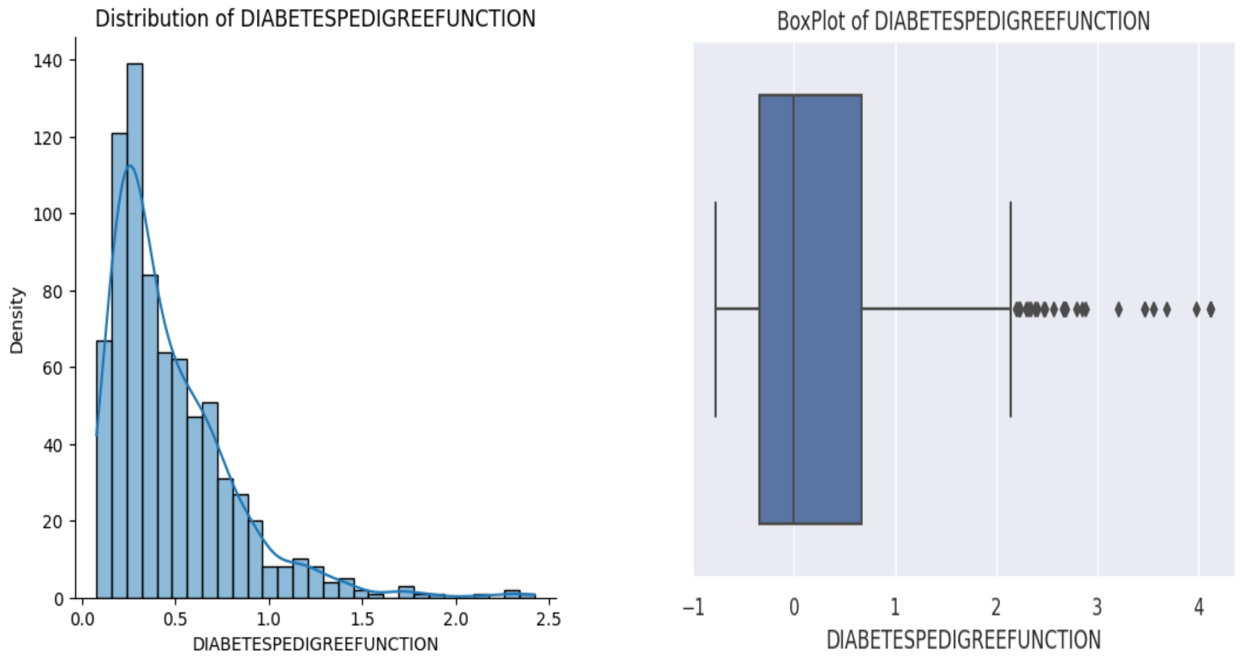

Figure 7: Distributions and Box Plot of Diabetes Degree Function

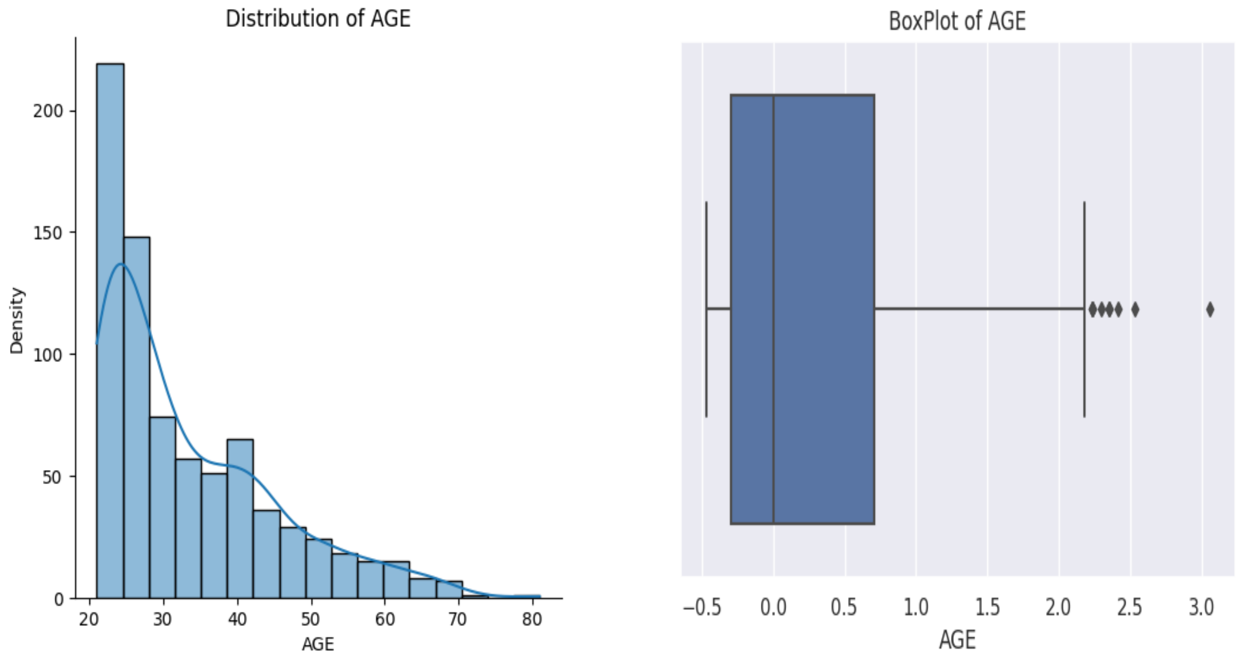

Figure 8: Distributions and Box Plot of Age Feature

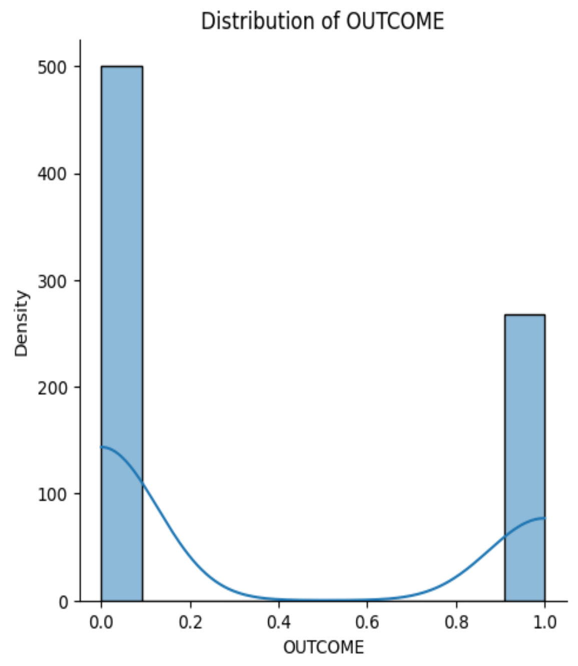

Figure 9: Distributions of Outcome

Feature analysis through distribution and boxplot:

Pregnancies: Pregnancies distribution graph is rightly skewed, presence of an outlier on the right. By nature, the graph is unimodal and the shape is asymmetrical. In most cases, a woman was pregnant once or none. But there can be seen cases of more than 13 times pregnant cases, which is highly unlikely and indicates possible outlier cases.

Glucose: The glucose feature's distribution is unimodal, slightly right-skewed, and similar to symmetrical. It depicts that most blood sugar level is between 100 and 125

Insulin: Highly right-skewed distribution. Most of the cases are below 200 mu U/ml insulin level. Asymmetrical graph. Peak from the start, and gradual decrease later.

Blood Pressure: Slightly bimodal. At first gradual decrease can be seen from the left. Then again rise to the highest peak. High density from range 70 mm Hg to 80 mmHg. But the cases left of the peak, especially the range from 0 to 30 is a highly unlikely, possible outlier situation.

Skin Thickness: By shape and nature it is bimodal. Presence of two peak points. A bit right-skewed and has less presence of an outlier on the right. A slight anomaly can be seen. A huge surge of density is from 0 to 5 mm and then a sudden drop after 5. Afterward, a bell-shaped curve can be seen with peaks ranging from 20 mm to 40 mm.

BMI: Representing the most symmetrical of all distribution curves of the features. By nature, is unimodal. It is a bit leptokurtic graph. Peak density is in the range of 25 to 35. A small isolated surge at the beginning indicates an unrealistic outlier.

Diabetes Pedigree Function: Highly right-skewed. Gradually decreases from the left. Unimodal by nature and a high presence of outliers on the right. Peak density ranges from 0.3 to 0.5.

Age: Right skewed, less presence of outliers. High-density peaks around 25 years. Unimodal distribution by nature.

Outcome: Bimodal distribution contains only 2 possible outcome values. 0 indicates "No" whereas 1 indicates "Yes". Peak density can be seen in the "No" output prediction. As it is a binary class, it doesn't need boxplot analysis.

4.2 Feature Correlation:

The correlation between the features is shown below:

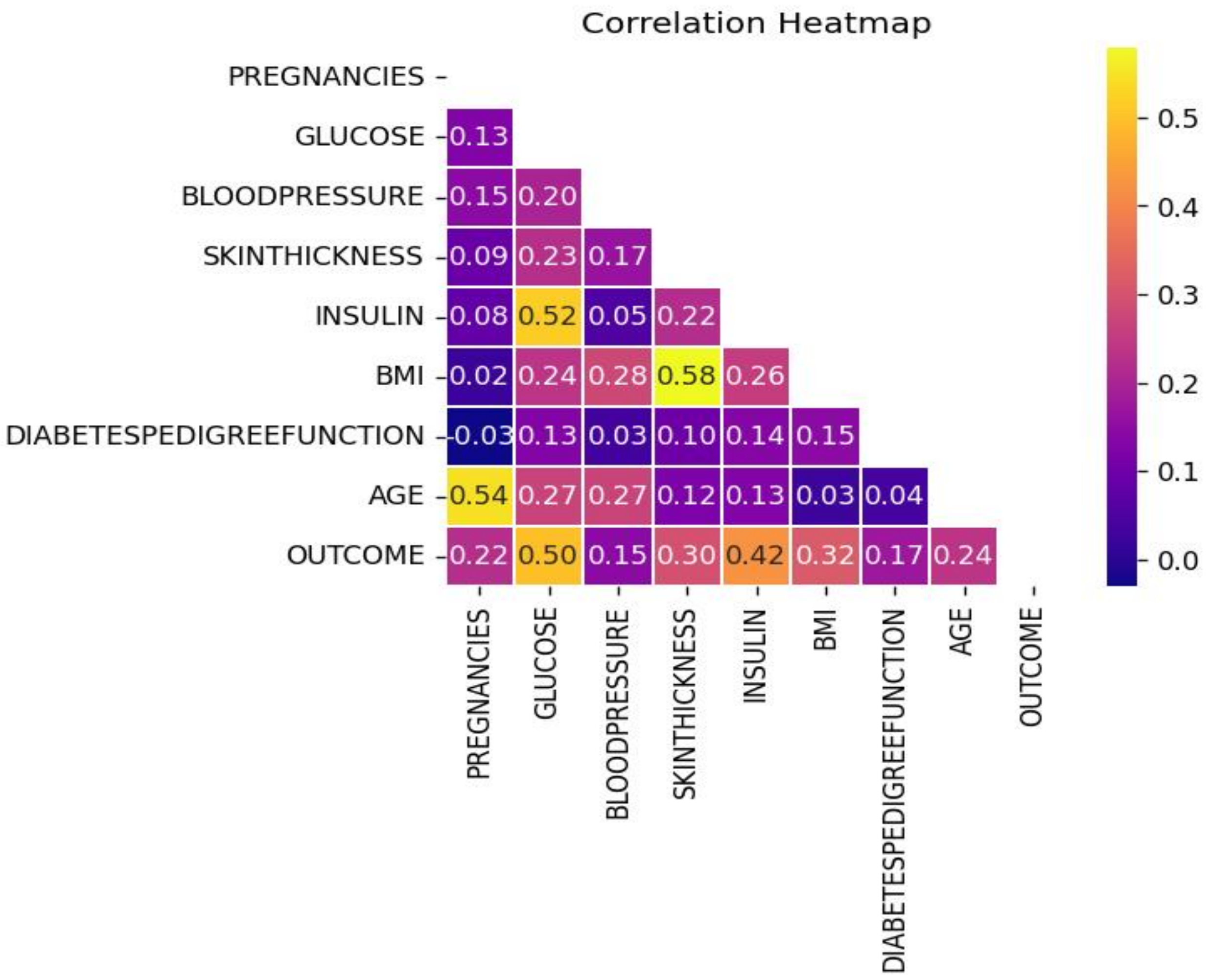

Figure 10: Correlation between features of the PIMA dataset

The correlation heatmap visually represents the correlation matrix in our dataset. Warm colors (yellow) indicate positive correlations, while cool colors (blue) signify negative correlations, and white represents no correlation. According to the following correlation graph based on the PIMA dataset, we can understand a strong positive correlation between skin thickness and BMI (0.58), which means it changes linearly. Such positive correlation can be seen among insulin and glucose, age and no of pregnancies, glucose, and insulin. On the other hand, a strong negative correlation can be seen between BMI and pregnancies, and diabetes pedigree function (indicates inverse relation). The two features are also having a similar negative correlation with age. Correlation with the highest value is 0.58 (BMI and skin thickness) whereas the lowest is 0.02 (BMI and pregnancies). Finally, in the case of the Outcome feature, we can see that BMI, Glucose level, and Insulin show a significant correlation in diabetes detection than others.

4.3 Data Preprocessing:

There are multiple preprocessing techniques applied for this diabetic diagnosis on the PIMA dataset which are described below:

Drop or replace:

The Pima Indians Diabetes Database, which offers crucial information about diabetes in the population, was extensively examined for this inquiry. To deal with missing or incorrect numbers, the drop or replacement data is used in this study. Particularly individuals with a blood pressure under 60 and a BMI under 12 are considered outliers or improbable entries. The dataset's integrity is then maintained by replacing these instances with zero. The prevalence and features of diabetes among the Pima Indians are better-understood thanks to this method. The dataset is carefully scrutinized.

Missing values fill up:

The missing values in the dataset were addressed by filling them with the corresponding median values based on the outcome category. This approach was chosen because there were variations in the median of each variable depending on the outcome, making it a more suitable method. Additionally, due to the large number of missing values, it was not feasible to remove them from the dataset. However, using mean, KNN imputer, or MICE for filling in missing values also has drawbacks. The mean might not accurately represent the distribution of data, leading to biased imputations. KNN imputer could be computationally expensive and may not perform well with high-dimensional data. MICE, while effective, might be sensitive to extreme values and require careful handling of the imputation process. Despite these limitations, the choice of median imputation proved to be a practical and appropriate approach for handling missing values in this context.

Handling Outliers:

By analyzing the data, outliers are detected using quadratic interpolation involving fitting a quadratic curve to neighboring data points, providing more accurate estimates for potential outliers. In machine learning, the process of locating and managing data points that significantly differ from the rest of the data is known as outlier analysis. The performance and accuracy of machine learning models can be adversely impacted by outliers, which are data examples that are very different from the bulk of the data points. Outliers can occur for several causes, including data collecting problems, measurement noise, or uncommon occurrences. The goal of outlier analysis in machine learning is threefold:

- **Detection**: Identify outliers using statistics (z-scores, IQR) or visual exploration (scatter plots, box plots).
- **Handling**: Address outliers by removal, capping, or robust models.
- **Impact Assessment**: Evaluate outlier effect on models, they may carry insights or indicate anomalies, consider context and domain knowledge.

IQR method has been used here which benefits outlier analysis by identifying data points deviating from the overall trend, improving accuracy, and handling nonlinear relationships for better machine learning model performance.

Binning:

The PIMA Indians Diabetes (PID) dataset feature extraction process uses binning to convert continuous numerical values into discrete categories. In particular, glucose levels are classified as "Hypoglycemia," "Normal," "Impaired_Glucose," and "Diabetic_Glucose" to make it possible to examine the connection between glucose levels and the risk of developing diabetes. Age is classified as "Young," "Young Adult," "Adult," and "Old," making it easier to examine age-related trends in the prevalence of diabetes. Additionally, the number of pregnancies is divided into "Never," "One Time," and "Many Times," allowing researchers to examine the relationship between a history of pregnancies and the chance of developing diabetes.

These categorized features improve the study by illuminating how age groups, blood glucose levels, and previous pregnancies affect diabetes incidence. They make it possible to identify risk variables linked to particular glucose ranges, age groups, and pregnant statuses, adding to our understanding of diabetes overall in the dataset.

Feature Encoding:

On the dataset of 768 female diabetic patients from the PIMA Indians Diabetes (PID) study, one-hot encoding was used. Binary features are created from category variables. A categorical column's categories are transformed into independent binary columns. Examples of characteristics that are encoded include the number of pregnancies, glucose levels, blood pressure, triceps thickness, insulin levels, BMI, pedigree function, and age.

With one-hot encoding, the dataset is increased by adding new columns for each distinct category in categorical variables. One-hot encoding enables the study and use of categorical information in machine learning models by describing categories as binary features. It improves the dataset's applicability for tasks like estimating the prevalence of diabetes and facilitates analysis and interpretation because it can offer a thorough representation of categorical data. Overall, one-hot encoding enhanced the PIMA Indians Diabetes dataset's compatibility with machine learning methods and proved itself useful in this case.

Feature Scaling:

Robust Scaling is used for feature scaling. In machine learning, feature scaling is a crucial preprocessing step that seeks to uniformly scale numerical features to allow for fair comparison. Two scaling methods are used with the PIMA Indians Diabetes dataset. The numerical columns are subjected to the Robust Scaler, which divides by the interquartile range and subtracts the median. Because of its resistance to outliers, this approach is appropriate for datasets having extreme values. Scaling the numerical features makes them comparable and equally important in the machine learning models' learning process.

In addition, existing columns are subjected to mathematical procedures to produce new features. For instance, the "NEW_GLUCOSEINSULIN" feature is created by multiplying the "GLUCOSE" and "INSULIN" columns, while the "AGEBMI_NEW" feature is created by multiplying the "AGE" and "BMI" columns. By capturing potential correlations and interactions between variables, these designed characteristics provide the models with more data.

By using robust scaling, the dataset is prepared for machine learning models in a way that is less affected by outliers and maintains the integrity of the data distribution. It ensures that the numerical features are on a similar scale (in our case between 0 and 1), enabling fair comparisons and accurate analysis in the prediction of diabetes outcomes.

Pipelining:

While implementing, pipelining is used for data preprocessing techniques. Pipelining is the process of assembling various data transformation stages into a flowchart or pipeline. An input is given to each stage in the pipeline, which then processes it and sends the output to the following phase. The result of systematically applying each modification to the initial input data is the pipeline's final output. Here, each step of data preprocessing in the pipeline is executed in the order defined previously. And hence all the preprocessing steps are ordered and automated by using a pipeline. A Functional transformer and pipeline module were imported here in this case.

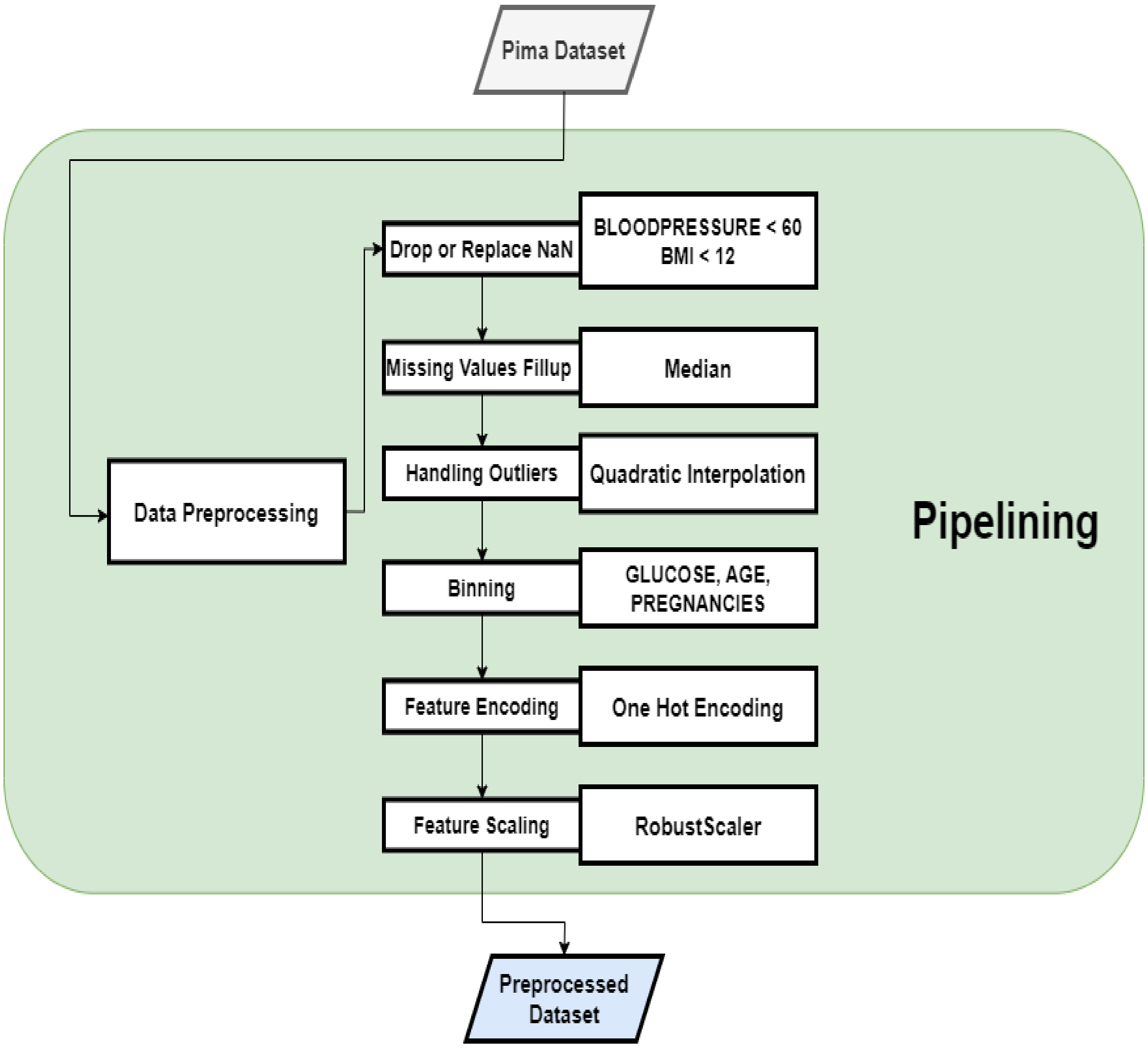

Figure 11: Dataset preprocessing steps performed in Pipeline

5. Methodology:

5.1. Model training & testing:

The dataset is split into train and test datasets such that the models are trained with 30% of the dataset and 30% used as test data. Then the data is passed to further processing for models like handling imbalanced data. After that the data is ready for training and the training data is fed into multiple models by using different machine learning algorithms which are Logistic Regression, SVC, KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier, XGBClassifier, GaussianNB, SGDClassifier, RidgeClassifier, BaggingClassifier, LGBMClassifier, CatBoostClassifier. After training the data with the above machine learning algorithm, all the best performer models passed into a Voting classifier algorithm and a Stacking algorithm, which is the process that combines the predictions of several learning algorithms.

5.2. SMOTE (Handling Imbalanced Datasets before training):

From the distribution of Outcomes, it can be seen that the target variable is imbalanced. There are more data on the normal patient than on the diabetic patient. To handle this, SMOTE is applied. SMOTE (Synthetic Minority Over-sampling Technique) is a popular machine learning method for dealing with the issue of class imbalance in datasets. Class imbalance happens when one class (often the minority class) has a disproportionately low number of samples compared to the other classes. To reduce such disproportionality and overfitting smote has been used to yield synthetic samples of the minority class to improve the accuracy of the model. By creating synthetic samples for the minority class, SMOTE solved this problem. The artificial samples are produced by interpolating feature vectors from nearby cases belonging to the same class. Through this procedure, the minority class, diabetic patient's data is essentially oversampled, bringing it closer to a balanced distribution with the majority class, normal patient's data and hence establishing the credibility of the model by improving its accuracy by a significant margin.

5.3. Algorithm description:

Support Vector Machine:

A support Vector Machine is used in the dataset for better classification. Support Vector Machine (SVM) is a powerful classification algorithm that finds a hyperplane in high-dimensional space to maximize the margin between classes. It adapts to data sets' separability: linear SVM for separable data and non-linear SVM for complex boundaries.

C and gamma are two crucial variables in the case of SVM

C: Classification error and margin maximization are balanced. High C emphasizes accurate categorization, which could result in overfitting. Low C favors a more broad-based model at the expense of some accuracy.

Gamma: Manages the impact of the data point on the boundaries of the decision. Low gamma makes the model more universal, whereas high gamma makes it more responsive to the local data structure.

SVM is effective in high-dimensional spaces. The dataset used in the research has nine features or dimensions. This number of dimensions is not high enough to make SVM the model (accuracy: 85.28) producing the best accuracy model. Other models are yielding better results than SVM.

Cross-validation and grid search methods are implemented to produce the hyperparameters yielding the best accuracy in SVM.

Logistic regression:

As the outcome of the dataset used in the research is binary (producing outputs whether the patient has Diabetes or not), Logistic regression is used. Logistic regression is a popular statistical and machine-learning approach for binary classification applications. It is used to forecast the likelihood that a particular instance will belong to a specific class. Contrary to its name, logistic regression is a classification algorithm. When doing logistic regression, the input characteristics are first merged linearly using weights before being processed through a logistic function (sigmoid function), which converts the continuous output into a probability score between 0 and 1. The instance is labeled as belonging to one class if the probability exceeds a predetermined threshold (often 0.5), and the other class if not. Maximum Likelihood Estimation is a technique used by the logistic regression algorithm during the training phase to learn the ideal weights. It aims to maximize the likelihood of the observed data being predicted correctly.

In the dataset used in the research, logistic regression gave us an accuracy of 0.79 which is quite low compared to other models used in this case. This comparative low accuracy gave an insight that the dataset is not well balanced and linearly separable. Though SMOTE analysis was utilized before to establish balance, and cross-validation techniques as well as grid search were implemented still the accuracy is not satisfactory indicating that lack of correlation between feature and linear separability to use logistic regression effectively.

KNN:

Here, K-nearest neighbors (KNN) is used to classify or predict an outcome by identifying the majority class of its k-nearest neighbors in the feature space. In our study, it measures the distance between the input sample and other training samples to determine the nearest neighbors. The value of k, a hyperparameter, determines the number of neighbors to consider. KNN can handle both categorical and numerical features and can be used for classification or regression tasks. It is a non-parametric algorithm and does not require explicit model training.

- n_neighbors: Determines the number of nearest neighbors considered for classification. Smaller values lead to more flexible models but may be sensitive to noise, while larger values may result in overly smooth decision boundaries.

Decision Tree:

In this study, Decision Tree algorithm is used to build a tree-like model to make predictions or decisions based on the set of input features. It recursively splits the data based on the feature that maximally reduces the impurity of the target variable. Here, each internal node represents a feature test, and each leaf node represents a predicted outcome. The algorithm selects the best feature at each step using metrics like information gain or Gini index. Decision trees can handle both categorical and numerical features and can be used for classification or regression tasks. They are interpretable and can handle missing values and outliers.

- Criterion: Chooses the quality measurement for node splits (gini or entropy).
- max_depth: Limits the depth of the decision tree, preventing overfitting.
- min_samples_split: Specifies the minimum samples required to split an internal node.

Random Forest:

The Random Forest algorithm plays an important role in this research that combines multiple decision trees to make predictions or decisions. It creates a collection of decision trees through bootstrap sampling and random feature selection. Each tree is trained independently on different subsets of the data. The final prediction is determined by aggregating the predictions of all the trees, either through voting (for classification) or averaging (for regression). In this work, it serves as a robust defense against overfitting, even in scenarios with high-dimensional data. It not only maintains predictive accuracy but also furnishes valuable insights into the importance of features.

- n_estimators: Determines the number of decision trees in the Random Forest ensemble. More trees improve performance but increase computational cost.
- max_depth: Sets the maximum depth of individual decision trees. Higher values can lead to overfitting, while None allows trees to grow until all leaves are pure.
- min_samples_split: Specifies the minimum number of samples required to split an internal node. Higher values reduce model complexity and overfitting.
- max_features: Controls the number of features considered for the best split at each node. Smaller values lead to less correlated trees and reduced overfitting.

XGBoost:

As the decision tree yielded an accuracy of 0.8398 before, XGboost was used to generate a better accuracy than the decision tree. XG Boost (Extreme Gradient Boosting) is a potent ensemble learning technique built on gradient boosting. To increase prediction accuracy, it iteratively combines weak learners, often decision trees. XG Boost continuously improved its forecasts by concentrating on the remaining errors and learning from the errors of earlier used models (such as decision tree). Overfitting is avoided by the regularization algorithm. Right control parameter was acquired by grid search and used later in the model to ensure the best possible result. The final accuracy generated by XG boost was .8831 which shows success in our attempt. The accuracy increased because the dataset used in the research is not small and XGboost avoided overfitting.

Ridge Classifier:

The utilization of the Ridge Classifier algorithm brings valuable benefits to the model's performance which mainly uses ridge regression to perform binary classification. It is a variant of logistic regression with L2 regularization, which helps to control the model complexity and prevent overfitting. Ridge Classifier optimizes a cost function that combines the log-loss and the L2 regularization term. It works by finding the weights that minimize the cost function and assigns class labels based on the decision boundary.

Its significance within our work stems from its ability to effectively manage a combination of categorical and numerical features. Moreover, its particular utility shines when addressing multicollinearity challenges in the input data. Notably, it boasts computational efficiency, interpretability, and excels when confronted with linearly separable data, thereby making it a pivotal asset in our endeavors.

- alpha: Controls L2 regularization strength (a higher value for stronger regularization)
- solver: Specifies a method for computing ridge coefficients (e.g., 'auto', 'svd', 'lsqr')

Bagging Classifier:

Bagging classifier gave us a slightly better accuracy than most other models used on our dataset. The ensemble learning method known as Bagging Classifier (Bootstrap Aggregating) enhances classifier performance and resilience. Across many applications, Bagging Classifier is frequently used to create reliable and precise classification models. By using replacement sampling, it divided the training data of the dataset into many subsets, each of which was used to train a different base classifier (typically a decision tree). Through majority voting (for classification) or averaging (for regression), Bagging Classifier combined the predictions of the base classifiers during prediction. As the dataset is quite large, variance of prediction may occur while using other models. The bagging classifier removed those variances, avoided overfitting, and hence improved the accuracy of our dataset.

CatBoost Classifier:

After implementing cat boosting in the dataset, we have got an accuracy same as the XGboost which is larger than most other methods use. Cat Boost Classifier handles category features without the need for manual encoding. To increase predictive accuracy and decrease overfitting, it makes use of gradient boosting and ordered boosting. Cat Boosting handled missing values in-house, improving robustness. To comprehend their impact, the algorithm produced categorical feature importance. Cat Boost is appropriate for handling huge datasets because of its efficient performance and parallelization of its methods. For the best outcomes, hyperparameter tuning was done properly and rigorously by implementing a grid search. Comparatively better accuracy after using the CatBoost classifier indicates that the used dataset is large, fairly complex, and contains categorical data.

SGD (Stochastic Gradient Descent) Classifier:

The accuracy of the SGD classifier is not satisfactory enough (81.39) compared to other models used on the dataset. It is partly because of the non-linear separability of the dataset.

SGD classifier is a type of linear classifier that uses stochastic gradient descent (SGD) to learn the model parameters. SGD is an iterative optimization algorithm that updates the model parameters one data point at a time. This makes the SGD classifier very efficient for training on large datasets.

As we have implemented SMOTE to eradicate the imbalance of the dataset, the relatively low accuracy of SGD indicates that the dataset is noisy and not linearly separable.

Voting Classifier:

After rigorous training of our dataset with various classifier models, the voting classifier was used to generate a combined prediction using five base classifiers to improve accuracy. The used base classifiers are Random Forest, XGBoost, Bagging classifier, LightGBM, and Cat Boost classifiers. A voting classifier is an ensemble learning method combining predictions of multiple base classifiers to make final predictions. Two types of voting classifiers are used here, hard voting and soft voting.

Hard voting classifier combines the predictions of multiple base classifiers to make a single prediction. The hard-voting classifier works by first training multiple base classifiers on the same training data. Then it asks each of the base classifiers to predict that data point whenever a new data point is presented. The hard-voting classifier then takes the majority vote of the base classifiers as its prediction.

The soft voting classifier combines the predictions of multiple base classifiers to make a single prediction. The soft voting classifier works by first training multiple base classifiers on the same training data. Then, when a new data point is presented to the soft voting classifier, it asks each of the base classifiers to predict that data point. The soft voting classifier then takes the weighted average of the predictions of the base classifiers as its prediction. The weights of the base classifiers are determined by the confidence of each classifier in its prediction.

In the research conducted, the accuracy of the final prediction, after implementing hard voting based on the previously stated classifiers, is 0.8788. Similarly, soft voting also gave us the same accuracy of 0.8788. In this case, the final prediction is made with the highest average probability. The accuracy after implementing a voting classifier is less than each of the base classifiers used.

Stacking Classifier:

In this study, a stacking algorithm is employed which merges predictions from multiple learning algorithms.

Stacking, an ensemble learning technique, employs meta-learning to fuse various machine-learning algorithms. Base-level algorithms are trained on the entire dataset, and a meta-model is then trained using the outputs of all base-level models as inputs.

Here, the training dataset is trained on different classifiers (LogisticRegression, SVC,KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier, XGBClassifier,GaussianNB, SGDClassifier, RidgeClassifier, BaggingClassifier, LGBMClassifier, CatBoostClassifier) and the stack algorithm takes the best 5 predictions which are random forest, Xgboost, Bagging, Catboost, LightGBM classifier in this case, and passed their output into a meta classifier, NuSVM and finally give the final predictions.

Using a stacking classifier, the ultimate prediction attains an accuracy of approximately 89.18%. In the confusion matrix of the stacking classifier, notable values include 139 True Positives and 11 False Positives which is much less and which is a very important factor for medical datasets. Additionally, the Receiver Operating Characteristic (ROC) curve exhibits an area where about 88% falls under the curve, indicating a favorable outcome.

These observations collectively suggest that the stacking classifier is performing well, particularly in terms of correctly identifying positive instances, which is especially critical in medical contexts.

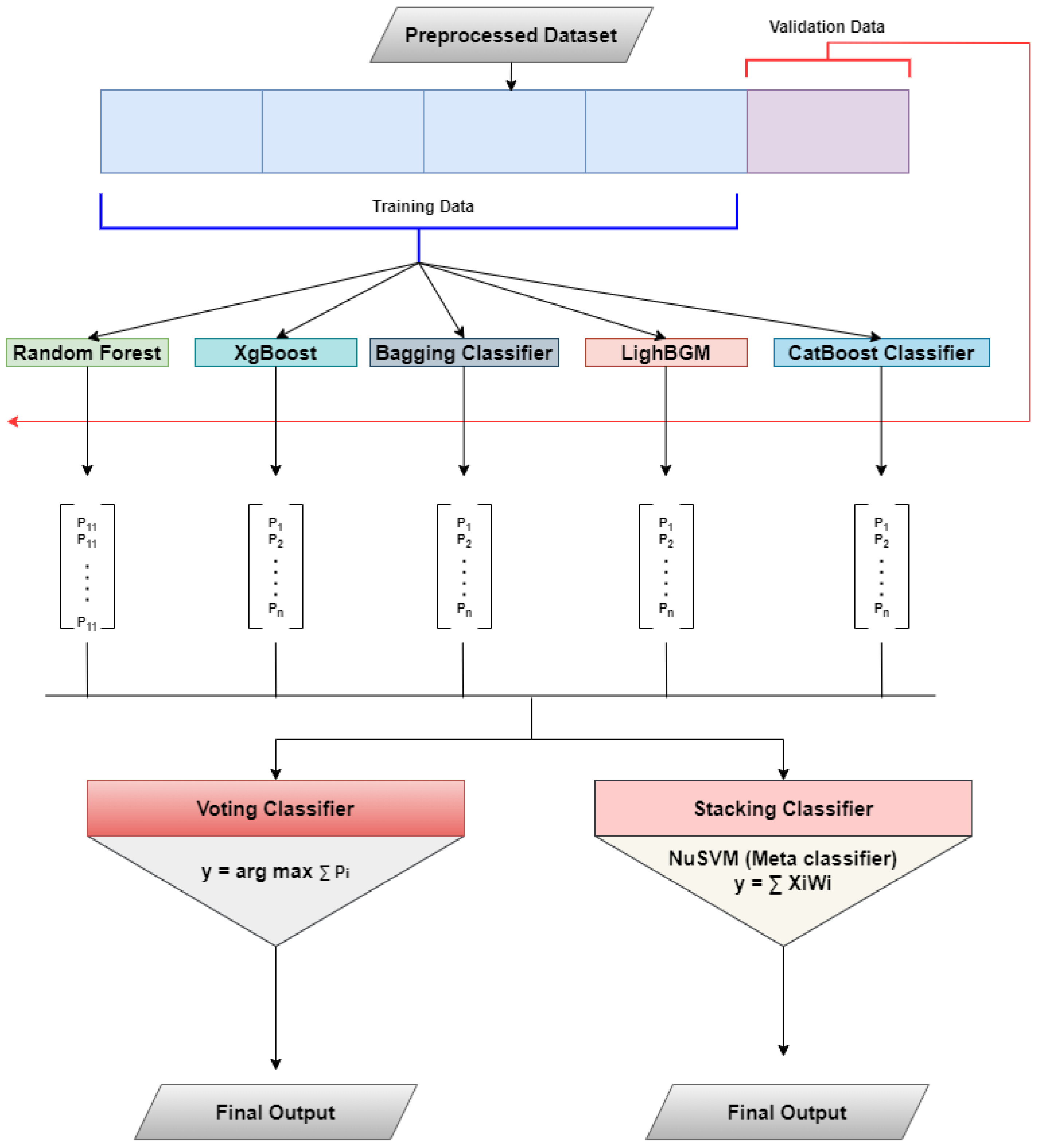

Figure 12: Stacking and Voting Classifier with best models.

5.4. Cross Validation Using Grid Search:

In this study, a common method in machine learning for evaluating a model's performance and generalizability is cross-validation. It aids in obtaining more accurate and trustworthy predictions of how well the model would function with hypothetical data. Cross-validation's fundamental premise is the division of the available data into numerous subsets or folds. Then, using various combinations of these subsets, the model is repeatedly trained and assessed. K-fold cross-validation, which divides the data into k equal-sized folds, is the most typical type. In k-fold cross-validation, the dataset is divided into k subsets, the model is trained on k-1 folds, it is then evaluated on the remaining fold, and the procedure is then repeated k times, rotating the validation fold each time. A more accurate evaluation of the model's generalization performance is produced by averaging the performance metrics over all iterations to produce the final evaluation score.

In the case of previously implemented code, the GridSearchCV function was used provided by the scikit learn library. Scikit-learns widely used hyperparameter tuning algorithm, called GridSearchCV, makes it easier to fine-tune machine learning models. In machine learning models, hyperparameters refer to those parameters which are chosen before the training process rather than ones that are learned from the data. Finding the ideal set of hyperparameters to enhance the performance of the model is known as tuning hyperparameters. By doing a thorough search over an established grid of hyperparameters, GridSearchCV automates this procedure. Each combination of the hyperparameters is tested for performance using cross-validation, commonly k-fold cross-validation, and the optimal performance is determined.

5.5. Algorithm parameters:

In the code, cross-validation is used in combination with GridSearchCV to evaluate the performance of multiple machine learning models that are used previously with different hyperparameter settings. This automated process helps identify the best combination of hyperparameters for each model and provides a more robust evaluation of their generalization performance as well as streamlining the process of selecting the best-performing model and hyperparameters.

The chosen hypermeters for each algorithm are shown below:

Algorithm	Best parameter
LogisticRegression	{'C': 0.1, 'penalty': 'l2'}
SVC	{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
KNeighborsClassifier	{'n_neighbors': 5, 'weights': 'distance'}
DecisionTreeClassifier	{'criterion': 'gini', 'max_depth': None, 'min_samples_split': 2}
RandomForestClassifier	{'max_depth': None, 'max_features': 0.3, 'min_samples_split': 2, 'n_estimators': 200}
XGBClassifier	{'colsample_bytree': 0.8, 'eta': 0.1, 'max_depth': 6, 'subsample': 0.7}
GaussianNB	{'var_smoothing': 1e-09}
SGDClassifier	{'alpha': 0.01, 'penalty': 'l1'}
RidgeClassifier	{'alpha': 0.1, 'solver': 'auto'}
BaggingClassifier	{'max_features': 0.7, 'max_samples': 0.9, 'n_estimators': 150}
LGBMClassifier	{'learning_rate': 0.05, 'max_depth': 5, 'num_leaves': 10}
CatBoostClassifier	{'depth': 4, 'iterations': 200, 'learning_rate': 0.1}

6. Result and Discussion:

In this study, the performance has been evaluated by many approaches. There showed an analysis of the confusion matrix, evaluation matrices, learning curve, and ROC curve to evaluate the implemented model performance. Then the feature selection and model interpolation are also analyzed to get a proper idea of a diabetes diagnosis.

6.1. Confusion matrix:

At first, the confusion matrices of all implemented model had explored. A confusion matrix is a table used to evaluate the performance of a classification model in machine learning, particularly in binary and multi-class classification problems. It provides a comprehensive summary of the model's predictions and their agreement with the actual ground truth.

In a binary classification confusion matrix, there are four important elements:

1. True Positives (TP): The number of instances correctly predicted as the positive class.

2. False Positives (FP): The number of instances incorrectly predicted as the positive class.

3. True Negatives (TN): The number of instances correctly predicted as the negative class.

4. False Negatives (FN): The number of instances incorrectly predicted as the negative class.

The confusion matrices of all implemented models are shown below:

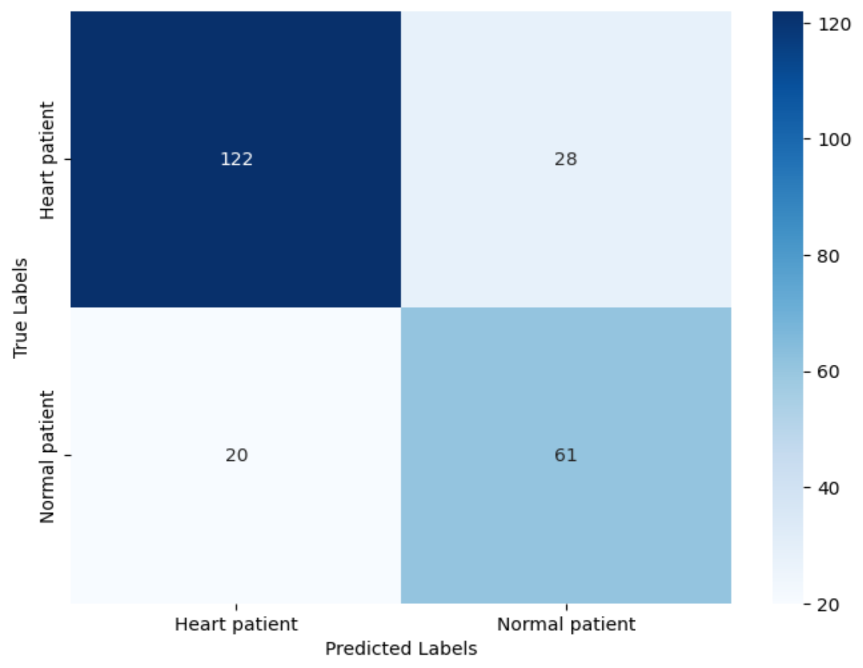

Figure 13: Confusion Matrix for Logistic Regression

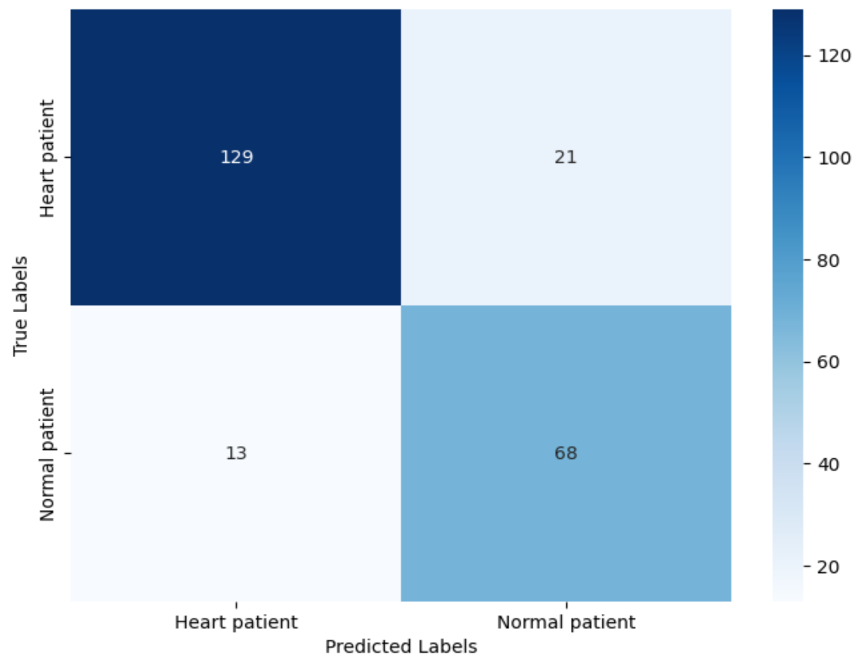

Figure 14: Confusion matrix for Support Vector Machine

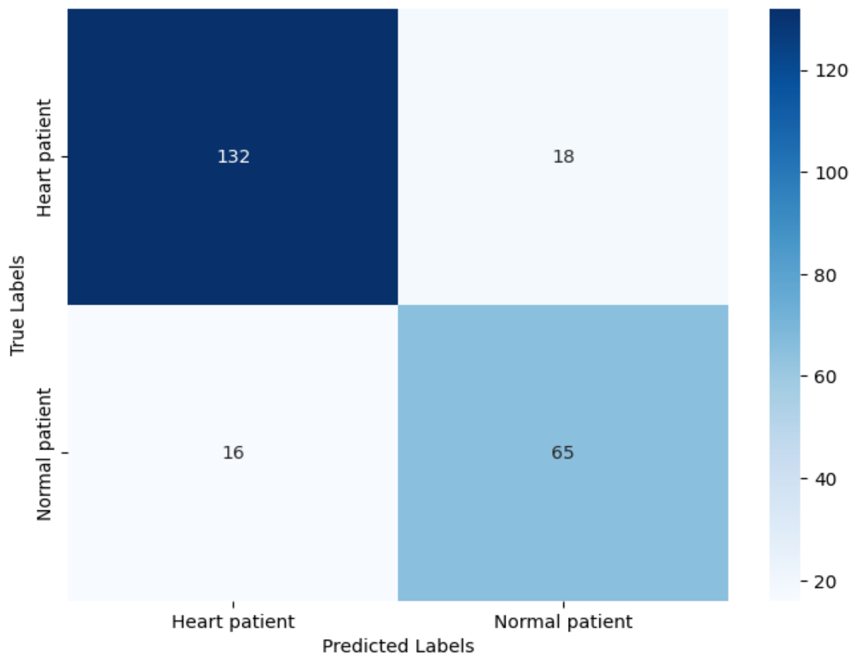

Figure 15: Confusion matrix for K-Nearest Neighbors

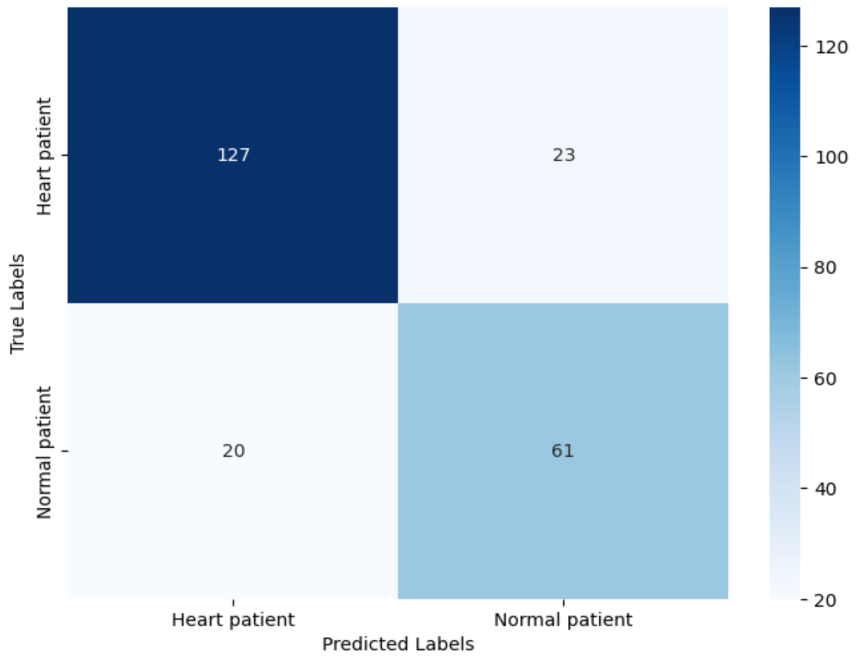

Figure 16: Confusion matrix for Decision Tree Classifier

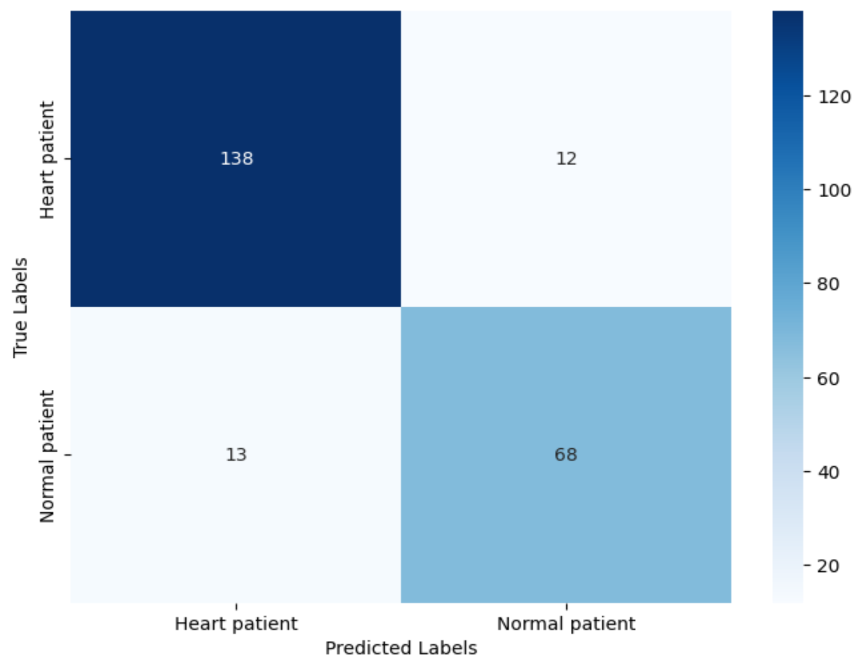

Figure 17: Confusion matrix for Random Forest Classifier

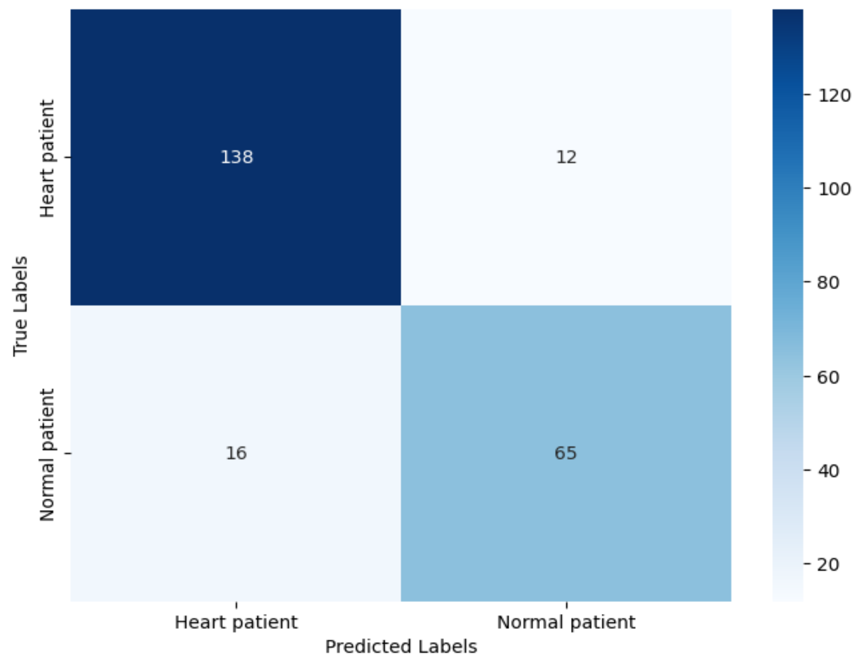

Figure 18: Confusion matrix for XGBoost Classifier

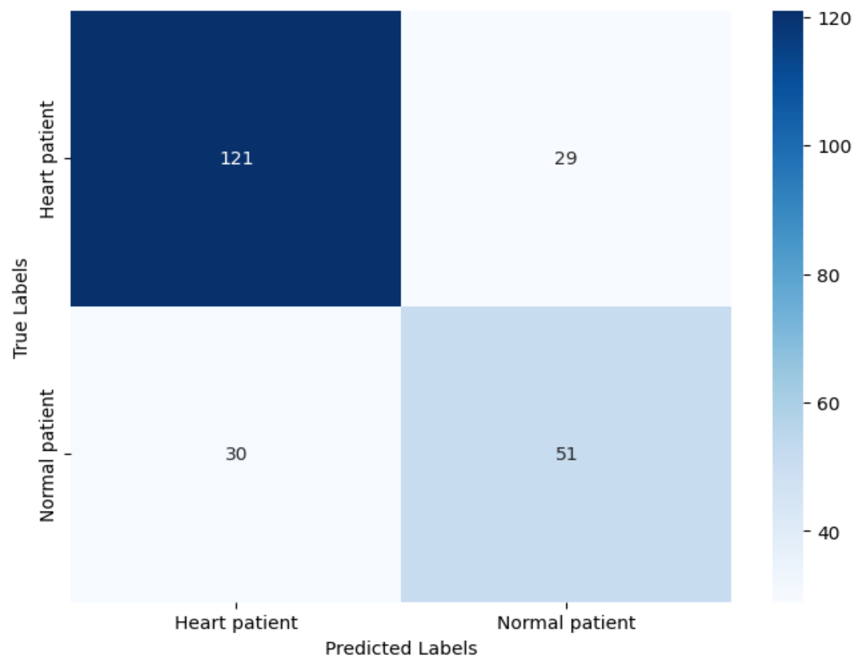

Figure 19: Confusion matrix for Gaussian Naive Bayes Classifier

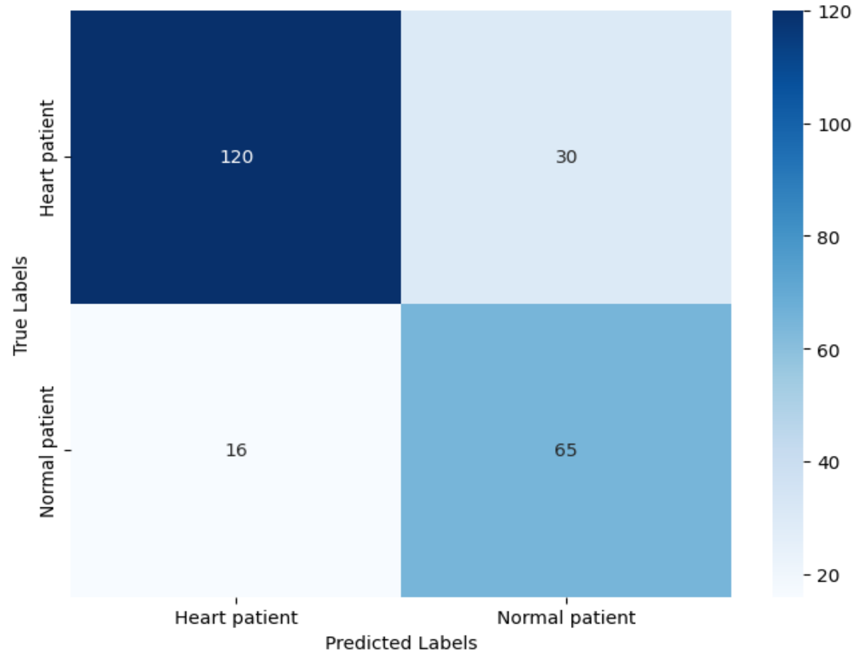

Figure 20: Confusion matrix for Stochastic Gradient Descent Classifier

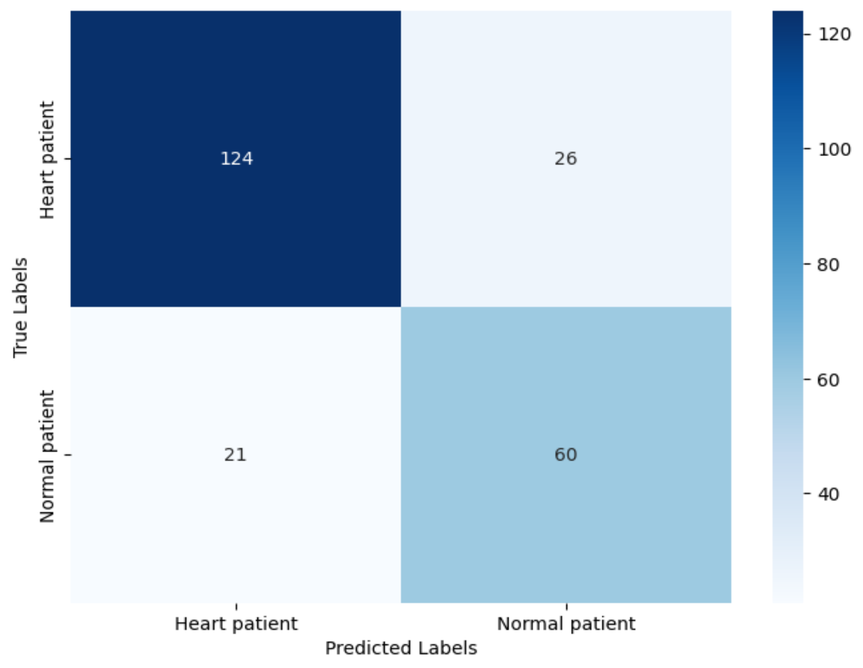

Figure 21: Confusion matrix for Ridge Classifier

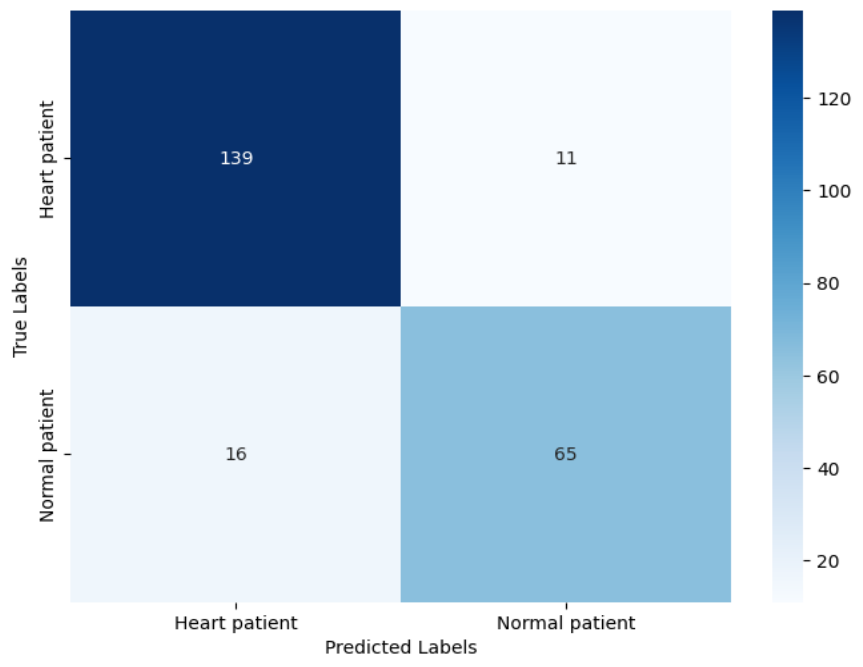

Figure 22: Confusion matrix for Bagging Classifier

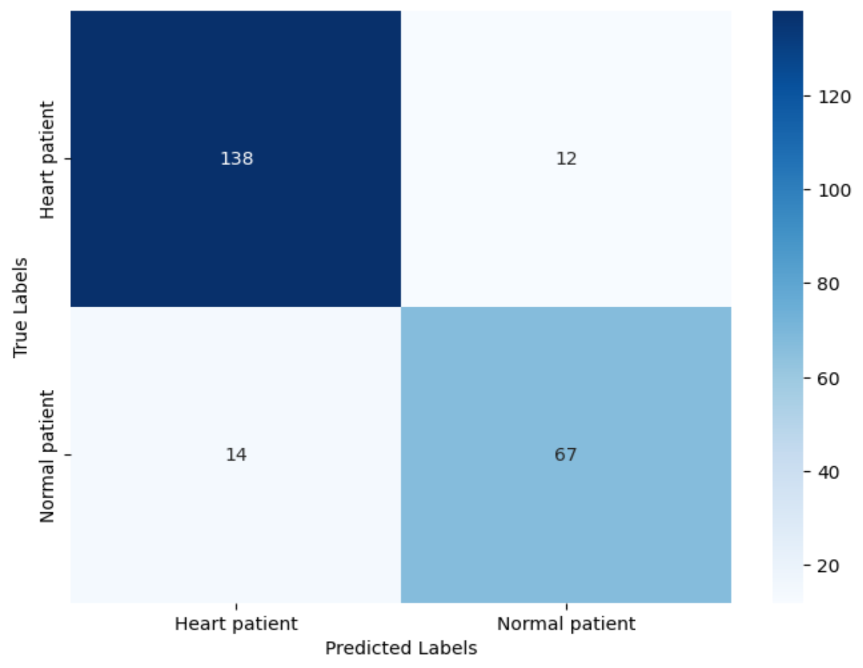

Figure 23: Confusion matrix for LightGBM

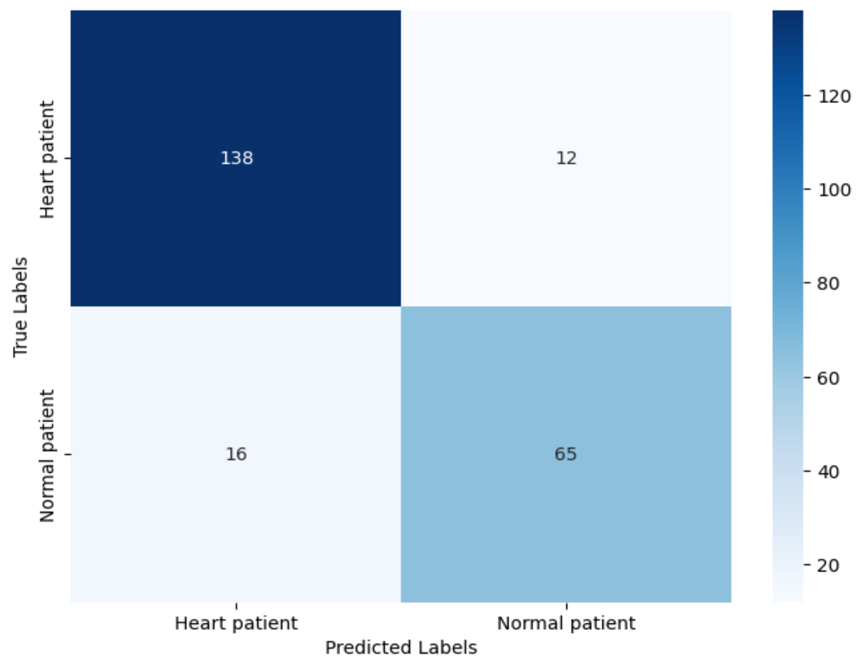

Figure 24: Confusion matrix for CatBoost Classifier

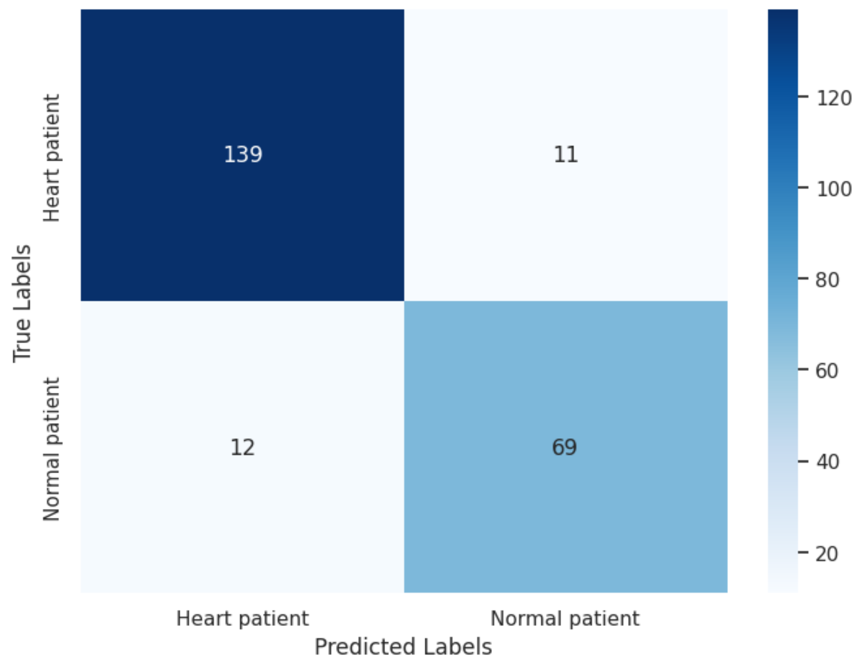

Figure 25: Confusion matrix for Voting Classifier (Soft)

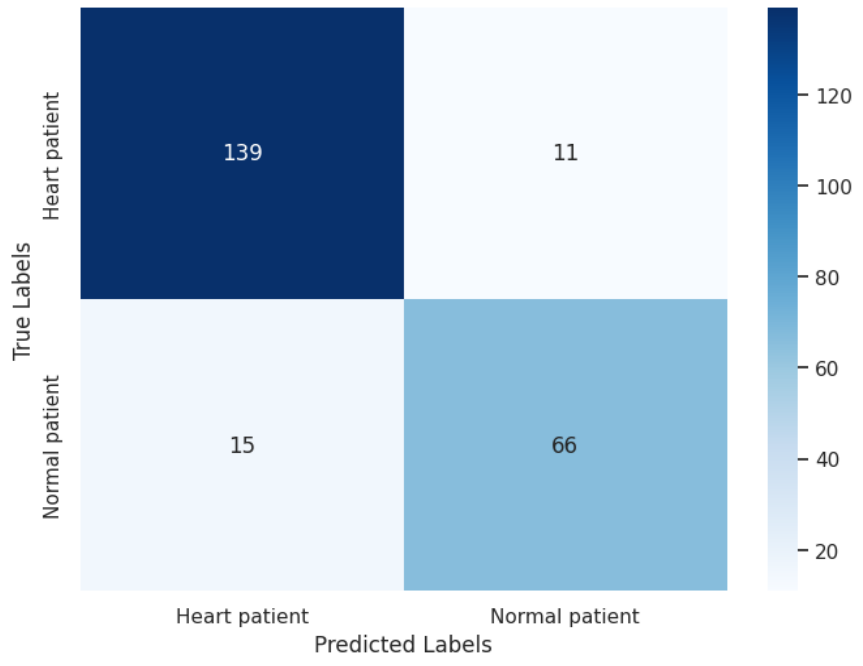

Figure 26: Confusion matrix for Voting Classifier (hard)

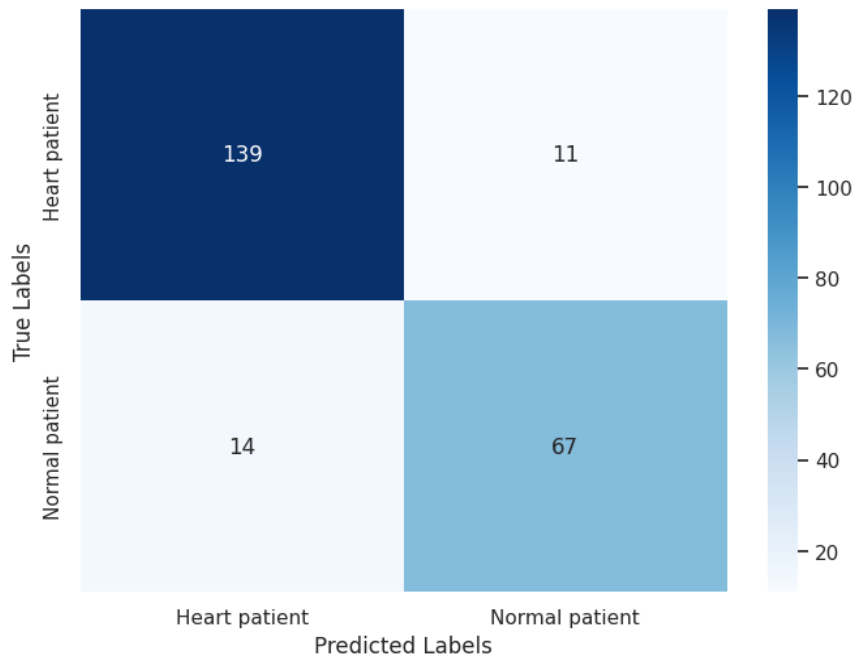

Figure 27: Confusion matrix for Stacking Classifier

True Positives (TP) represent the number of instances correctly classified as positive (correctly identified medical conditions). These are the cases that the model successfully detected as having the condition, which is crucial for early diagnosis and timely treatment.

False Negatives (FN) represent the number of instances incorrectly classified as negative (missed positive cases). These are the cases that the model mistakenly classified as healthy when they have the medical condition. False negatives are particularly concerning in a medical setting because they could lead to delayed diagnosis and treatment, potentially worsening the patient's health.

Here it is found that higher Ture Positive (138) and lower False Negative (12) in XGBoost Classifier, CatBoost Classifier, Random Forest Classifier, LightGBM, and Bagging Classifier models.

Also, we find higher False Negatives (33) in Stochastic Gradient Descent Classifier and Logistic Regression have also False Negate (28).

As for medical datasets, we focused on minimizing false negatives. This is because false negatives could lead to patients not receiving the necessary medical attention, which can have severe consequences. So, Models with higher True Positive rates (sensitivity) and lower False Negative rates are generally preferred in medical applications. Due to low False Negative XGBoost Classifier and CatBoost Classifier models are preferable in this scenario.

6.2. Evaluation Matrices:

Accuracy, precision, recall, and f1-score are considered for evaluation in this case:

Accuracy:

A frequent evaluation criterion for evaluating the effectiveness of a classification model is accuracy. It calculates the percentage of the model's total predictions that were accurate predictions. In other words, accuracy reveals the frequency of accurate model predictions. The formula is:

$$Accuarcy = \frac{Total\ Number\ of\ Predictions}{Number\ of\ Correct\ Predictions} = \frac{TP+TN}{TP+TN+FP+FN}$$

- Here Number of Correct Predictions: The sum of true positives and true negatives. These are the instances that were correctly classified by the model as either belonging to the positive class (e.g., class 1) or the negative class (e.g., class 0).
- Total Number of Predictions: The total number of instances (samples) for which the model made predictions.

Precision:

When evaluating binary classification models, precision is an essential evaluation statistic. It assesses the precision of the model's positive predictions, specifically the share of true positive predictions (positive instances that are accurately predicted) among all positive predictions (including true positives and false positives). The formula is:

$$Precision = \frac{True\ Positives\ (TP)}{True\ positives(TP) + false\ positives(FP)}$$

Recall:

The recall is a classification model evaluation metric that assesses a model's capacity to properly identify positive cases. Out of all real positive cases in the dataset, it indicates the percentage of true positive predictions (positive instances that were correctly classified). A high recall suggests the model is efficient at capturing positive examples and has fewer false negatives, making it appropriate for circumstances when failing to detect positive instances is crucial (such as establishing a medical diagnosis). The high recall could, however, lead to an increase in false positives. It is possible to evaluate recall combined with other metrics, like precision and F1-score, to determine how well the model performs and whether it is appropriate for a given classification task. The formula is denoted below:

$$Recall\ = \frac{True\ Positives\ (TP)}{True\ positives(TP) + false\ negative(FN)}$$

F1-score:

The F1-score is a classification model evaluation metric that combines precision and recall into one number. It assesses the model's capacity for both accurate positive instance identification and the avoidance of false positives. The F1-score is a number between 0 and 1, with 1 denoting flawless performance. When working with datasets that are unbalanced and in which one class may predominate over another, it is especially helpful. For tasks where both measures are equally significant for evaluation, a high F1 score implies a well-performing model with a good trade-off between recall and precision. The formula stated below:

$$\text{f1-score}=\frac{2*precision*recall}{precision+recall}$$

The values of evaluation matrices for all models are shown below:

Model	Accuracy (%)	Precision (%)	Recall (%)	f1-score (%)
LogisticRegression	79.22	68.54	75.31	71.76
SVC	85.28	77.40	83.95	80.00
KNeighborsClassifier	85.28	78.31	80.25	79.27
DecisionTreeClassifier	83.12	76.25	75.31	75.78
RandomForestClassifier	89.61	85.19	85.19	85.19
XGBClassifier	87.88	84.42	80.25	82.28
GaussianNB	74.46	63.75	62.96	63.35
SGDClassifier	79.65	68.48	77.78	72.83
RidgeClassifier	79.65	69.77	74.07	71.86
BaggingClassifier	88.31	86.49	79.01	82.58
LGBMClassifier	88.74	84.81	82.72	83.75
CatBoostClassifier	87.88	84.42	80.25	82.28
Voting Classifier (Soft)	90.04	86.25	85.19	85.71
Voting Classifier (Hard)	89.18	85.90	82.72	84.28
Stacking Classifier	89.61	86.08	83.95	85.00

6.3. Learning Curve:

To understand the model convergence technique in cross-validation, a learning curve is applied. The learning curve of cross-validation is a useful tool in machine learning to evaluate how a model's performance improves as the size of the training data increases. It provides insights into the relationship between the model's training set size and its generalization performance. The learning curve is typically created by plotting the model's performance (e.g., accuracy, error rate, F1 score) on both the training set and the validation set as a function of the number of training samples. The x-axis represents the number of training samples, and the y-axis represents the performance metric.

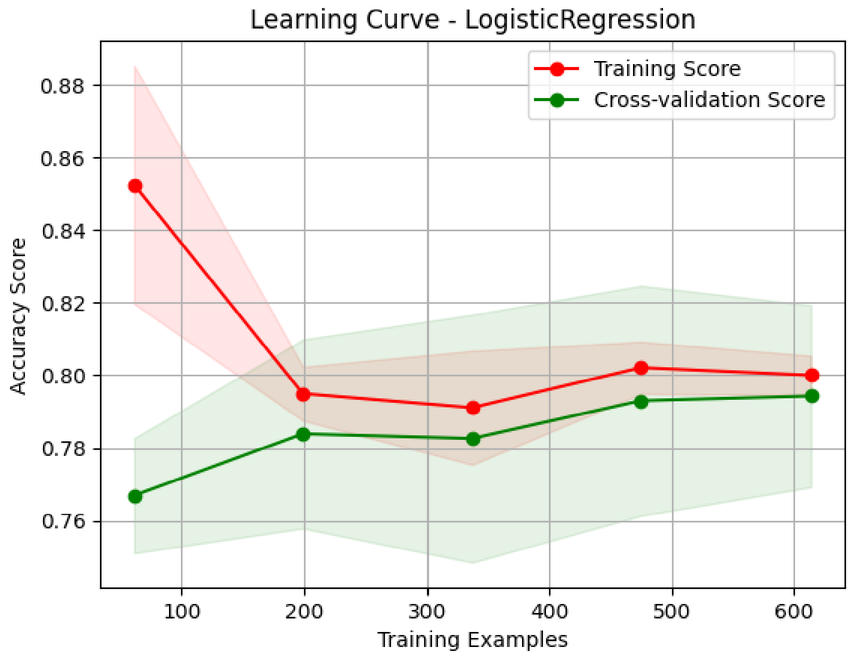

Figure 28: The Learning curve of cross-validation for Logistic Regression

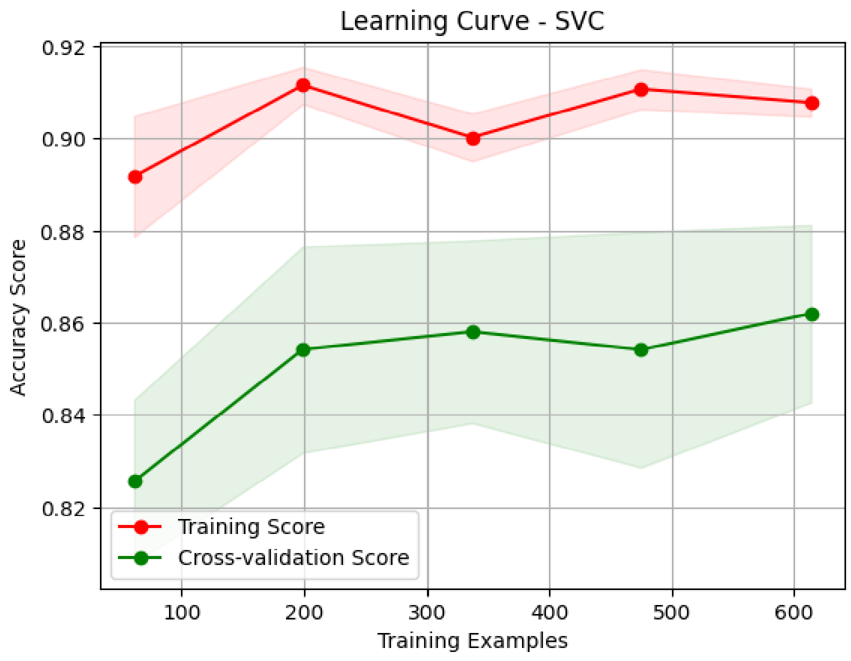

Figure 29: The learning curve of cross-validation for Support Vector Machine

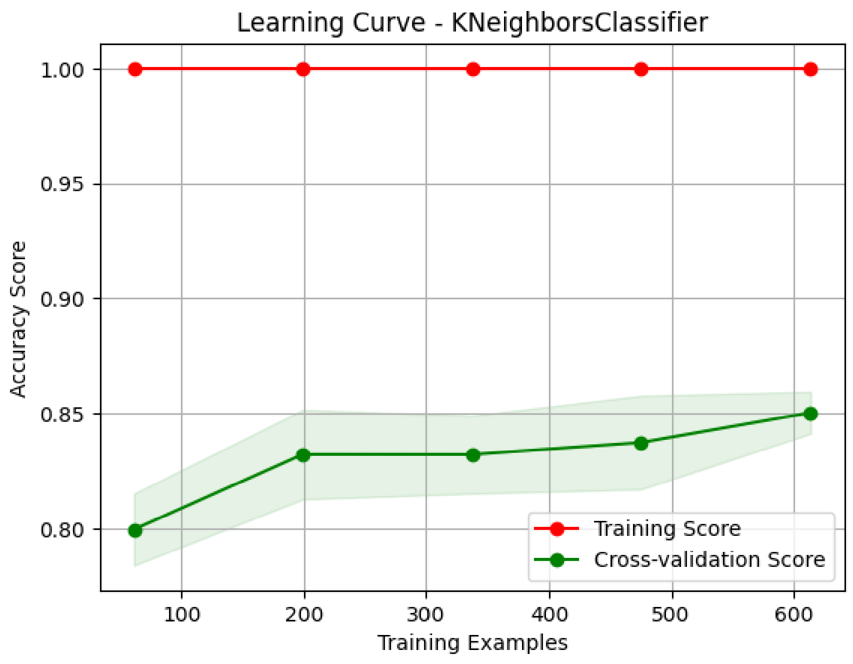

Figure 30: The Learning curve of cross-validation for K-Nearest Neighbors

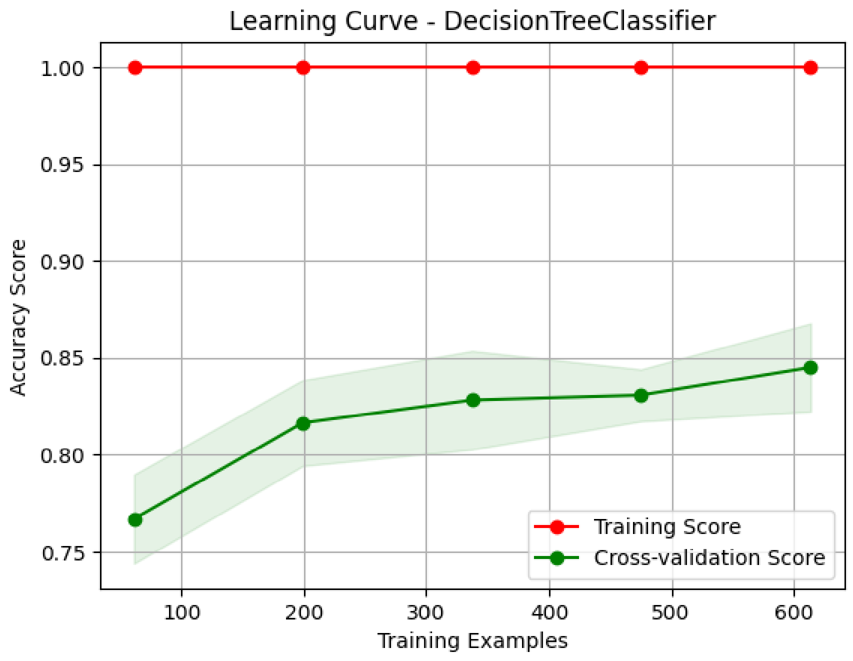

Figure 31: The Learning curve of cross-validation for Decision Tree Classifier

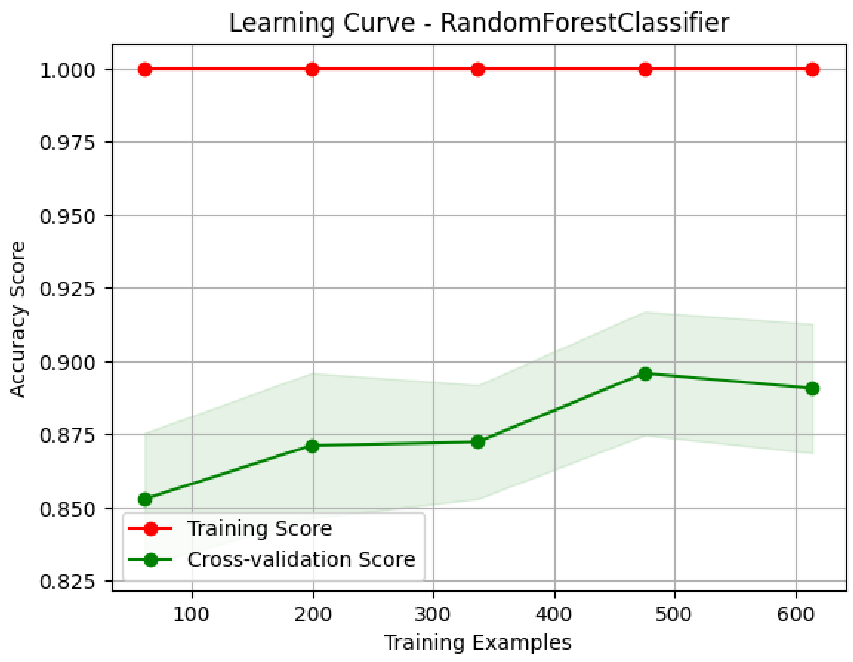

Figure 32: The learning curve of cross-validation for the Random Forest Classifier

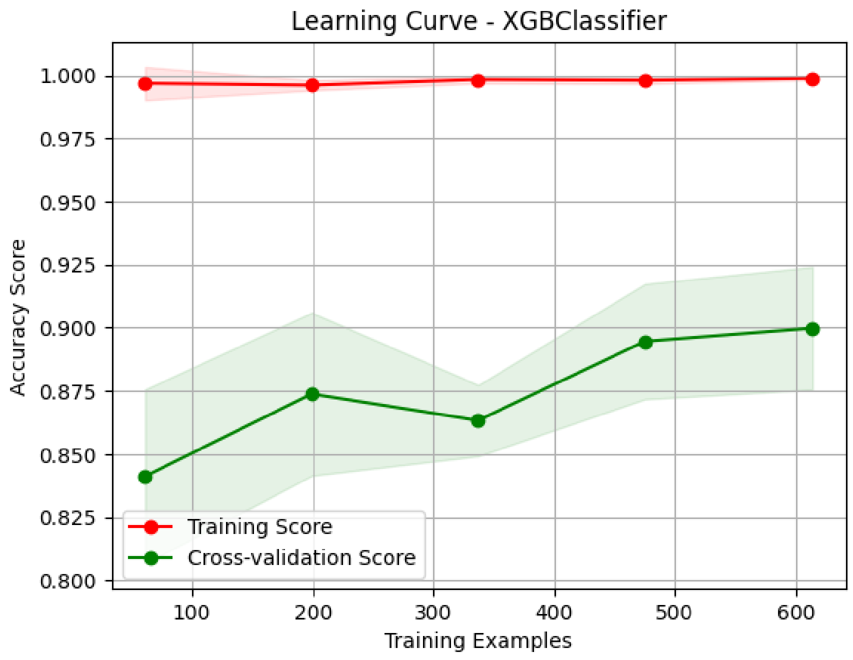

Figure 33: The Learning curve of cross-validation for the XGBoost Classifier

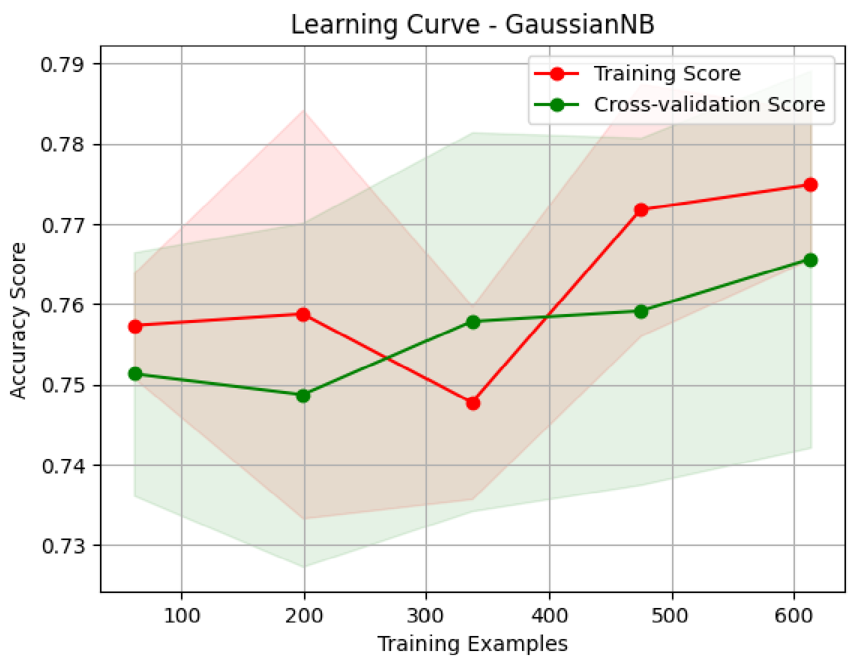

Figure 34: The Learning curve of cross-validation for the Gaussian Naive Bayes Classifier

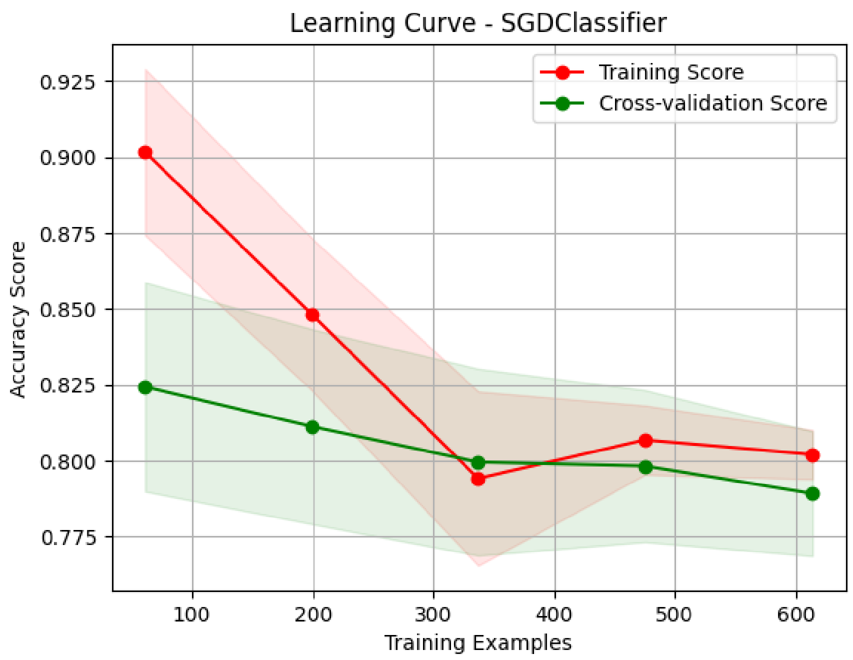

Figure 35: The Learning curve of cross-validation for the Stochastic Gradient Descent Classifier

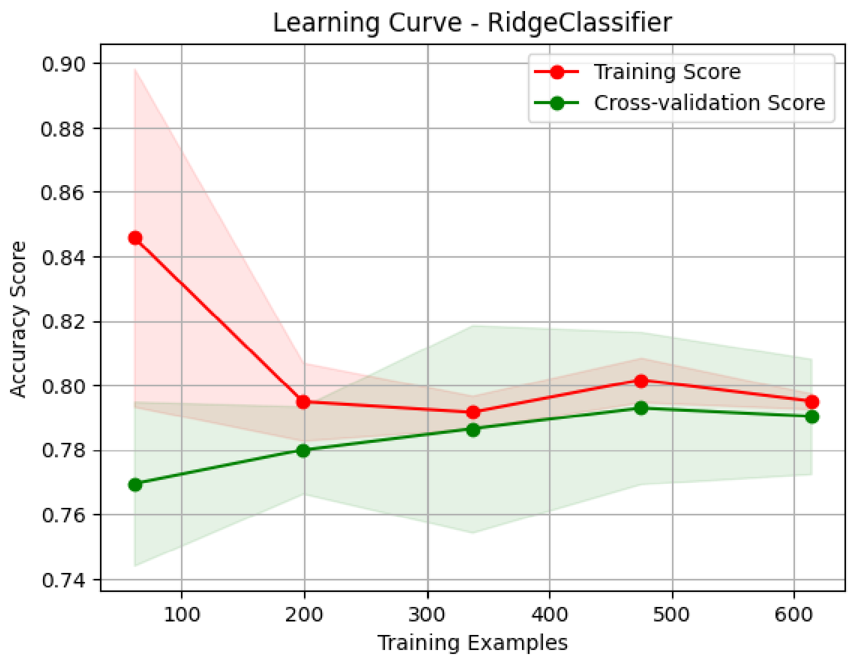

Figure 36: The Learning curve of cross-validation for the Ridge Classifier

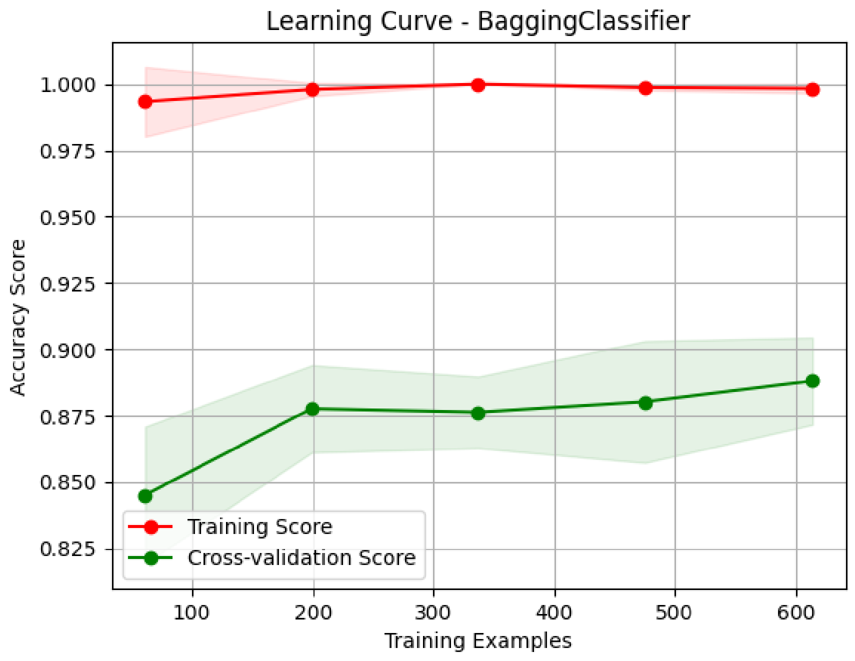

Figure 37: The learning curve of cross-validation for the Bagging Classifier

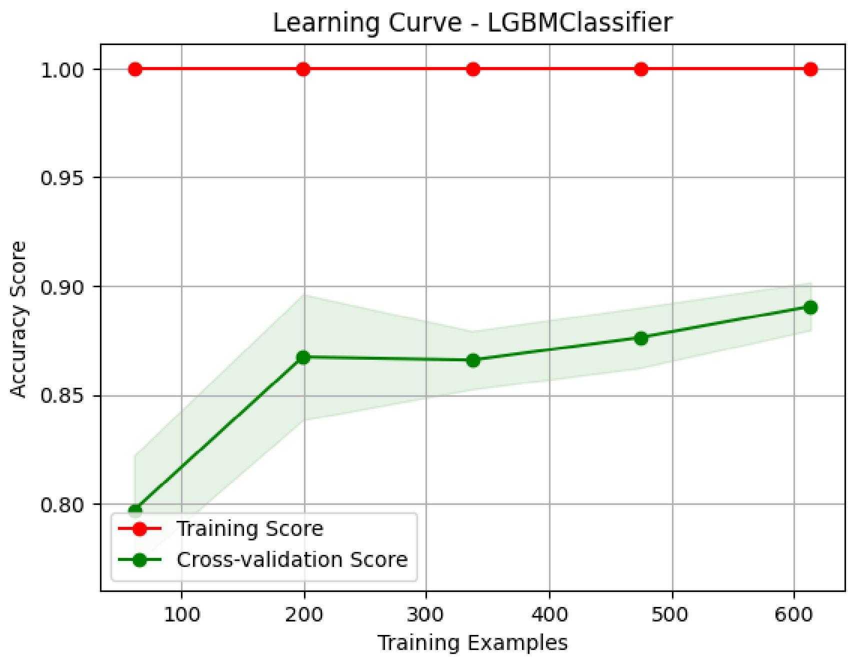

Figure 38: The Learning curve of cross-validation for LightGBM

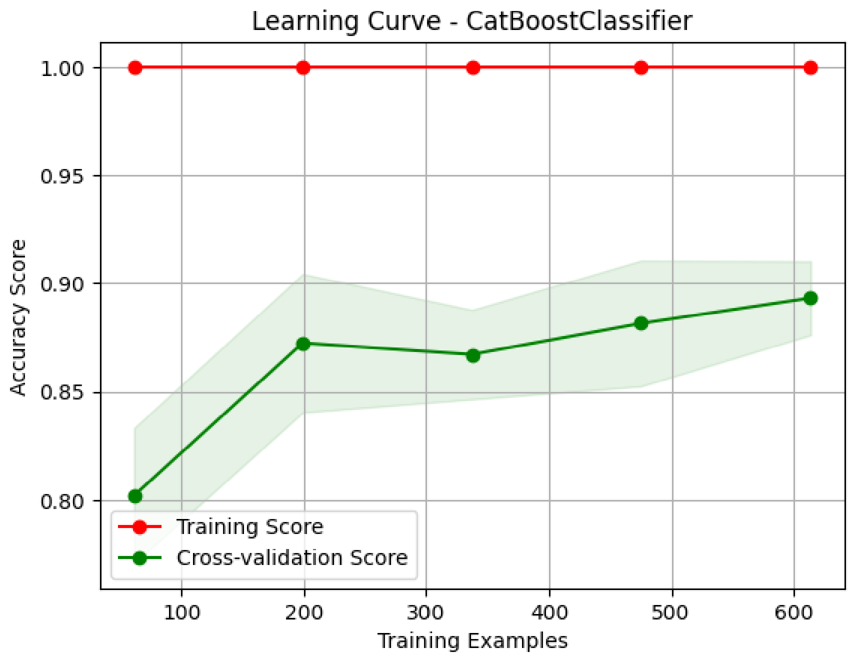

Figure 39: The learning curve of cross-validation for the CatBoost Classifier

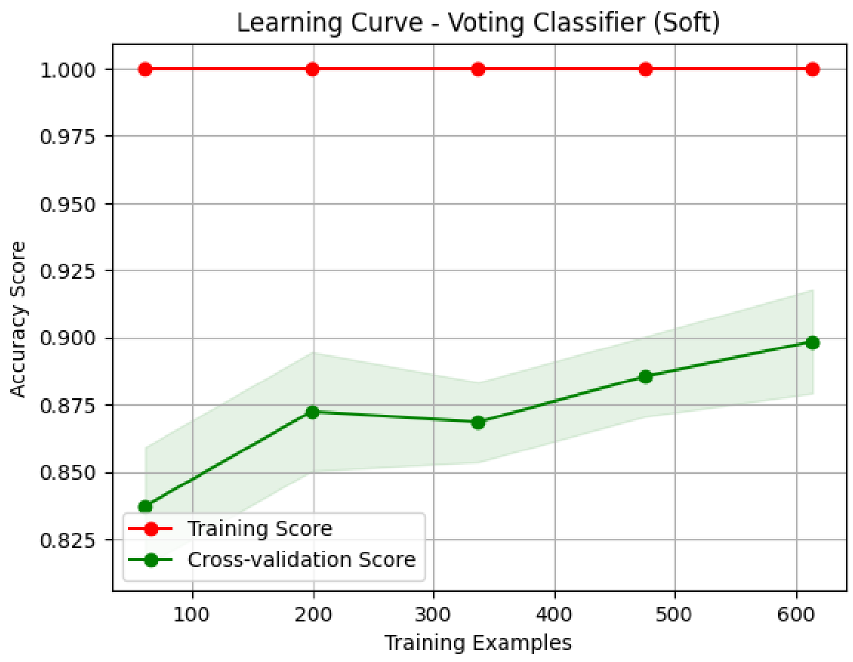

Figure 40: The Learning curve of cross-validation for Voting Classifier (Soft)

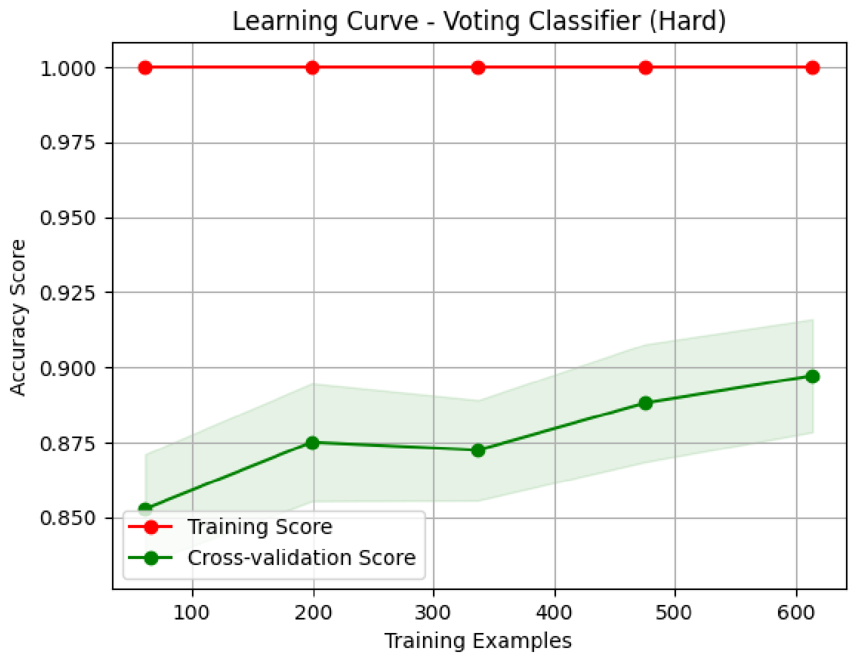

Figure 41: The learning curve of cross-validation for Voting Classifier (Hard)

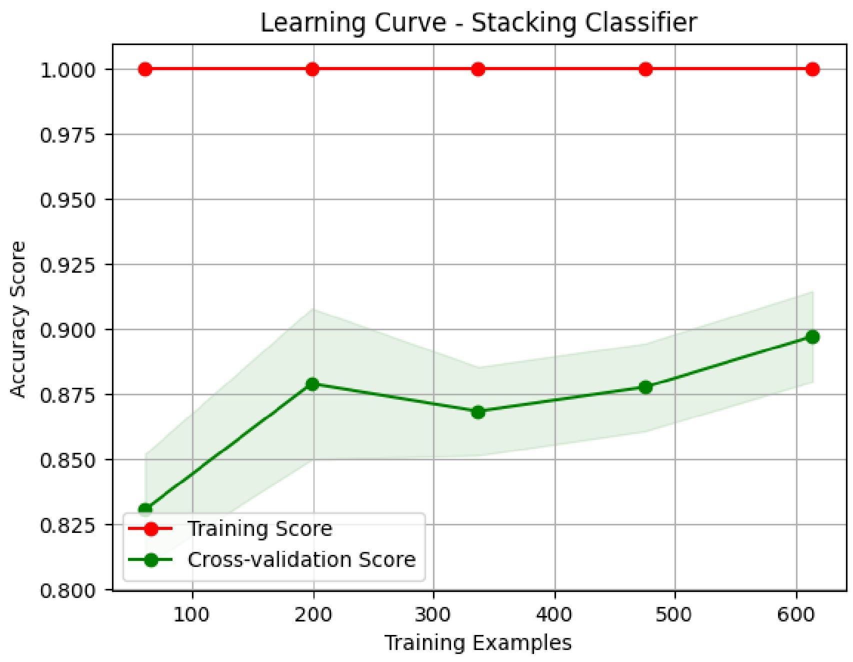

Figure 42: The Learning curve of cross-validation for Stacking Classifier

If the training score is much greater than the validation score then the model probably requires more training examples in order to generalize more effectively. Here it can be seen that CatBoost Classifier, XGBoost Classifier, Decision Tree Classifier, and K-Nearest Neighbors models the training and test scores have not yet converged, so potentially this model would benefit from more training data. This model will do better for a large number of training sets.

If the training and cross-validation scores converge together as more data is added then the model will probably not benefit from more data. Logistic Regression, Gaussian Naive Bayes Classifier, and Stochastic Gradient Descent Classifier models show too much variability. With random data, these curves are highly variable, but serve to point out some clustering-specific items.

6.4. ROC Curve:

To understand how the model can separate opposite class and is thresholding work perfectly, the ROC curve is used. ROC curve (Receiver Operating Characteristic curve) is a graphical representation that illustrates the performance of a binary classification model at various classification thresholds. It is a powerful tool for evaluating the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) as the model's discrimination threshold is adjusted. The AUC is a performance metric that ranges from 0 to 1, where higher values indicate better model performance. The AUC values represent the performance of each model in distinguishing between positive and negative instances. A higher AUC generally indicates better model performance and stronger discriminative power.

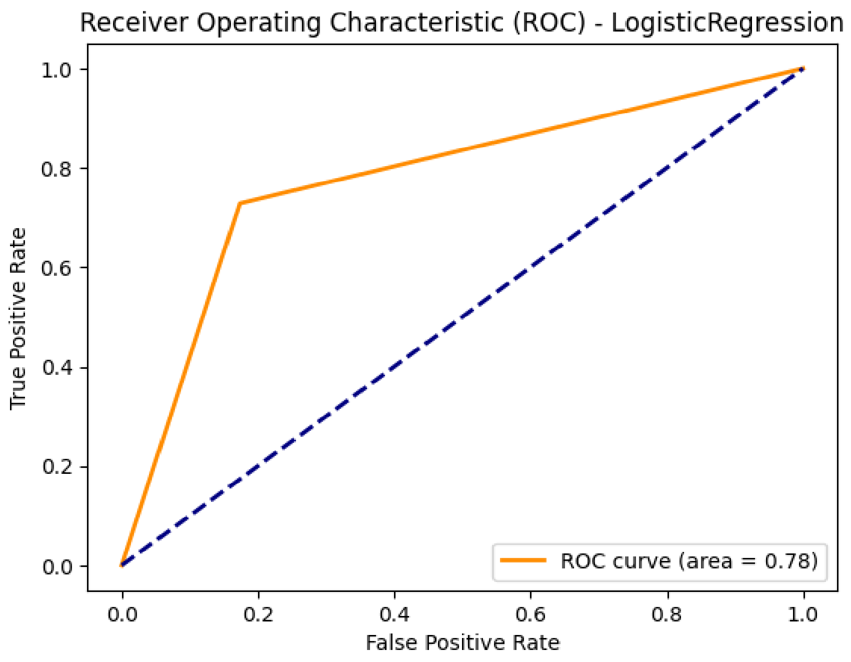

Figure 43: ROC Curve for Logistic Regression

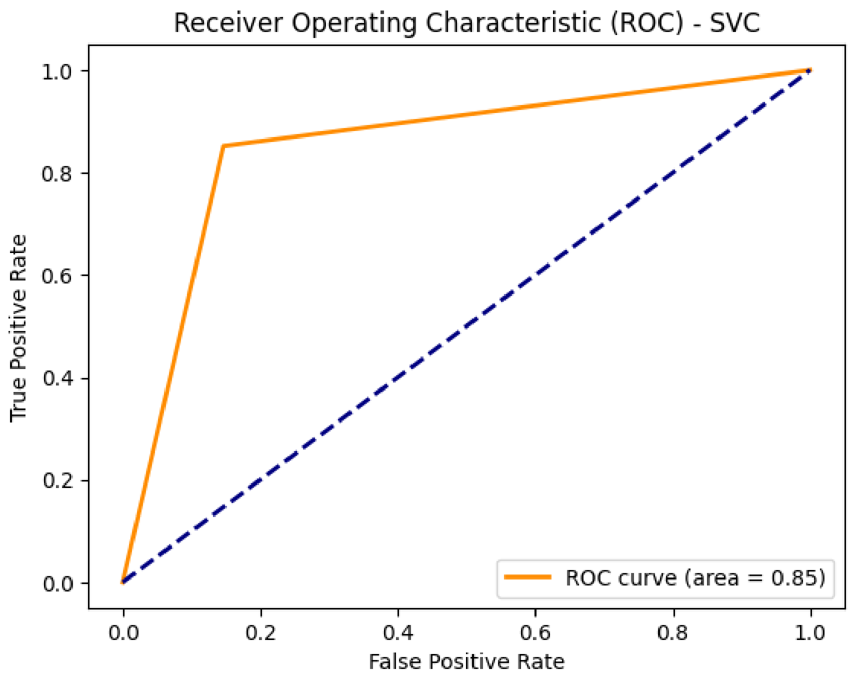

Figure 44: ROC Curve for Support Vector Machine

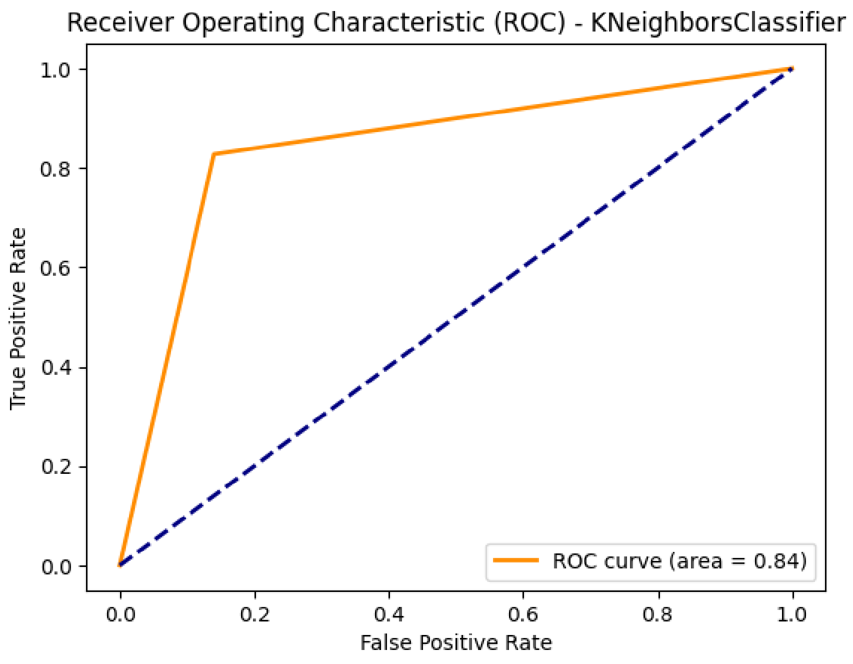

Figure 45: ROC Curve for K-Nearest Neighbors

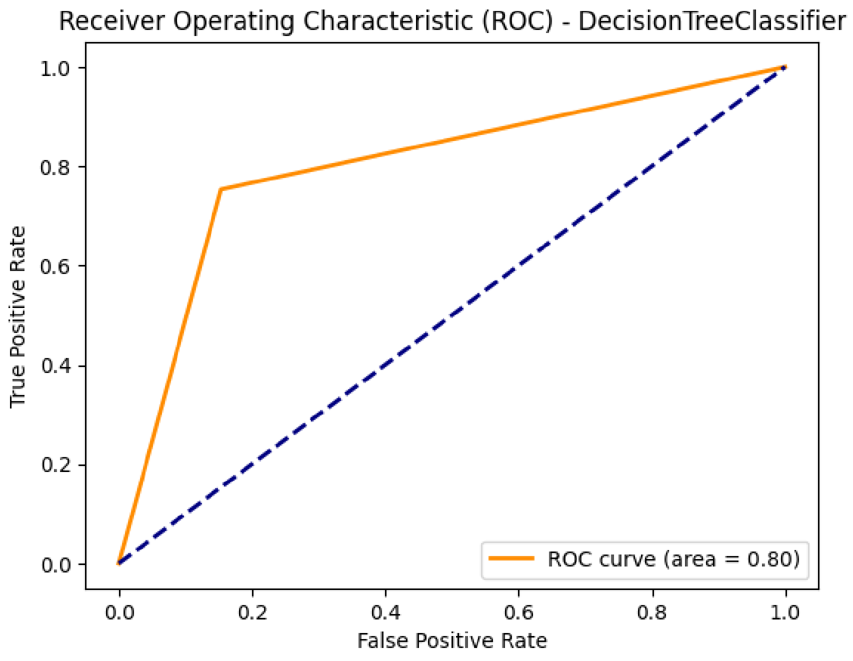

Figure 46: ROC Curve for Decision Tree Classifier

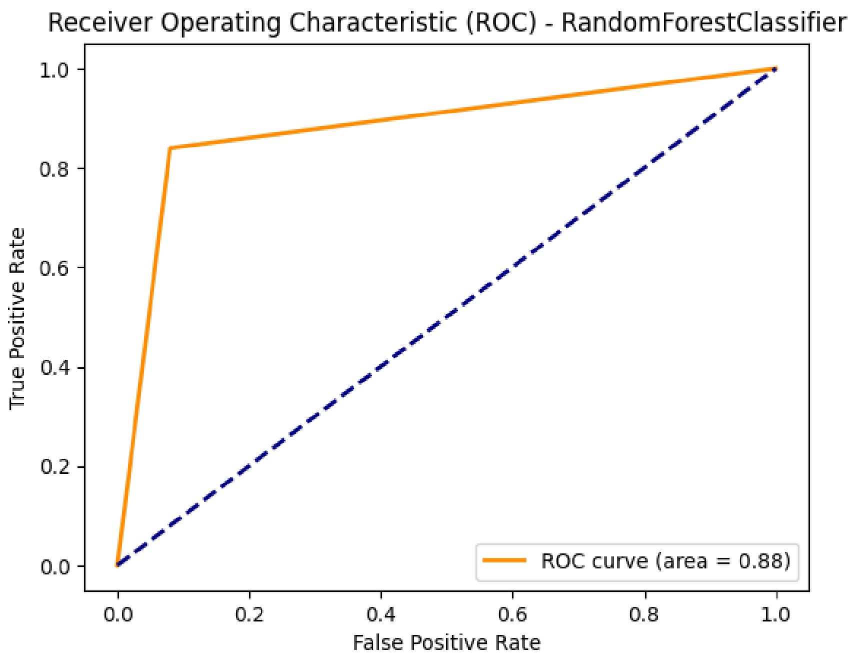

Figure 47: ROC Curve for Random Forest Classifier

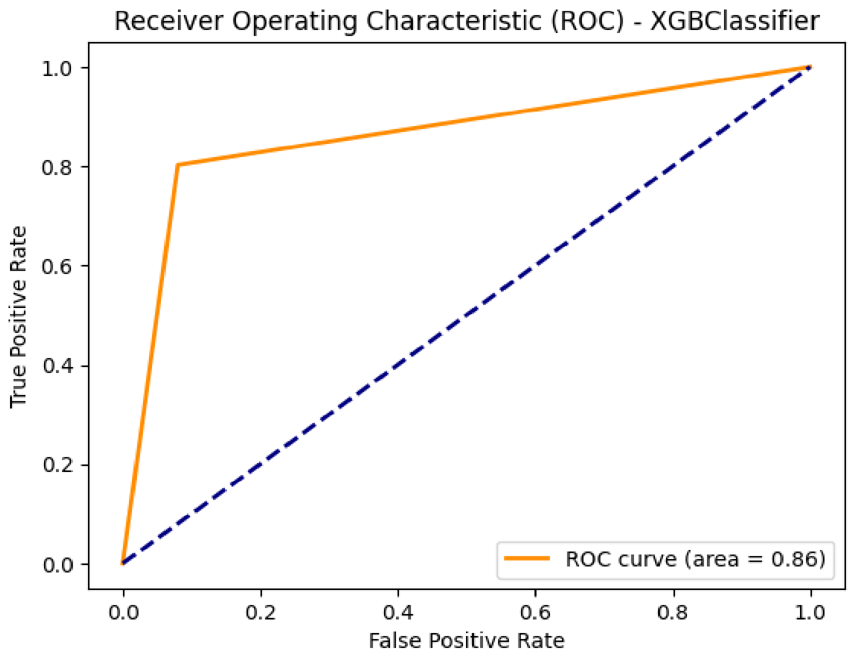

Figure 48: ROC Curve for XGBoost Classifier

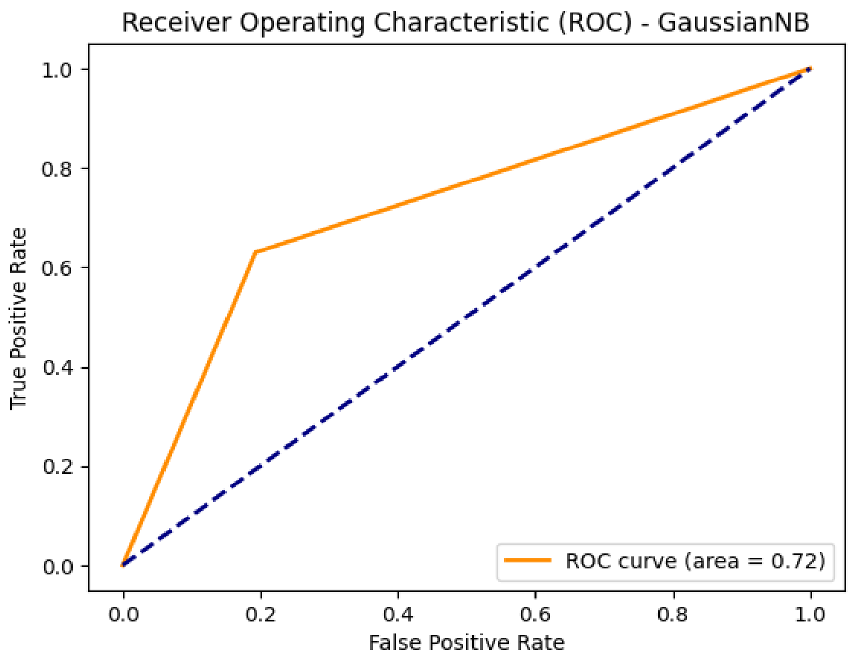

Figure 49: ROC Curve for Gaussian Naive Bayes Classifier

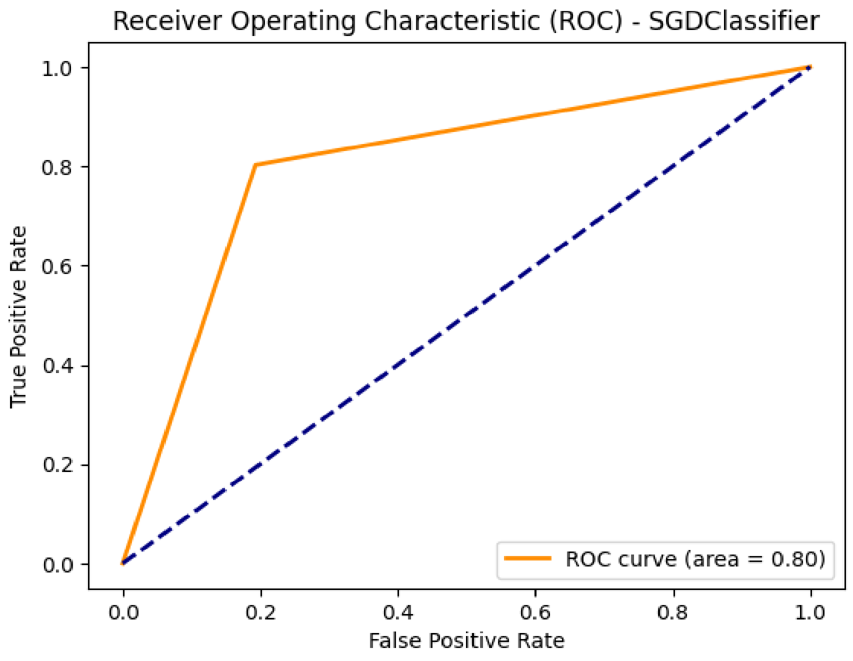

Figure 50: ROC Curve for Stochastic Gradient Descent Classifier

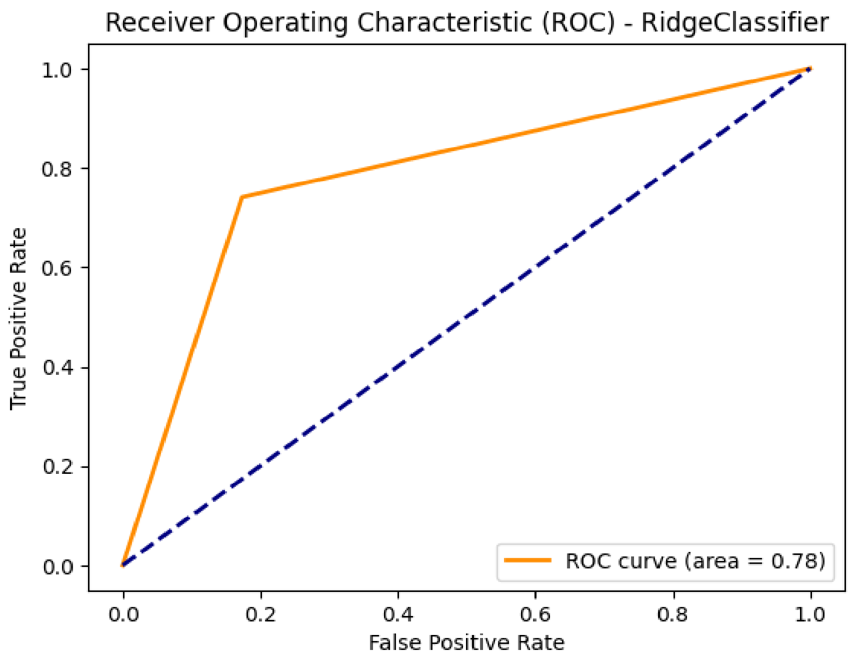

Figure 51: ROC Curve for Ridge Classifier

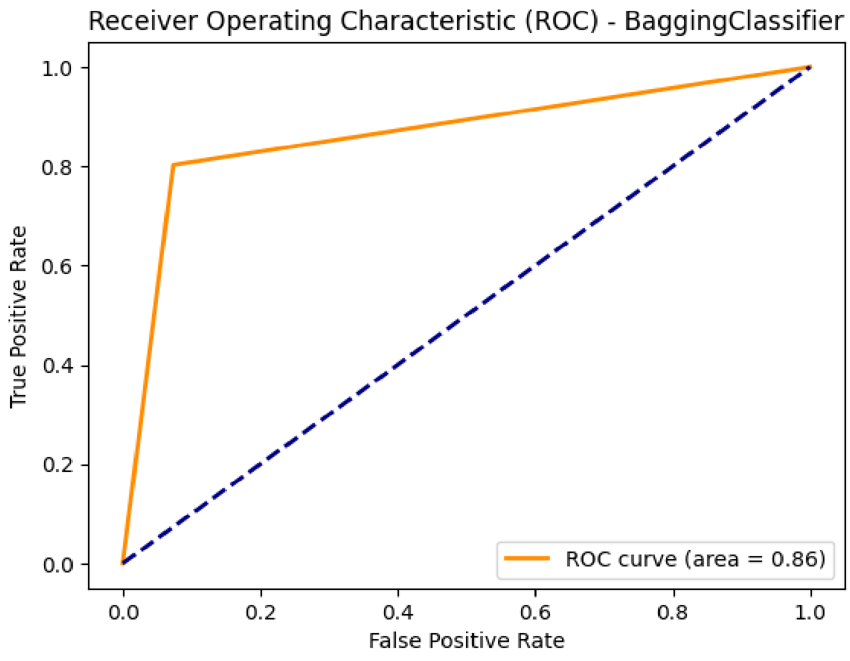

Figure 52: ROC Curve for Bagging Classifier

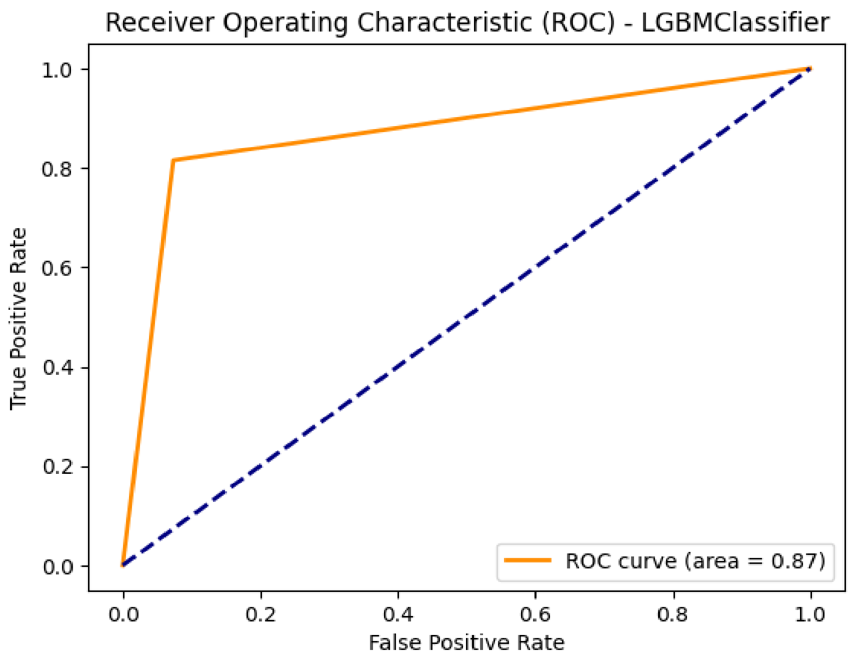

Figure 53: ROC Curve for LightGBM

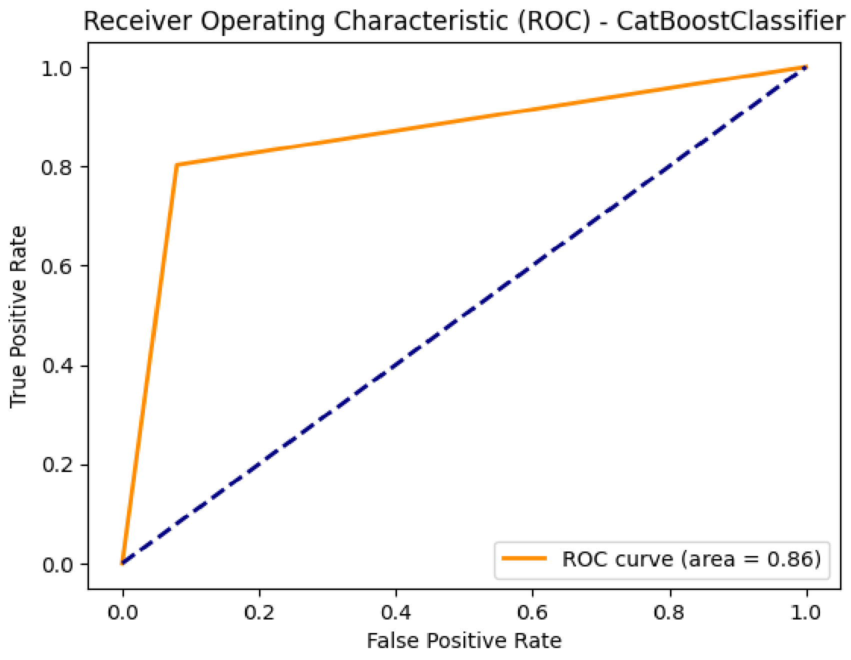

Figure 54: ROC Curve for CatBoost classifier

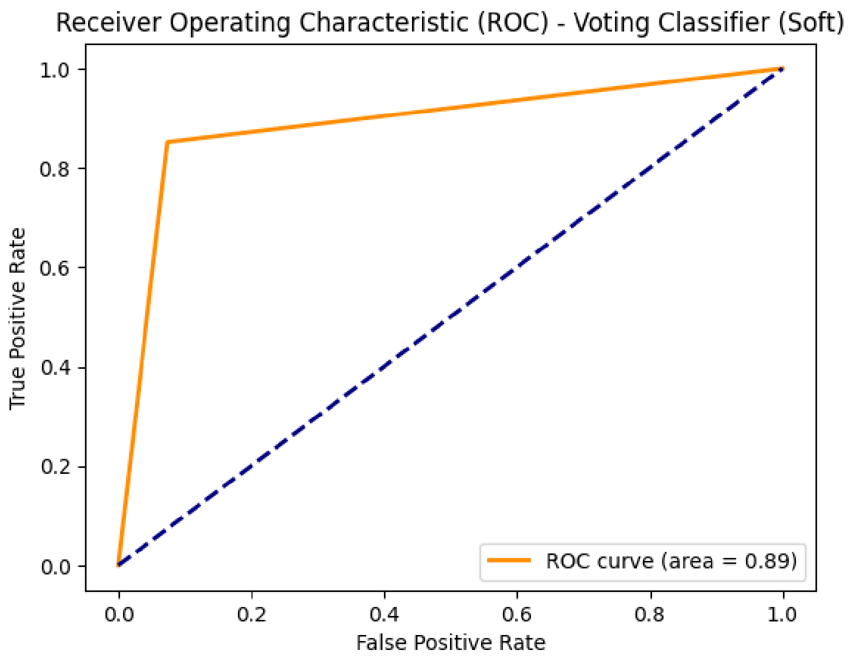

Figure 55: ROC Curve for Voting Classifier (Soft)

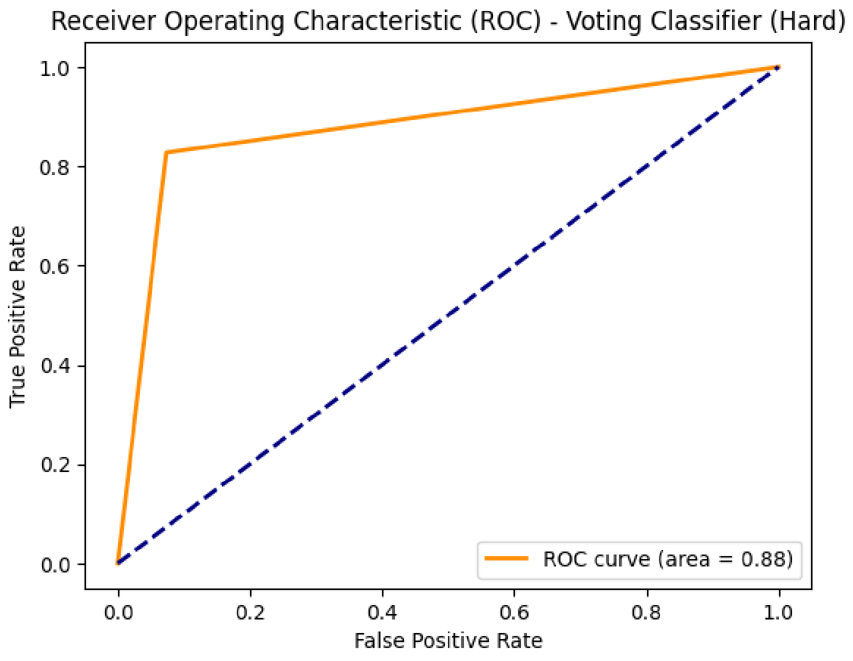

Figure 56: ROC Curve for Voting Classifier (Hard)

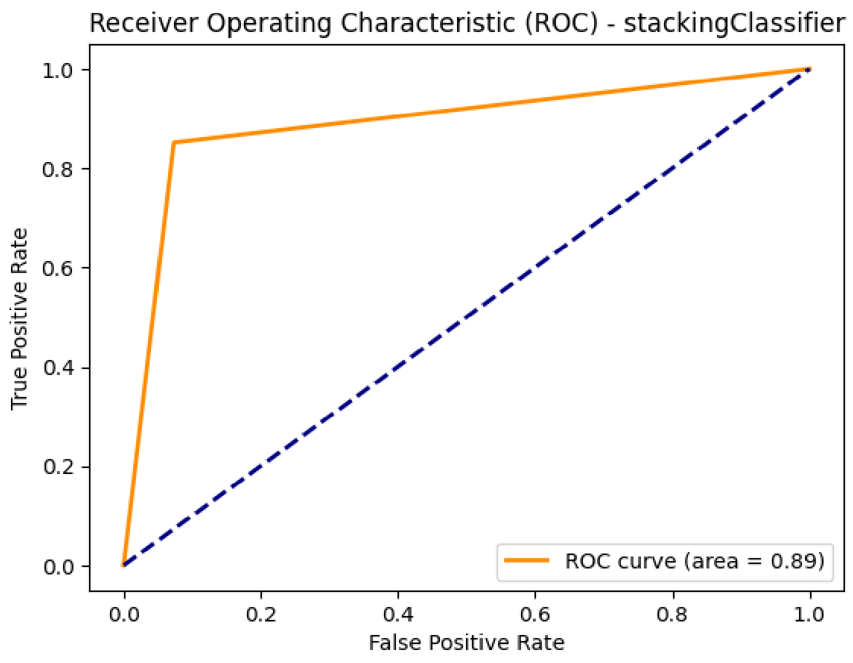

Figure 56: ROC Curve for Voting Classifier (Hard)

Model Name	**AUC (Area Under the ROC Curve) %**
Stacking Classifier	89
Voting Classifier (Soft)	89
Random Forest Classifier	89
Voting Classifier (Hard)	88
Bagging Classifier	88
LightGBM	87
XGBoost Classifier	86
CatBoost Classifier	86
Support Vector Machine	85
K-Nearest Neighbors	84
Stochastic Gradient Descent Classifier	82
Decision Tree Classifier	80
Ridge Classifier	78
Logistic Regression	78
Gaussian Naive Bayes Classifier	72

Here Random Forest Classifier has an AUC of 89%, the higher value of AUC suggesting it has strong discriminatory power and can effectively differentiate between positive and negative instances. This model gives better performance than others.

The Bagging Classifier obtained an AUC of 88%. This high AUC score suggests that the model also possesses good discriminative power and performs well in classification tasks. The Bagging Classifier closely follows the Random Forest Classifier in performance.

LightGBM demonstrated an AUC of almost 87%. The model's high AUC indicates its effective discriminatory power and commendable classification performance.

Both the XGBoost Classifier and CatBoost Classifier achieved an AUC of 86%. These models display strong discriminatory power and competitive performance in classification tasks.

The Support Vector Machine obtained an AUC of 85%, indicating competent classification capabilities. The K-Nearest Neighbors algorithm achieved an AUC of 84%, demonstrating good performance in distinguishing between classes. The Stochastic Gradient Descent Classifier displayed an AUC of 82%, showcasing decent performance in classification tasks. The Decision Tree Classifier obtained an AUC of 80%, indicating relatively good discriminatory power.

Both the Ridge Classifier and Logistic Regression models shared an AUC of 78%. These models showed moderately lower discriminative power compared to the top-performing models.

The Gaussian Naive Bayes Classifier displayed the lowest AUC score of 72%. This score suggests that the model has weaker discriminatory power and lower overall performance compared to the other models in the list.

6.5. Feature Importance:

After all, the most influential features and their impact on the model's performance are identified through feature importance. This knowledge facilitates informed decisions in feature selection, model enhancement, and gaining potential data insights. Overall, feature importance analysis is a fundamental step in model interpretation, enabling us to comprehend how the model utilizes input features to make accurate predictions and better understand the underlying relationships in the data. Feature importance in machine learning refers to a measure of how much each feature (input variable) in a model contributes to the model's predictions or outcomes. It is a critical concept for understanding the relationships between features and the target variable and gaining insights into the underlying patterns in the data.

After assessing the feature importance, it is found that the "INSULIN" feature stands out as the most important feature among the features. Its importance indicates that it plays a significant role in distinguishing or classifying the target variable in our model. The features "SKINTHICKNESS," "NEW_GLUCOSE*INSULIN," "GLUCOSE," and "AGE" are also observed to have notable importance and contribute significantly to the model's predictions.

On the other hand, certain features like "AGE_CAT_Old," "PREGNANT_CAT_Never," and "GLUCOSE_CAT_IMPAIRED_Glucose" are considered less important because their contribution to the model's performance is relatively lower.

The understanding of feature importance enables us to make informed decisions regarding feature selection, model improvement, and potential data insights.

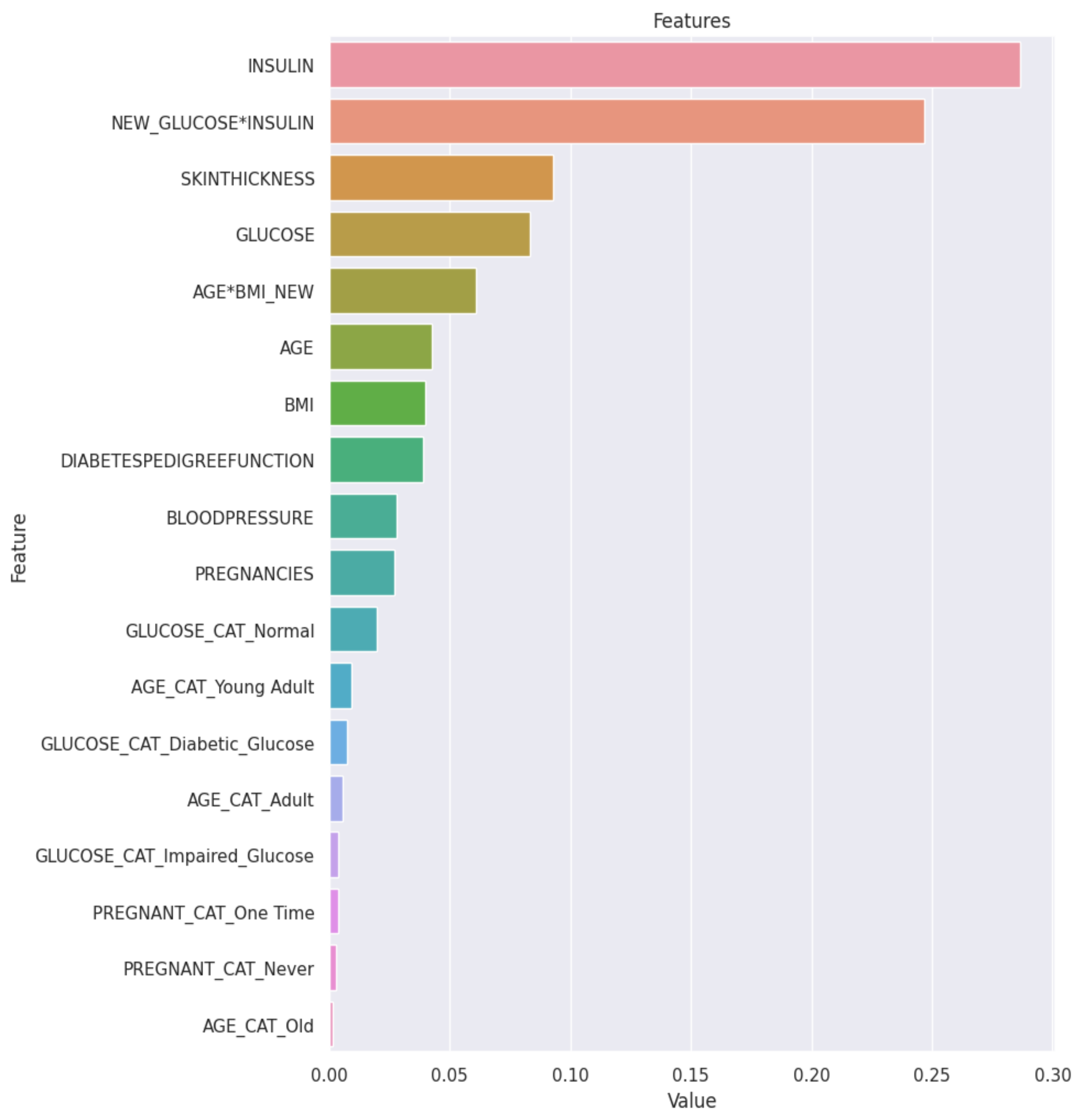

Figure 58: Feature Importance of Random Forest Model

6.6. Model Interpretation:

Model interpretation in machine learning refers to the process of understanding and explaining how a trained machine learning model makes predictions. It involves extracting insights and knowledge from the model to gain a deeper understanding of its decision-making process and how it relates to the input features.

Indeed, model interpretation techniques are essential for understanding how machine learning models make decisions and gaining insights into their behavior. SHAP (SHapley Additive exPlanations) is a powerful tool for model interpretation, providing valuable information about feature importance and contributions to predictions.

In this study, "INSULIN" and "NEW_GLUCOSE*INSULIN" are identified as the most crucial features of the model. These features play a significant role in accurately classifying the target variable and have a substantial impact on the model's predictions.

"SKINTHICKNESS," "NEW_GLUCOSEINSULIN," "GLUCOSE," "AGEBMI_NEW," and "AGE" are also considered important features. They contribute significantly to separating the target classes or making accurate regression predictions.

Features like "BMI," "AGE_CAT_Old," "PREGNANT_CAT_Never," and "GLUCOSE_CAT_IMPAIRED_Glucose" play a role in model separation but have a relatively lesser impact compared to the most important features. Their influence on the model's predictions is relatively minor.

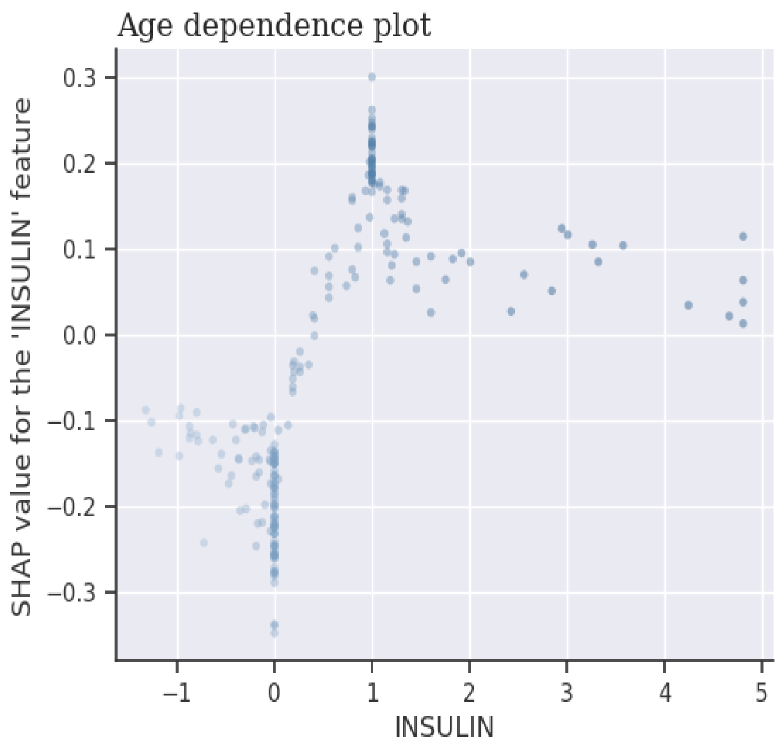

Figure 59: SHAP value for Insulin feature

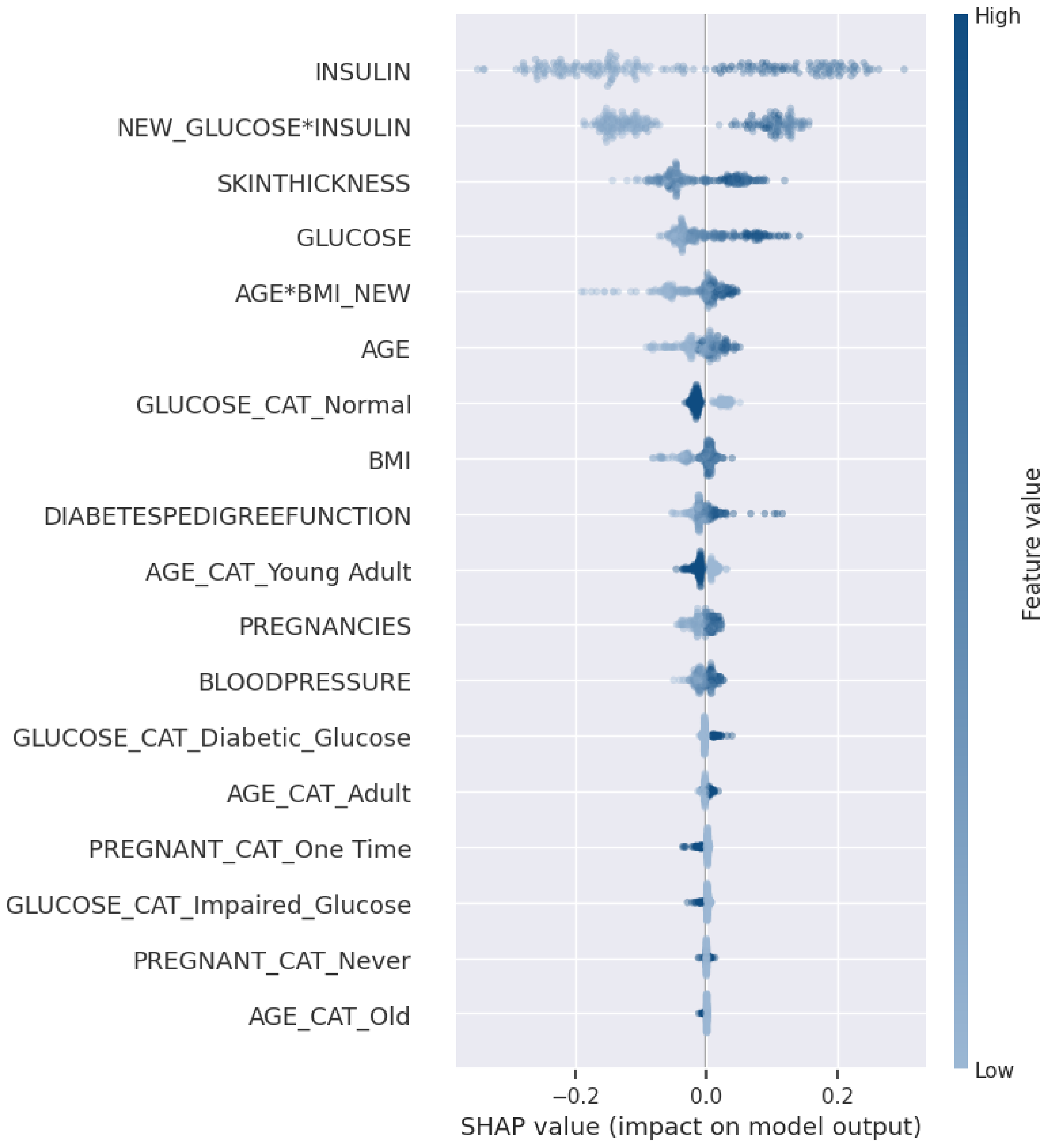

Figure 60: SHAP value of the features

7. Conclusion:

The prevalence of diabetes is increasing worldwide, highlighting the need for accurate and efficient diagnostic systems. We can’t prevent it, but still, there is a hope to get enough medical support if early detection will had introduced. In this study, multiple preprocessing techniques are applied as real-life data is not so well organized to feed into a mathematical machine-learning model. The missing data problem is handled also through the median value, which has proven to be the most effective algorithm for handling incomplete data.

To predict diabetes, the power of various classification algorithms, including XGBoost, Adaboost, gradient boosting, extra trees, light gradient boosting (LightGBM), SGDC, Nu SVM, and the stacking algorithm are harnessed. The stacking classifier yielded an impressive accuracy score of 90.04%, making it a robust tool for automatic diagnosis and supporting healthcare professionals in assessing their patients' medical conditions. The voting classifier also shows good enough performance with 89% accuracy so that it can also be chosen as a diagnosis tool for diabetic prediction. But another assumption finds out is that it has slightly improved the accuracy of stacking or voting classifier than a single or individual model but computational cost increased through considering 5 models.

Looking forward, this research can be expanded to develop an efficient prediction system for improving medical treatments and reducing healthcare expenses. Future studies may explore the correlation between diabetes and other clinical conditions, such as heart disease, as well as investigate potential associations between cardiac health and meteorological conditions or air quality using AI-based techniques.

Moreover, the prediction system's integration with other healthcare systems, such as context-aware security access control, holds the promise of bolstering the security foundations of the healthcare infrastructure. Overall, this work presents significant prospects for enhancing the functionality of diabetes prediction systems and contributing to improved patient care and medical decision-making.

8. REFERENCES:

Ashiquzzaman, A., Tushar, A.K., Islam, M.R., Shon, D., Im, K., Park, J.H., Lim, D.S. and Kim, J., 2018. Reduction of overfitting in diabetes prediction using deep learning neural network. In *IT Convergence and Security 2017: Volume 1* (pp. 35-43). Springer Singapore.

Babu, S.B., Suneetha, A., Babu, G.C., Kumar, Y.J.N. and Karuna, G., 2018. Medical disease prediction using grey wolf optimization and auto encoder based recurrent neural network. *Periodicals of Engineering and Natural Sciences*, *6*(1), pp.229-240.

Bansal, G. and Singla, M., 2020. Ensembling of non-linear SVM models with partial least square for diabetes prediction. In *Emerging Trends in Electrical, Communications, and Information Technologies: Proceedings of ICECIT-2018* (pp. 731-739). Springer Singapore.

Chen, M., Wang, Z., Zhao, Z., Zhang, W., Guo, X., Shen, J., Qu, Y., Lu, J., Xu, M., Xu, Y. and Wang, T., 2021, August. Task-wise split gradient boosting trees for multi-center diabetes prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (pp. 2663-2673).

Daghistani, T. and Alshammari, R., 2020. Comparison of statistical logistic regression and random forest machine learning techniques in predicting diabetes. *Journal of Advances in Information Technology Vol*, *11*(2), pp.78-83.

International Diabetes Federation. IDF Diabetes Atlas, 10th edn. Brussels, Belgium: 2021. [Online] Available at: https://www.diabetesatlas.org/ [Accessed 2nd June 2023].

Jakhmola, S. and Pradhan, T., 2015, August. A computational approach of data smoothening and prediction of diabetes dataset. In *Proceedings of the Third International Symposium on Women in Computing and Informatics* (pp. 744-748).

Kalpana, M. and Kumar, A.S., 2011. Fuzzy expert system for diabetes using fuzzy verdict mechanism. *International Journal of Advanced Networking and Applications*, *3*(2), p.1128.

Kumar Dewangan, A. and Agrawal, P., 2015. Classification of diabetes mellitus using machine learning techniques. *International Journal of Engineering and Applied Sciences*, *2*(5), p.257905.

Larabi-Marie-Sainte, S., Aburahmah, L., Almohaini, R. and Saba, T., 2019. Current techniques for diabetes prediction: review and case study. *Applied Sciences*, *9*(21), p.4604.

Maniruzzaman, M., Kumar, N., Abedin, M.M., Islam, M.S., Suri, H.S., El-Baz, A.S. and Suri, J.S., 2017. Comparative approaches for classification of diabetes mellitus data: Machine learning paradigm. *Computer methods and programs in biomedicine*, *152*, pp.23-34.

Madan, P., Singh, V., Chaudhari, V., Albagory, Y., Dumka, A., Singh, R., Gehlot, A., Rashid, M., Alshamrani, S.S. and AlGhamdi, A.S., 2022. An optimization-based diabetes prediction model using CNN and Bi-directional LSTM in real-time environment. *Applied Sciences*, *12*(8), p.3989.

Maniruzzaman, M., Kumar, N., Abedin, M.M., Islam, M.S., Suri, H.S., El-Baz, A.S. and Suri, J.S., 2017. Comparative approaches for classification of diabetes mellitus data: Machine learning paradigm. *Computer methods and programs in biomedicine*, *152*, pp.23-34.

NirmalaDevi, M., alias Balamurugan, S.A. and Swathi, U.V., 2013, March. An amalgam KNN to predict diabetes mellitus. In *2013 IEEE international conference on emerging trends in computing, communication and nanotechnology (ICECCN)* (pp. 691-695). IEEE.

Olaniyi, E.O. and Adnan, K., 2014. Onset diabetes diagnosis using artificial neural network. *Int J Sci Eng Res*, *5*(10), pp.754-759.

Osman, A.H. and Aljahdali, H.M., 2017. Diabetes disease diagnosis method based on feature extraction using K-SVM. *International Journal of Advanced Computer Science and Applications*, *8*(1).

Panwar, M., Acharyya, A., Shafik, R.A. and Biswas, D., 2016, December. K-nearest neighbor based methodology for accurate diagnosis of diabetes mellitus. In *2016 sixth international symposium on embedded computing and system design (ISED)* (pp. 132-136). IEEE.

Patra, R., 2021, February. Analysis and prediction of Pima Indian Diabetes Dataset using SDKNN classifier technique. In *IOP Conference Series: Materials Science and Engineering* (Vol. 1070, No. 1, p. 012059). IOP Publishing.

Ramesh, S., Balaji, H., Iyengar, N.C.S.N. and Caytiles, R.D., 2017. Optimal predictive analytics of pima diabetics using deep learning. *International Journal of Database Theory and Application*, *10*(9), pp.47-62.

Rakshit, S., Manna, S., Biswas, S., Kundu, R., Gupta, P., Maitra, S. and Barman, S., 2017. Prediction of diabetes type-II using a two-class neural network. In *Computational Intelligence, Communications, and Business Analytics: First International Conference, CICBA 2017, Kolkata, India, March 24–25, 2017, Revised Selected Papers, Part II* (pp. 65-71). Springer Singapore.

Srivastava, S., Sharma, L., Sharma, V., Kumar, A. and Darbari, H., 2019. Prediction of diabetes using artificial neural network approach. In *Engineering Vibration,* *Communication and Information Processing: ICoEVCI 2018, India* (pp. 679-687). Springer Singapore.

Soliman, O.S. and AboElhamd, E., 2014. Classification of diabetes mellitus using modified particle swarm optimization and least squares support vector machine. *arXiv preprint arXiv:1405.0549*.

Sridar, K. and Shanthi, D., 2014. Medical Diagnosis System For The Diabetes Mellitus By Using Back Propagation-Apriori Algorithms. *Journal of Theoretical & Applied Information Technology*, *68*(1).

Tan, Y., Chen, H., Zhang, J., Tang, R. and Liu, P., 2022. Early risk prediction of diabetes based on GA-Stacking. Applied Sciences, 12(2), p.632.

WebMD Editorial Contributors, Diagnosis of Diabetes 2023. [Online] Available at: https://www.webmd.com/diabetes/diagnosis-diabetes/ [Accessed 1st July 2023].

Yahyaoui, A., Jamil, A., Rasheed, J. and Yesiltepe, M., 2019, November. A decision support system for diabetes prediction using machine learning and deep learning techniques. In 2019 1st International informatics and software engineering conference (UBMYK)(pp.1-4).IEEE.

Yuvaraj, N. and SriPreethaa, K.R., 2019. Diabetes prediction in healthcare systems using machine learning algorithms on Hadoop cluster. *Cluster Computing*, *22*(Suppl 1), pp.1-9.

9. APPENDIX 1:

Sample Codes:

Importing of all necessary Libraries – Code Sample:

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

import missingno as msno
from datetime import date
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler, RobustScaler

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
# pd.set_option('display.width', 500)
from sklearn.metrics import classification_report  #classification report mainly used for comparing the value of predicted output and actual output
from sklearn.metrics import confusion_matrix, accuracy_score , recall_score , precision_score ,f1_score

import os


from sklearn.linear_model import LogisticRegression, Perceptron, RidgeClassifier, SGDClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import xgboost as xgb
from xgboost import XGBClassifier
import lightgbm as lgb
from lightgbm import LGBMClassifier

from catboost import CatBoostClassifier

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.metrics import confusion_matrix, accuracy_score , recall_score , precision_score
```

Description of the Dataset – Code Sample:

```
df.columns = [col.upper() for col in df.columns]
df.head()

def check_df(dataframe, head=5):
    print("The shape of dataframe:")
    print(dataframe.shape)
    print("The types of feature of dataframe:")
    print(dataframe.dtypes)

    print(dataframe.head(head))

    print("Checks Null any")
    print(dataframe.isnull().sum())
```

```
check_df(df)
def grab_col_names(dataframe, cat_th=10, car_th=20):

    cat_cols = [col for col in dataframe.columns if dataframe[col].dtypes == "O"]
    num_but_cat = [col for col in dataframe.columns if dataframe[col].nunique() < cat_th and
                   dataframe[col].dtypes != "O"]
    cat_but_car = [col for col in dataframe.columns if dataframe[col].nunique() > car_th and
                   dataframe[col].dtypes == "O"]
    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]
   num_cols = [col for col in dataframe.columns if dataframe[col].dtypes != "O"]
    num_cols = [col for col in num_cols if col not in num_but_cat]

 return cat_cols, num_cols, cat_but_car
cat_cols, num_cols, cat_but_car = grab_col_names(df)
num_cols
print(df.isnull().sum())
def target_summary_with_num(dataframe, target, numerical_col):
    print(dataframe.groupby(target).agg({numerical_col: "median"}), end="\n\n\n")
for col in num_cols:
    target_summary_with_num(df,"OUTCOME", col)
df.isnull().sum() # No more missing Values or wrong values
df.describe().T
```

Distribution Graph and Box plotting – Code Sample:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def plot_distplot(df):
    # Iterate over each column in the dataframe
    for column in df.columns:
        # Create a distplot for the current column
        sns.displot(df[column], kde=True)

        # Set plot title and labels
        plt.title(f"Distribution of {column}")
        plt.xlabel(column)
        plt.ylabel("Density")
        # Show the plot
        plt.show()
#Boxplotting code
fig, axes = plt.subplots(2, 4, figsize=(24, 10))
for i, col in enumerate(num_cols):
    sns.boxplot(x=df[col], ax=axes[i//4, i%4])
    axes[i//4, i%4].set_title(col)
plt.show(block=True)
```

Feature Engineering – Code Sample:

```
def feature_engineering(df):
  # Glucose
# BloodPressure
# SkinThickness
# Insulin
# BMI :Based on these figures a mean BMI of 12 as the lower limit for human survival emerges
- a value first proposed by James et al (1988)
# BMI lower than 12 will also be NaN
# 90/60mmHg or less is considered the lowest blood pressure before death.


  df.loc[df["BLOODPRESSURE"]<60, "BLOODPRESSURE"].count()      #121
  df.loc[df["BLOODPRESSURE"]<60, "BLOODPRESSURE"] = 0 # I changed all the values
under 60 with 0 (since they are wrong values)
```

```
  df[df["BMI"]<12]['BMI'].count() #11
  df.loc[df["BMI"]<12,"BMI"] = 0

  cat_cols, num_cols, cat_but_car = grab_col_names(df)
  for col in [col for col in num_cols if col not in "PREGNANCIES"]:
     df[col].replace(0, np.nan, inplace=True)
  return df
```

Missing Value Handle – Code sample:

```
def missing_value_Fillup(df):
  cat_cols, num_cols, cat_but_car = grab_col_names(df)
  for col in [col for col in num_cols if col not in "PREGNANCIES"]:
     df[col].fillna(df.groupby("OUTCOME")[col].transform("median"), inplace = True)
return df
```

Outlier Analysis– Code sample:

```
def outlier_thresholds(dataframe, col_name, q1=0.10, q3=0.90):
    quartile1 = dataframe[col_name].quantile(q1)
    quartile3 = dataframe[col_name].quantile(q3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit
def check_outlier(dataframe, col_name):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name)
    if dataframe[(dataframe[col_name] > up_limit) | (dataframe[col_name] <
low_limit)].any(axis=None):
        return True
    else:
        return False
```

```
check_outlier(df, num_cols)
```

Handling Outliers– Code sample:

```
def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit


def handle_outliers(df):
  cat_cols, num_cols, cat_but_car = grab_col_names(df)
  for col in num_cols:
    replace_with_thresholds(df,col)
  return df
```

Correlation– Code sample:

```
plt.figure(dpi = 120,figsize= (5,4))
mask = np.triu(np.ones_like(df.corr(),dtype = bool))
sns.heatmap(df.corr(),mask = mask, fmt = ".2f",annot=True,lw=1,cmap = 'plasma')
plt.yticks(rotation = 0)
plt.xticks(rotation = 90)
plt.title('Correlation Heatmap')
plt.show(block=True)
```

Categorical and Numerical Analysis– Code sample:

```
def cat_summary(dataframe, col_name, plot=False):
    print(pd.DataFrame({col_name: dataframe[col_name].value_counts(),
                        "Ratio": 100*dataframe[col_name].value_counts()/len(dataframe)}))
    print("##########################################")
if plot:
        sns.countplot(x=dataframe[col_name], data=dataframe)
        plt.show(block=True)
def target_summary_with_cat(dataframe, target, categorical_col):
    print(pd.DataFrame({"TARGET_MEAN":
dataframe.groupby(categorical_col)[target].mean()}), end="\n\n\n")
def num_summary(dataframe, numerical_col, plot=False):
    quantiles = [0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 0.99]
    print(dataframe[numerical_col].describe(quantiles).T)
     if plot:
        dataframe[numerical_col].hist()
        plt.xlabel(numerical_col)
        plt.title(numerical_col)
        plt.show(block=True)
def target_summary_with_num(dataframe, target, numerical_col):
    print(dataframe.groupby(target).agg({numerical_col: "mean"}), end="\n\n\n")
for col in cat_cols:
 cat_summary(df, col, plot=True)
for col in num_cols:
    num_summary(df, col, plot=True)
```

Feature Engineering and encoding- code sample:

```
def binning(df):
  df['GLUCOSE_CAT'] = pd.cut(x = df['GLUCOSE'], bins = [-1,80,140,160,200],
                              labels = ['Hypoglecimia','Normal', 'Impaired_Glucose', 'Diabetic_Glucose'])

  df['AGE_CAT']= pd.cut(df['AGE'], bins = [0, 20, 30 ,50, 81], labels=['Young','Young Adult',
'Adult', 'Old'])

  df.loc[(df['PREGNANCIES'] == 0), 'PREGNANT_CAT']  = 'Never'
  df.loc[(df['PREGNANCIES'] == 1), 'PREGNANT_CAT']  = 'One Time'
  df.loc[(df['PREGNANCIES'] > 1), 'PREGNANT_CAT']   = 'Many Times'
  return df

def feature_generating(df):
  df["AGE*BMI_NEW"] = df["AGE"] * df["BMI"]

  df["NEW_GLUCOSE*INSULIN"] = df["GLUCOSE"] * df["INSULIN"]
  return df

# df=feature_generating(df)
# df.head()

df[num_cols].head()

# Feature Scaling
def Normalization(df):
  rs= RobustScaler()
  cat_cols, num_cols, cat_but_car = grab_col_names(df)
  df[num_cols]= rs.fit_transform(df[num_cols])
  return df

# df=Normalization(df)
# df.head()

#Label Encoding
def one_hot_encoder( df,drop_first=True, dummy_na =False):
    categorical_cols = [col for col in df.columns if 10 >= df[col].nunique() > 2]
    dataframe = pd.get_dummies(df, columns=categorical_cols, drop_first=drop_first,
dummy_na=dummy_na)
    return dataframe

# df = one_hot_encoder(df)
# df.head()
```

Pipeline Preprocessing Steps – code sample:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
# Create the pipeline with FunctionTransformer for each step
preprocessing_pipeline = Pipeline([
    ('feature_engineering', FunctionTransformer(feature_engineering, validate=False)),
    ('missing_value', FunctionTransformer(missing_value_Fillup, validate=False)),
    ('outlier_handler', FunctionTransformer(handle_outliers, validate=False)),

    ('binning', FunctionTransformer(binning, validate=False)),
    ('feature_generation', FunctionTransformer(feature_generating, validate=False)),
    ('normalization', FunctionTransformer(Normalization, validate=False)),
    ('one_hot_encoding', FunctionTransformer(one_hot_encoder, validate=False)),
])

# Apply the pipeline to your DataFrame
processed_df = preprocessing_pipeline.fit_transform(df)
print(processed_df.head())
```

Train Test Split – sample Code:

```
y = processed_df["OUTCOME"]
X = processed_df.drop(["OUTCOME"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=17)
y.value_counts()
```

Handling Imbalanced Dataset (SMOTE) – sample Code:

```
#Using SMOTE to balance the data
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
ros_chd_plot=y_train.value_counts().plot(kind='bar')
plt.show()
```

Neural Network Model – Code sample:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l1
from keras.callbacks import EarlyStopping

# Define the model
model1 = Sequential()
model1.add(Dense(64, input_dim=X_train.shape[1], activation='relu',
kernel_regularizer=l1(0.001)))
model1.add(Dense(32, activation='relu', kernel_regularizer=l1(0.001)))

model1.add(Dense(1, activation='sigmoid'))

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1)
```

```
# Compile the model
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model1.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model1.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
print(model1.predict(X_test).shape)

y_pred1=model1.predict(X_test)
y_pred=(y_pred1>0.5).astype("int32")
```

Multiple Machine Learning Model Training (with cross validation and grid search) – Code sample:

```
# Create a list of models
models = [LogisticRegression(), SVC(), KNeighborsClassifier(n_neighbors=6),
        DecisionTreeClassifier(max_leaf_nodes=3, random_state=0, criterion='entropy'),
        RandomForestClassifier(max_features=0.5, max_depth=15, random_state=1),
        XGBClassifier(), GaussianNB(), SGDClassifier(),
        RidgeClassifier(), BaggingClassifier(), LGBMClassifier(random_state=0),
        CatBoostClassifier(verbose=False, random_state=0),
        ]


# List of model names
model_names = ["LogisticRegression", "SVC", "KNeighborsClassifier",
"DecisionTreeClassifier", "RandomForestClassifier", "XGBClassifier", "GaussianNB",
"SGDClassifier", "RidgeClassifier", "BaggingClassifier", "LGBMClassifier",
"CatBoostClassifier"]
# Create a list of dictionaries for the hyperparameters to be tuned for each model
```

```
param_grids = [
# Logistic Regression
  {'penalty': ['l2'], 'C': [0.1, 1, 10]},

# Support Vector Machine (SVC)
{'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']},

# K-Nearest Neighbors (KNN)
{'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']},

# Decision Tree Classifier
{'criterion': ['gini', 'entropy'], 'max_depth': [None, 5, 10, 15], 'min_samples_split': [2, 5, 10]},

# Random Forest Classifier
{'n_estimators': [200, 300], 'max_depth': [None, 5], 'min_samples_split': [2, 5], 'max_features':
[0.3, 0.5]},



# XGBoost Classifier
{'eta': [0.01, 0.1, 0.3], 'max_depth': [3, 6, 9], 'subsample': [0.7, 0.8, 0.9], 'colsample_bytree':
[0.7, 0.8, 0.9]},

# Gaussian Naive Bayes
{},

# Stochastic Gradient Descent (SGD) Classifier
{'alpha': [0.0001, 0.001, 0.01], 'penalty': ['l1', 'l2', 'elasticnet']},

# Ridge Classifier
{'alpha': [0.1, 0.5, 1.0], 'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']},

# Bagging Classifier
{'n_estimators': [50, 100, 150], 'max_samples': [0.5, 0.7, 0.9], 'max_features': [0.5, 0.7, 0.9]},
# LightGBM Classifier
{'num_leaves': [10, 20, 30], 'learning_rate': [0.01, 0.05, 0.1, 0.3], 'max_depth': [-1, 5, 10]},

 # CatBoost Classifier
{'iterations': [100, 150, 200], 'learning_rate': [0.01, 0.05, 0.1], 'depth': [4, 6, 8]}
]

best_params_dict = {}
best_models = []
# Loop through each model and perform GridSearchCV
for i, model in enumerate(models):
model_name = model_names[i]
print("Tuning hyperparameters for Model", model_name)
grid_search = GridSearchCV(estimator=model, param_grid=param_grids[i], cv=5)
grid_search.fit(X_train, y_train)  # Replace X_train and y_train with your training data
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("Best parameters:", best_params)
print("Best score:", best_score)
print()

# Store the best parameters in the dictionary

best_params_dict[model_name] = best_params




# Create a new instance of the model with the best parameters and add it to the best_models list
if model_name == 'CatBoostClassifier':
  best_model = model.__class__(**best_params, verbose =0)
else:
  best_model = model.__class__(**best_params)
best_models.append(best_model)


for model, model_name in zip(best_models, model_names):
  print(model_name)
  if model_name == 'CatBoostClassifier':

  model.fit(X_train, y_train, verbose =0) #fit those best model
  else:

  model.fit(X_train, y_train) #fit those best model
```

Evaluation Matrix for all models– Code sample:

```
from sklearn.metrics import confusion_matrix
# Iterate over models
for model, model_name in zip(best_models, model_names):
    # Assuming you have test data and predictions for each model
    y_pred = model.predict(X_test)

    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Print confusion matrix
    print("Evaluation metrices for", model_name)
    print(classification_report(y_test, y_pred))
     # Calculate and print evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Accuracy for {model_name}: {accuracy*100:.2f}")
    print(f"Precision for {model_name}: {precision*100:.2f}")
    print(f"Recall for {model_name}: {recall*100:.2f}")
    print(f"F1 Score for {model_name}: {f1*100:.2f}")
    print()
    print()
```

Confusion Matrix Sample Code:

```
def plot_cm(cm, model_name):
  # Define class labels (replace with your own if necessary)
  class_labels = ["Heart patient", "Normal patient",]
# Plot confusion matrix
  plt.figure(figsize=(8, 6))
  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)
  #plt.title("Confusion Matrix for ", model_name)
  plt.xlabel("Predicted Labels")
  plt.ylabel("True Labels")
  plt.show()
from sklearn.metrics import confusion_matrix
# Iterate over models
for model, model_name in zip(best_models, model_names):
    # Assuming you have test data and predictions for each model
    y_pred = model.predict(X_test)
    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    # Print confusion matrix
    print("Confusion Matrix for", model_name)
    print(cm)
    print()
    plot_cm(cm,model_name)
    print()
```
ROC curve and Learning Curve – sample Code:
```
#ROC CURVE
def roc_plot(y_pred):
      # Compute the false positive rate, true positive rate, and thresholds for ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)

    # Compute the area under the ROC curve
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) - ' + model_name)
    plt.legend(loc="lower right")
    plt.show()



from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np
```

```
# Iterate over models
for model, model_name in zip(best_models, model_names):
    # Get the corresponding prediction for the model
    y_pred = model.predict(X_test)

    roc_plot(y_pred)

    print()
    print()
    print()
    print()
```

```
"""# Learning curve"""
def plot_learning_curve(model):
# Compute learning curve
if model_name == 'CatBoostClassifier':
train_sizes, train_scores, test_scores = learning_curve(model, X,y,cv=5,verbose=0)
else:
train_sizes, train_scores, test_scores = learning_curve(model, X, y, cv=5)
train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
```

```
 # Plot learning curve
    plt.figure()
    plt.title('Learning Curve - ' + model_name)
    plt.xlabel("Training Examples")
    plt.ylabel("Accuracy Score")
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + 
train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + 
test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation Score")
    plt.legend(loc="best")
    plt.show()


from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np


# Iterate over models
for model, model_name in zip(best_models, model_names):
    plot_learning_curve(model)
```

Feature Importance and Model Interpretation (SHAP) – Code Sample:

```
#Feature Importance
```

```
def plot_importance(model, features, num=len(X)):
    feature_imp = pd.DataFrame({'Value': model.feature_importances_, 'Feature':
features.columns})
    plt.figure(figsize = (10, 10))
    sns.set(font_scale = 1)
    sns.barplot(x = "Value", y = "Feature", data = feature_imp.sort_values(by = "Value",
ascending = False)[0:num])
    plt.title('Features')
    plt.tight_layout()
    plt.show(block=True)
plot_importance(best_models[4], X_train)  # show the feature importance generated by random
forest as it shows better result
```

```
"""# Model Interpretation (SHAP)"""
import shap
import matplotlib
explainer = shap.TreeExplainer(best_models[4])
# calculate shap values. This is what we will plot.
shap_values = explainer.shap_values(X_test)
# custom colour plot
colors = ["#9bb7d4", "#0f4c81"]
cmap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)
shap.summary_plot(shap_values[1], X_test,cmap=cmap,alpha=0.4)
shap.dependence_plot('INSULIN', shap_values[1], X_test,
interaction_index="INSULIN",cmap=cmap,alpha=0.4,show=False)
plt.title("Age dependence plot",loc='left',fontfamily='serif',fontsize=15)
plt.ylabel("SHAP value for the 'INSULIN' feature")
plt.show()
```

```
def model_evaluation(model,model_name,y_pred):
    #Evaluation metrices
    print("Evaluation metrices for", model_name)
    print(classification_report(y_test, y_pred))
    print()
    print(f"Accuracy for {model_name}: {accuracy*100:.2f}")
    print(f"Precision for {model_name}: {precision*100:.2f}")
    print(f"Recall for {model_name}: {recall*100:.2f}")
    print(f"F1 Score for {model_name}: {f1*100:.2f}")
    print()
    # Print confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix for", model_name)
    print(cm)
    print()
    plot_cm(cm,model_name)
    print()
    #plot the roc curve
    roc_plot(y_pred)
    print()
    #plot the learning curve
    plot_learning_curve(model)
    print()
    print()
```

Voting Classifier – Code Sample:

```
from sklearn.ensemble import StackingClassifier
from sklearn.svm import NuSVC
#soft voting classifier
from sklearn.ensemble import VotingClassifier
estimator = []
# Indices of models to be used as voting estimators
voting_indices = [4, 5, 9, 10, 11]
for i in voting_indices:
 estimator.append((model_names[i],best_models[i]))
# Create a voting classifier with 'soft' voting using the selected models
voting_classifier = VotingClassifier(estimators=estimator, voting='soft')
# Train and evaluate the voting classifier
voting_classifier.fit(X_train, y_train)
y_pred = voting_classifier.predict(X_test)

model_evaluation(voting_classifier,'voting classifier soft',y_pred)
#hard voting classifier
from sklearn.ensemble import VotingClassifier
estimator = []



# Indices of models to be used as voting estimators
voting_indices = [4, 5, 9, 10, 11]
for i in voting_indices:
  estimator.append((model_names[i],best_models[i]))
# Create a voting classifier with 'soft' voting using the selected models
```

```
voting_classifier = VotingClassifier(estimators=estimator, voting='hard')
# Train and evaluate the voting classifier
voting_classifier.fit(X_train, y_train)
y_pred = voting_classifier.predict(X_test)
model_evaluation(voting_classifier,'voting classifier hard',y_pred)
```

Stacking Classifier – Code Sample:

```
"""# StackingClassifier"""
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import classification_report
estimator = []
# Indices of models to be used as voting estimators
stacking_indices = [4, 5, 9, 10, 11]
for i in stacking_indices:
  estimator.append((model_names[i],best_models[i]))

# Create the stacking classifier with 'soft' voting
stacking_classifier = StackingClassifier(estimators=estimator,  final_estimator=NuSVC())

# Train and evaluate the stacking classifier
stacking_classifier.fit(X_train, y_train)
y_pred = stacking_classifier.predict(X_test)

model_evaluation(stacking_classifier,'stacking classifier soft',y_pred)

# Define the hyperparameters to tune
param_grid = {
    'final_estimator__nu': [0.3, 0.5, 0.7],
    'final_estimator__kernel': ['linear', 'rbf', 'sigmoid']
}

# Perform grid search for hyperparameter tuning
```

```
stacking_grid_search = GridSearchCV(estimator=stacking_classifier, param_grid=param_grid, cv=5)
stacking_grid_search.fit(X_train, y_train)

# Get the best model
best_model = stacking_grid_search.best_estimator_

# Evaluate the best model on the test set
accuracy = best_model.score(X_test, y_test)
print("Accuracy:", accuracy)
```

Appendix 2:

- **Progress to date**

Date	Task Complete	Notes
08/02/2023	1st Meeting with supervisor	Introductory Session
15/02/2023	Project proposal	Planning about my project proposal
20/03/2023	Interim Progress Report (IPR)	Planning about my IPR
01/05/2023	Interim Progress Report (IPR) submitted	
05/05/2023	Download 10 papers and read 4 papers about ML in prediction	
15/05/2023	Build a voting classifier and stacking classifier with those models which showed significant performances individually.	
25/05/2023	Download more papers and starting research of it.	
1/6/2023	Find the best models by tuning best parameters of all the models implemented	
15/6/2023	Complete the coding part and finalize the model, sent it to my supervisor through email	
1/7/2023	Finish the Introduction and literature review writing and sent it to my supervisor through email	

10/07/2023	Meeting with my supervisor through email	
12/07/2023	Meeting with supervisor	Deeply discussion about my whole work and my research
14/07/2023	Meeting with my supervisor	About my coding part and work progress
15/7/2023	Complete the methodology part writing.	
1/8/2023	Complete the result and discussion part writing with the observed value by the final best model	
15/8/2023	Write the abstract and conclusion and revise the whole paper work.	
28/8/2023	Final editing and submit the FPR	