Machine Learning, I DATS 6202

Final Report

Stroke Prediction

Instructor - Amir Jafari

Presented by Group 2:

Shikha Sharma

Aron Rock

Sanjana

Data- 04/29/2023

ABSTRACT

Stroke is a life-threatening condition in which the blood flow to the heart is disrupted, leading to Heart Attack. According to the World Health Organization (WHO), stroke is the leading cause of death and disability worldwide. Early recognition of the warning signs of a stroke and prompt medical attention can reduce the severity of the condition. This study aims to develop machine learning models to predict the likelihood of a stroke. Different algorithms, including Logistic Regression (LR), Decision Tree (DT) Classification, Random Forest (RF) Classification, and SVM, were trained on the open-access Stroke Prediction dataset. The results of this study indicate that the models developed in this investigation are more reliable compared to previous studies and provide promising accuracy for predicting strokes.

Table of Contents

1. Introduction

Stroke, a critical global health concern, stands as the second most common cause of death and the third most common cause of disability worldwide, according to the World Health Organization (WHO). Every year, approximately 800,000 individuals in the United States alone experience a stroke, with nearly 75% of these cases being first-time occurrences. Early detection and prevention are crucial, as 80% of strokes can be averted with timely intervention and proper education on risk factors and warning signs.

The primary objective of our project is to leverage the capabilities of machine learning (ML) techniques for the prediction of strokes, an area that has received comparatively less attention than the prediction of heart attacks. By analyzing risk factors such as age, systolic blood pressure, BMI, cholesterol, diabetes, smoking status and intensity, physical activity, alcohol consumption, and family history, we aim to develop a model capable of accurately predicting stroke risk and informing preventive measures.

This project presents the implementation of eight ML classification methods in the context of stroke prediction, contributing to the growing body of knowledge in this critical area of healthcare. Our findings have the potential to enhance early intervention strategies, promote healthier lifestyles, and ultimately save lives by reducing the impact of strokes on individuals and communities worldwide.

1.1 Problem Statement

Stroke, the second leading cause of death worldwide, poses a significant health burden for individuals and national healthcare systems alike. Numerous potentially modifiable risk factors for stroke exist, such as hypertension, cardiac disease, diabetes, glucose metabolism dysregulation, atrial fibrillation, and various lifestyle factors. Our project aims to harness machine learning principles to analyze large existing datasets effectively, allowing for accurate stroke prediction based on these modifiable risk factors. The goal is to develop a that delivers personalized stroke risk warnings to users, along with tailored messages addressing lifestyle modifications to mitigate stroke risk factors. This approach has the potential to transform stroke prevention and management, ultimately reducing the impact of this life-threatening condition on individuals and healthcare systems worldwide.

2. Dataset Description

The "healthcare-dataset-stroke-data" is a stroke prediction dataset from Kaggle that contains 5,110 observations (rows) with 12 attributes (columns). Each observation corresponds to one patient, and the attributes represent variables related to the health status of each patient.

Dataset Description:

1. ID: Patient identifier
2. Gender: Patient's gender (Male, Female, or Other)
3. Age: Patient's age (quantitative variable)
4. Hypertension: Presence of hypertension (0 = no, 1 = yes; categorical variable)
5. Heart Disease: Presence of heart disease (0 = no, 1 = yes; categorical variable)
6. Ever Married: Marital status (categorical variable)
7. Work Type: Patient's occupation (children, government job, never worked, private, self-employed; categorical variable)
8. Residence Type: Patient's residence (urban, rural; categorical variable)
9. Avg. Glucose Level: Average glucose level in the blood (quantitative variable)
10. BMI: Body Mass Index (quantitative variable)
11. Smoking Status: Smoking habits (formerly smoked, never smoked, smokes; categorical variable)
12. Stroke: Stroke history (0 = no stroke risk, 1 = stroke risk; target variable)

3. Experiment Setup

This section provides an overview of the dataset, a block diagram, a flow diagram, and evaluation metrics, as well as the process and methodology employed in the project.

3.1. Proposed System

The data has become available for model construction once it has been processed. A preprocessed dataset and machine learning techniques are needed for the model construction. The methods used include Logistic Regression (LR), Random Forest (RF) classification, Gradient Boosting, XGBoost, **MLP Classifier, Keras Classifier,** Support Vector Machine, and K-Nearest Neighbors. After developing eight different models, performance metrics such as accuracy score, precision score, recall score, and F1 score are employed to compare them. The block diagram of the designed system is illustrated in Figure 1.

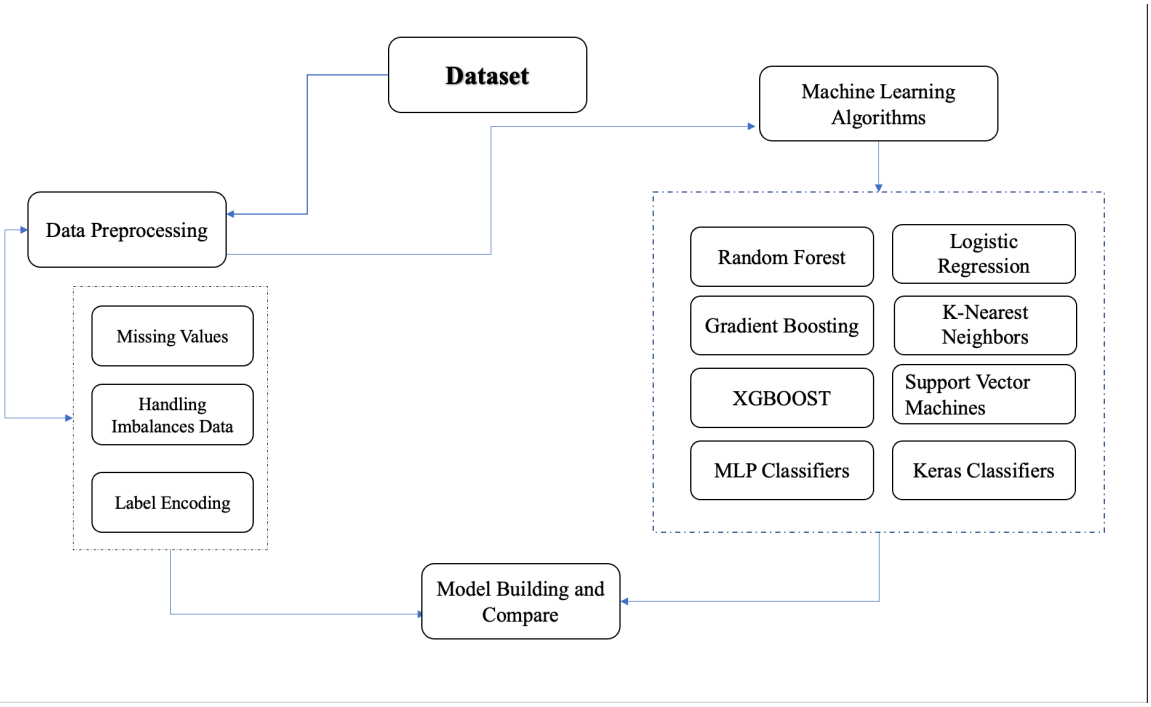


Figure 1 Block diagram

3.2 Dataset

The number of non-stroke cases (0) is significantly higher than stroke cases (1), with only 249 rows having a value of 1, while 4,861 rows have a value of 0. This imbalance may affect model accuracy, necessitating data preprocessing to balance the dataset. Figure 2 illustrates the total number of stroke and non-stroke records in the target column before preprocessing.

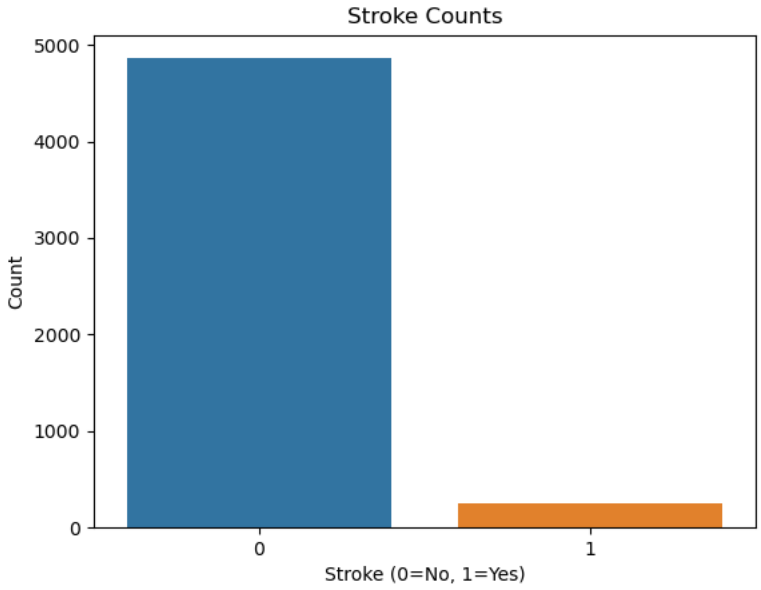


Figure 2 Total number of strokes and normal data.

As evident from Figure 2, the dataset is imbalanced. To address this issue, the Synthetic Minority Over-sampling Technique (SMOTE) has been applied to balance the dataset.

3.3 Preprocessing

Before developing a model, data preprocessing is essential to eliminate unwanted noise and outliers from the dataset, which could otherwise interfere with the model's intended training. This step involves addressing any factors that may hinder the model's efficiency. After acquiring the relevant dataset, it must be cleaned and prepared for model development. As previously mentioned, the dataset contains twelve features. Firstly, the 'id' column is excluded, as it does not contribute to model construction. The dataset is then checked for missing values, which are filled in if found. In this case, the missing values in the 'BMI' column are filled using the mean of the column's data.

Label encoding is used to convert the dataset's string literals into integer values that can be processed by the computer. Since computers typically operate on numerical data, strings must be converted into integers. The collected dataset has five columns with string data types. During label encoding, all strings are encoded, and the entire dataset is transformed into a collection of numbers. The stroke prediction dataset is highly imbalanced, with 5,110 rows, of which 249 indicate a stroke risk and 4,861 indicate no stroke risk. Training a machine learning model on such data may yield high accuracy; however, other performance metrics like precision and recall might be insufficient. If the imbalanced data is not adequately addressed, the results may be inaccurate, leading to ineffective predictions. Consequently, it is crucial to handle the imbalanced data to achieve an efficient model. The SMOTE technique was employed for this purpose. Figure 3 demonstrates the balanced output column of the dataset.

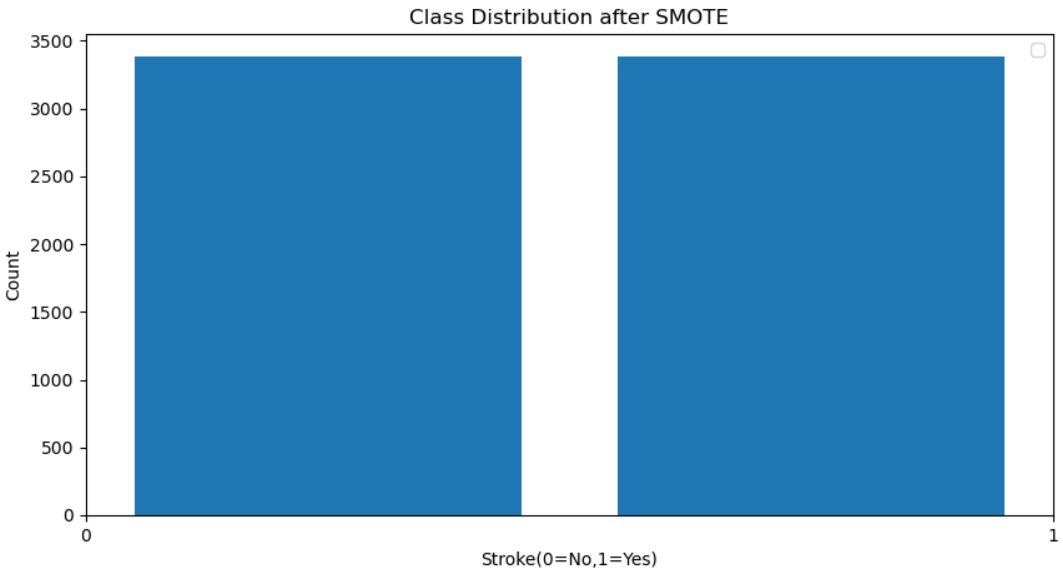


Figure 3 Output column after Preprocessing

The next stage is to construct the model after finishing data preparation and managing the imbalanced dataset. To enhance the accuracy and efficiency of this task, the data is divided into training and testing sets, with a ratio of 80% training data and 20% testing data. Following the split, the model is trained using various classification techniques. In this study, Logistic Regression (LR), Random Forest (RF) classification, Gradient Boosting, XGBoost, **MLP Classifier, Keras Classifier,** Support Vector Machine, and K-Nearest Neighbors are the algorithms employed.

3.4. Proposed Algorithms

The most common disease identified in the medical field is stroke, which is on the rise year after year. Using the publicly accessible stroke prediction dataset, the study measured four commonly used machine learning methods for predicting stroke recurrence, which are as follows:

1. Random Forest
2. Gradient Boosting Classifier
3. XGBOOST
4. Logistic Regression
5. K-Nearest Neighbors
6. Support vector Machine
7. MLP Classifier
8. Keras Classifier

4. Machine Learning Algorithms

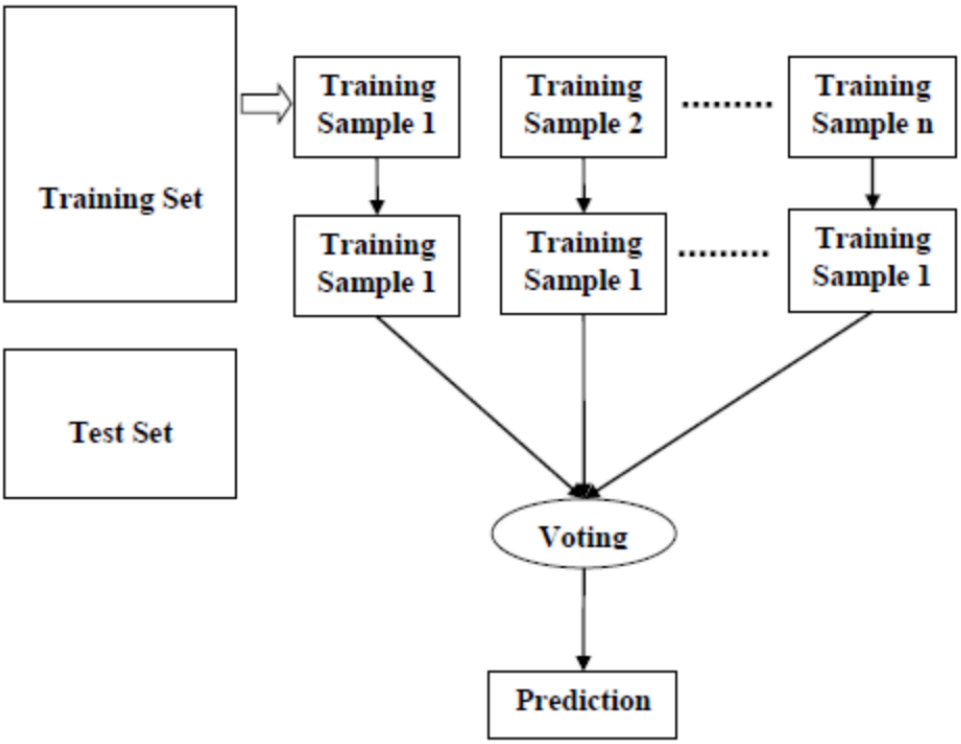
Background and development of algorithms Machine learning techniques have shown promise in various applications, including healthcare. For this project, we employ eight different ML classification methods to predict stroke risk. These methods are chosen due to their diverse strengths and proven track record in classification tasks.

4.1. Random Forest

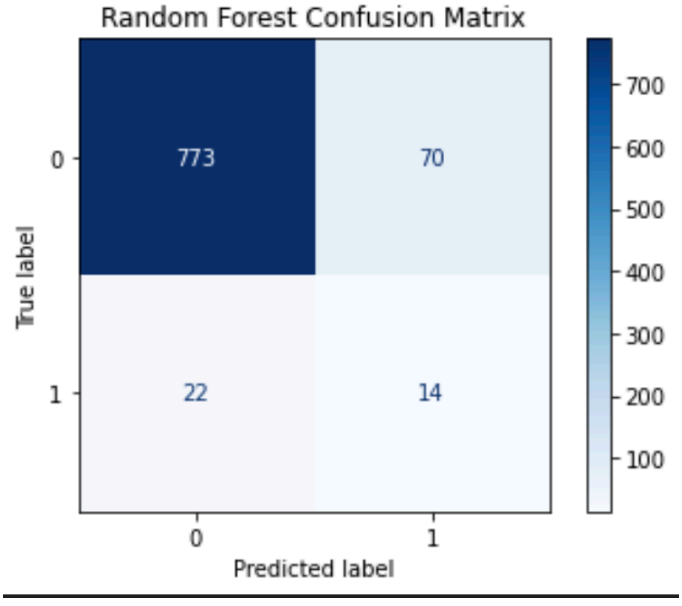
Random forest is a supervised learning algorithm that is used for both classifications as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means a more robust forest. Similarly, a random forest algorithm creates decision trees on data samples and then gets the prediction from each of them, and finally selects the best solution by means of voting. It is an ensemble method that is better than a single decision tree because it reduces the over-fitting by averaging the result.

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, predicts the final output.

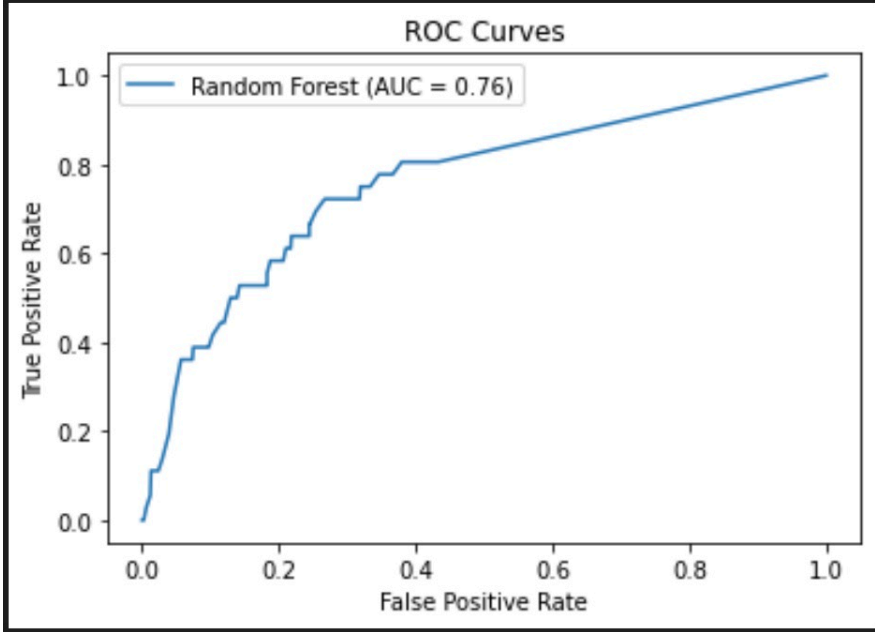
The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on most results, the Random Forest classifier predicts the final decision.



The result of our project for Random Forest



```
random-forest confusion matrix
 [[773  70]
 [ 22  14]]
random-forest Classification report
              precision    recall  f1-score   support

           0       0.97      0.92      0.94       843
           1       0.17      0.39      0.23        36

    accuracy                           0.90       879
   macro avg       0.57      0.65      0.59       879
weighted avg       0.94      0.90      0.91       879

Accuracy-Random-forest
: 0.8953356086461889
```



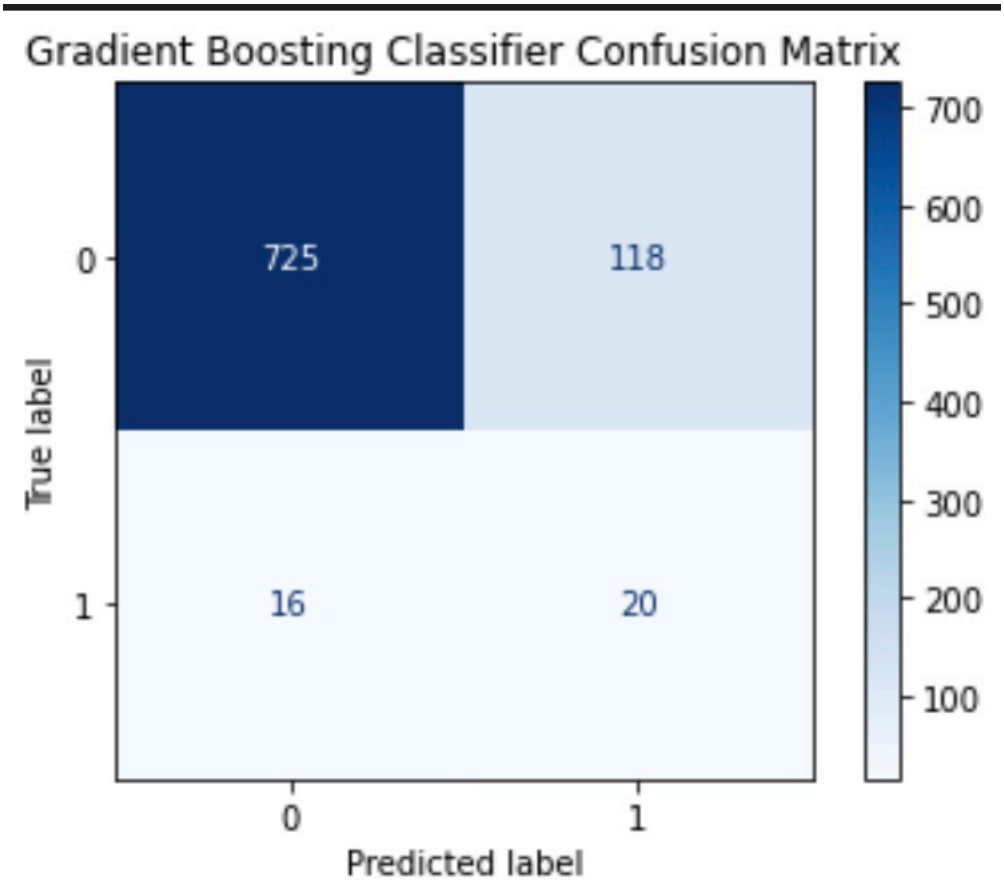
4.2. Gradient Boosting Classifier

Gradient boosting is one of the variants of ensemble methods where you create multiple weak models and combine them to get better performance.
The gradient boosting algorithm is one of the most powerful algorithms in the field of machine learning. As we know that the errors in machine learning algorithms are broadly classified into two categories i.e., Bias Error and Variance Error. As gradient boosting is one of the boosting algorithms it is used to minimize the bias error of the model.

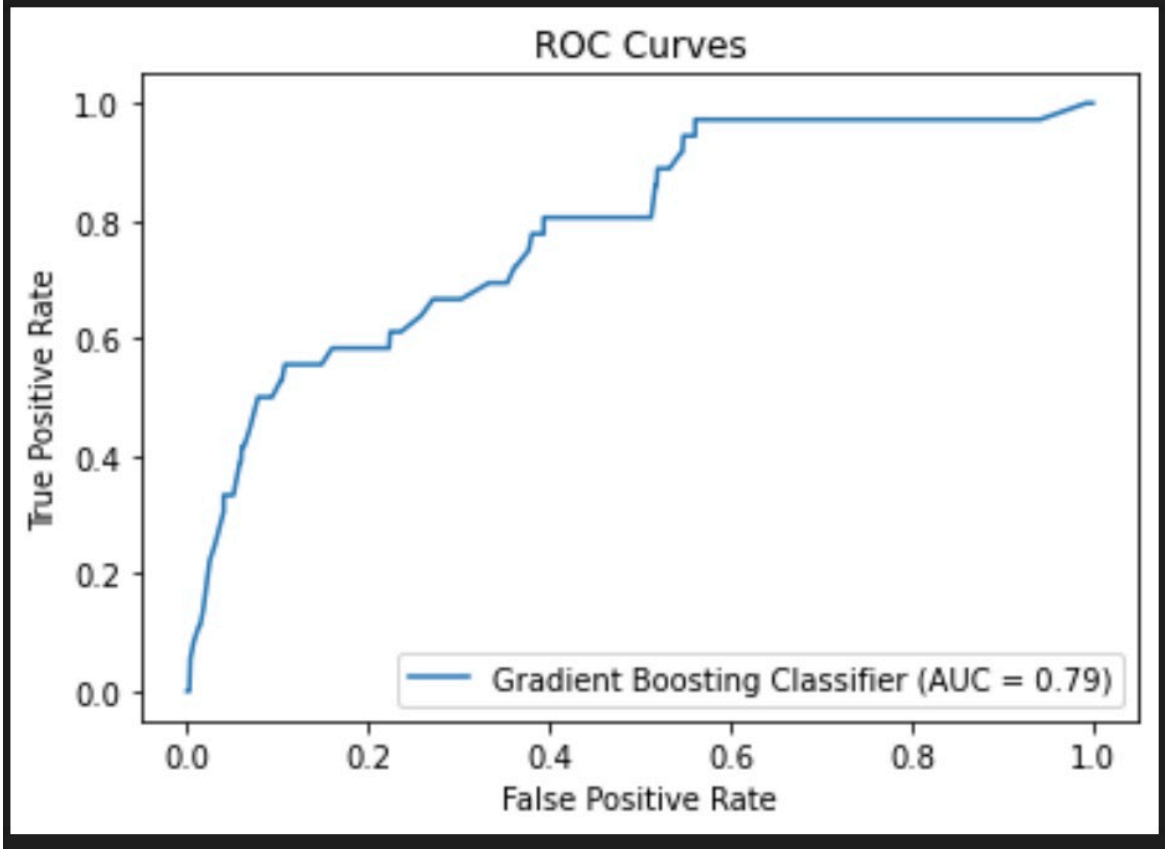
Gradient boosting algorithm can be used for predicting not only continuous target variable (as a Regressor) but also categorical target variable (as a Classifier). When it is used as a regressor, the cost function is Mean Square Error (MSE) and when it is used as a classifier then the cost function is Log loss.

Gradient boosting is a highly robust technique for developing predictive models. It applies to several risk functions and optimizes the accuracy of the model's prediction. It also resolves multicollinearity problems where the correlations among the predictor variables are high.

The result of our project for Gradient Boosting Classifier

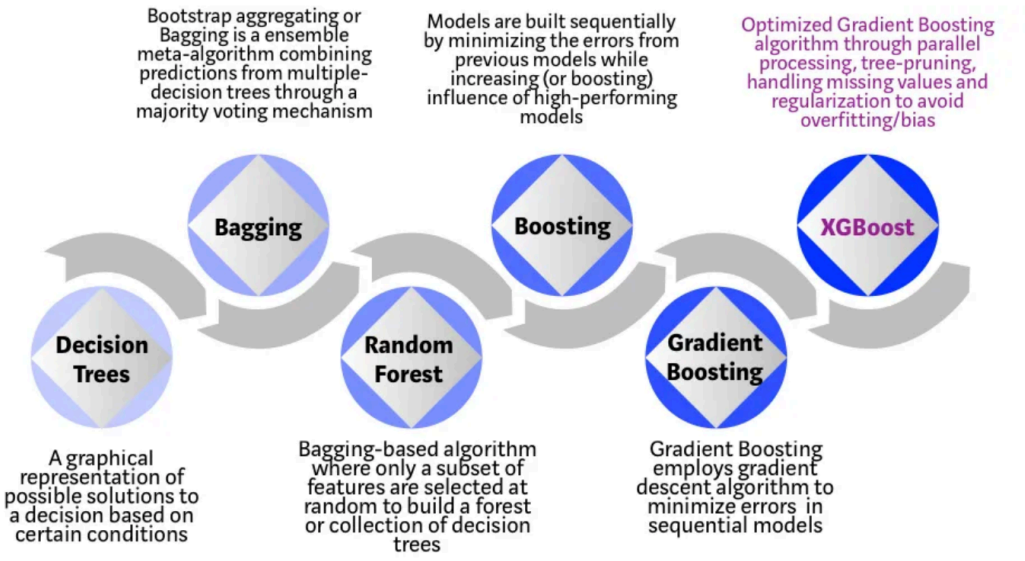


```
Accuracy-Gradient Boosting Classifier
: 0.8475540386803185
Gradient Boosting Classifier confusion matrix-
 [[725 118]
 [ 16  20]]
Gradient Boosting Classifier Classification report
              precision    recall  f1-score   support

           0       0.98      0.86      0.92       843
           1       0.14      0.56      0.23        36

    accuracy                           0.85       879
   macro avg       0.56      0.71      0.57       879
weighted avg       0.94      0.85      0.89       879
```



4.3. XGBOOST

XGBoost algorithm is an extended version of the gradient boosting algorithm. It is basically designed to enhance the performance and speed of a Machine Learning model.
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient-boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.



Each step of the evolution of tree-based algorithms can be viewed as a version of the interview process.

Decision Tree: Every hiring manager has a set of criteria such as education level, number of years of experience, and interview performance. A decision tree is analogous to a hiring manager interviewing candidates based on his or her own criteria.

Bagging: Now imagine instead of a single interviewer, now there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.

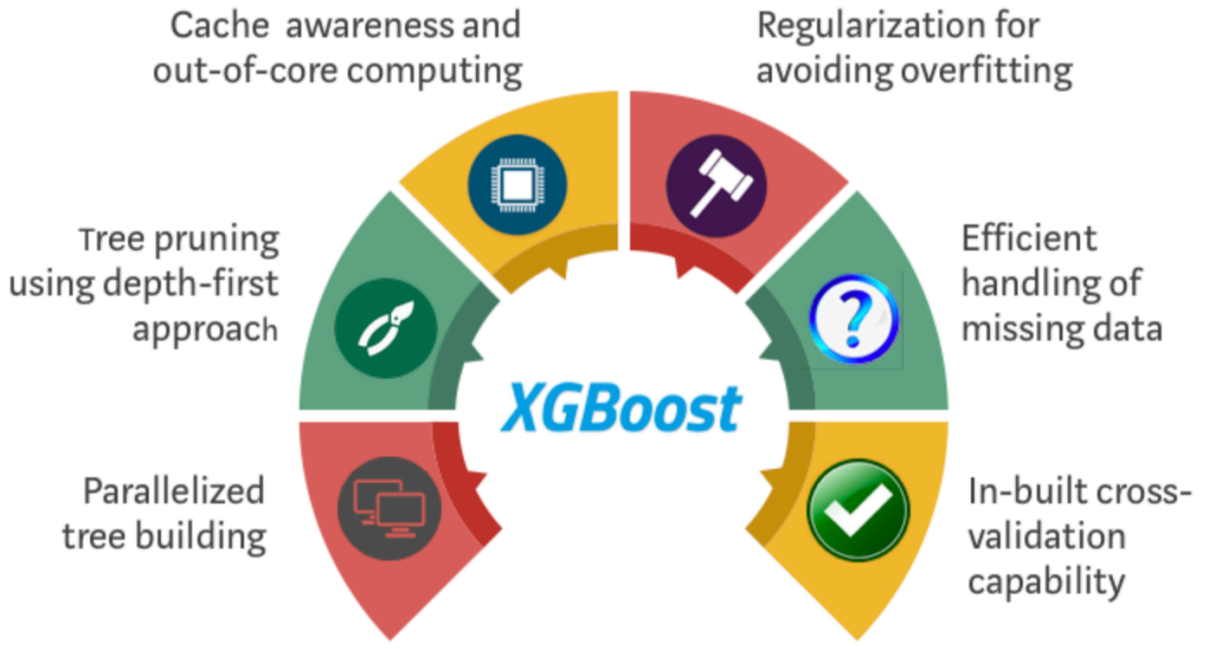
Random Forest: It is a bagging-based algorithm with a key difference wherein only a subset of features is selected at random. In other words, every interviewer will only test the interviewee on certain randomly selected qualifications (e.g., a technical interview for testing programming skills and a behavioral interview for evaluating non-technical skills).

Boosting: This is an alternative approach where each interviewer alters the evaluation criteria based on feedback from the previous interviewer. This 'boosts' the efficiency of the interview process by deploying a more dynamic evaluation process.

Gradient Boosting: A special case of boosting where errors are minimized by a gradient descent algorithm e.g., the strategy consulting firms leverage by using case interviews to weed out less qualified candidates.

XGBoost: Think of XGBoost as gradient boosting on 'steroids' (well it is called 'Extreme Gradient Boosting' for a reason!). It is a perfect combination of software and hardware optimization techniques to yield superior results using fewer computing resources in the shortest amount of time.

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



Parallelization: XGBoost employs a parallelized implementation for sequential tree building. This is achieved by interchanging the order of the nested loops used to construct base learners: the

outer loop enumerating leaf nodes of a tree, and the inner loop calculating features. The original nesting order hinders parallelization, as the outer loop cannot begin until the computationally demanding inner loop is completed. By initializing through a global scan of all instances and sorting with parallel threads, the loop order switch enhances algorithmic performance by counterbalancing parallelization overheads.

Tree Pruning: In contrast to the GBM framework's greedy stopping criterion, which depends on the negative loss criterion at the point of the split, XGBoost uses the 'max_depth' parameter first and begins pruning trees backward. This 'depth-first' approach significantly improves computational performance.

Hardware Optimization: XGBoost is designed to efficiently utilize hardware resources. It achieves cache awareness by allocating internal buffers in each thread to store gradient statistics. Additional enhancements, such as 'out-of-core' computing, optimize disk space usage when handling large data frames that do not fit into memory.

Algorithmic Enhancements:

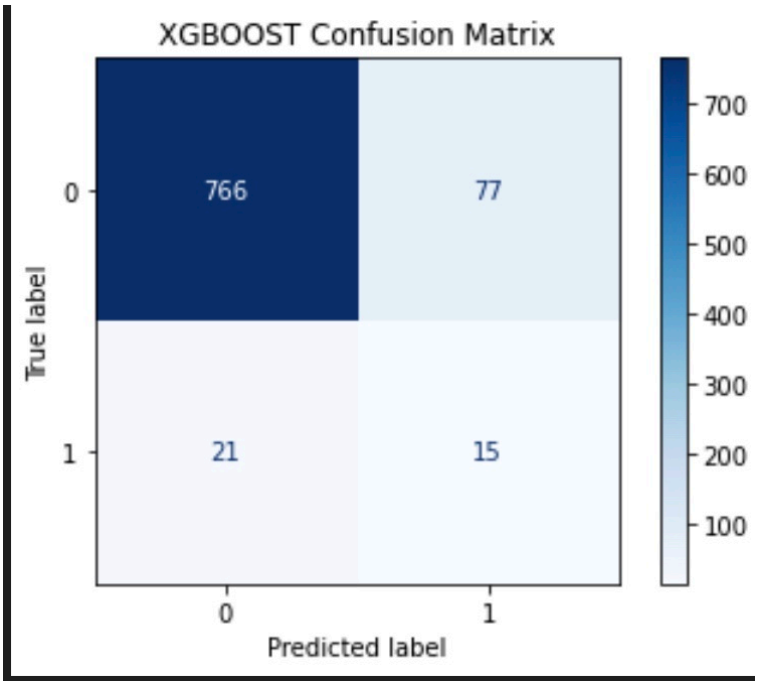
Regularization: It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.

Sparsity Awareness: XGBoost naturally admits sparse features for inputs by automatically ‘learning’ the best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently.

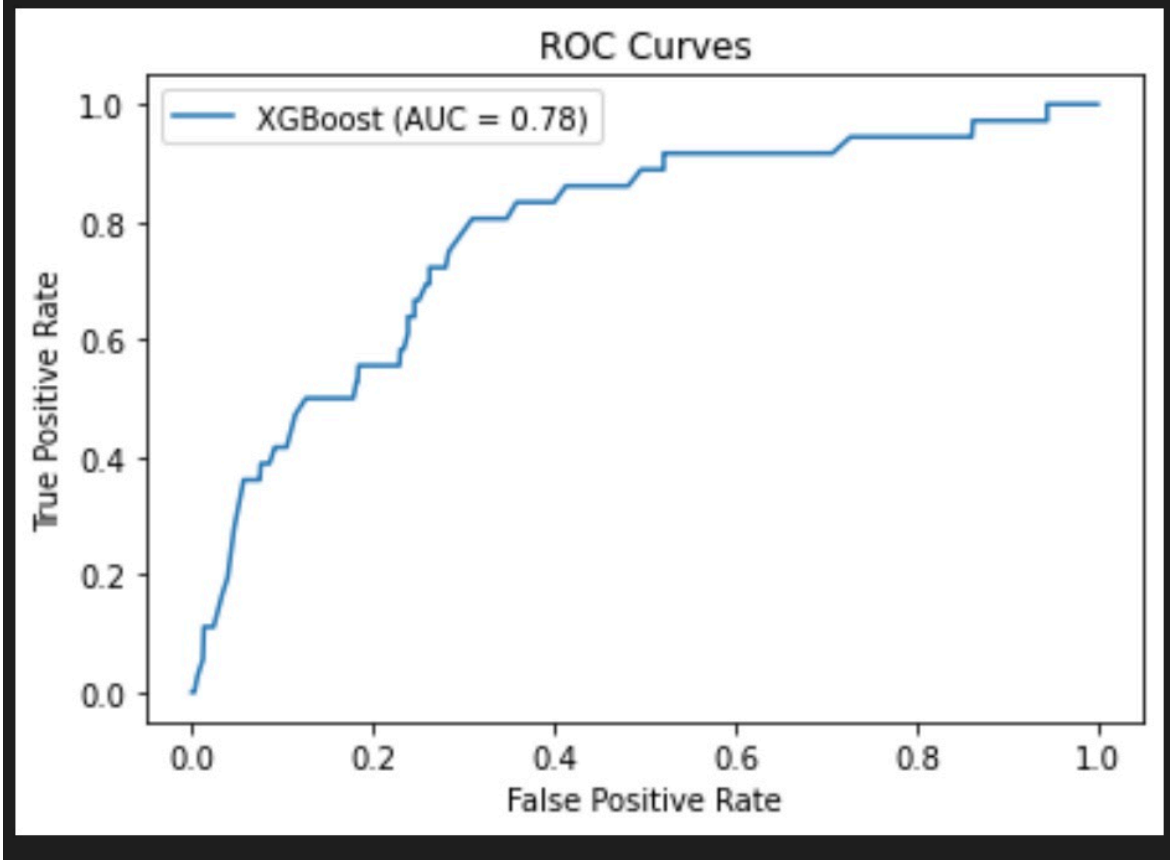
Weighted Quantile Sketch: XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.

Cross-validation: The algorithm comes with a built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

The result of our project for XGBoost


XGBOOST Confusion Matrix
766
77
21
15
0
1
True label
0
1
Predicted label
700
600
500
400
300
200
100

```
XGBOOST confusion matrix-
 [[766  77]
 [ 21  15]]
XGBOOST Classification report
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       843
           1       0.16      0.42      0.23        36

    accuracy                           0.89       879
   macro avg       0.57      0.66      0.59       879
weighted avg       0.94      0.89      0.91       879

Accuracy-XGBOOST
: 0.888509670079636
```



Cross Validation Scores:

```
Random Forest Accuracy: 0.9057097969636316
Gradient Boosting Classifier Accuracy: 0.8488043316755963
XGBoost Accuracy: 0.9051180809873003
```

4.4. Logistic Regression

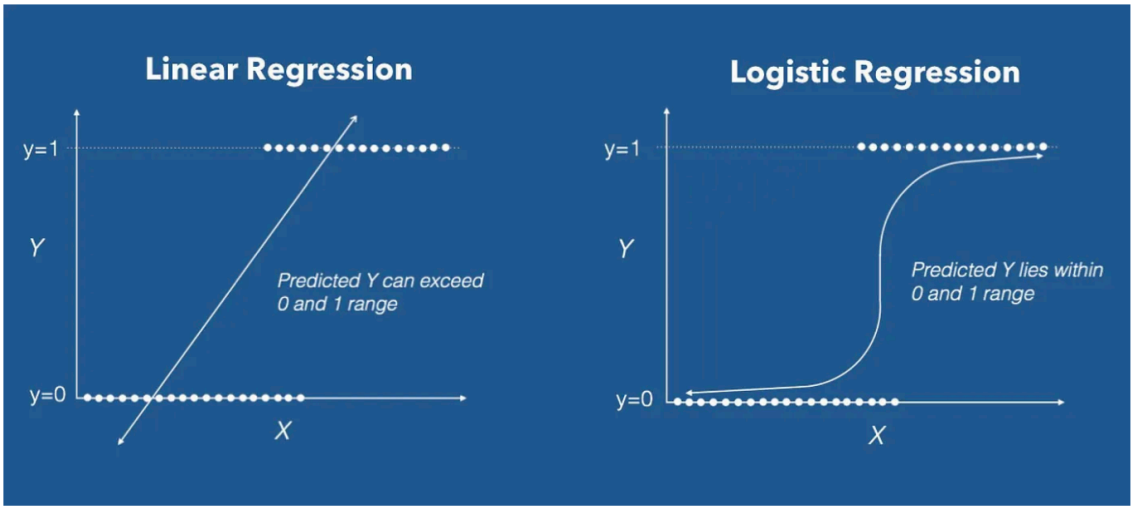
The logistic function, also known as the sigmoid function, is an S-shaped curve that maps any real-valued number to a value between 0 and 1. It is given by the formula:

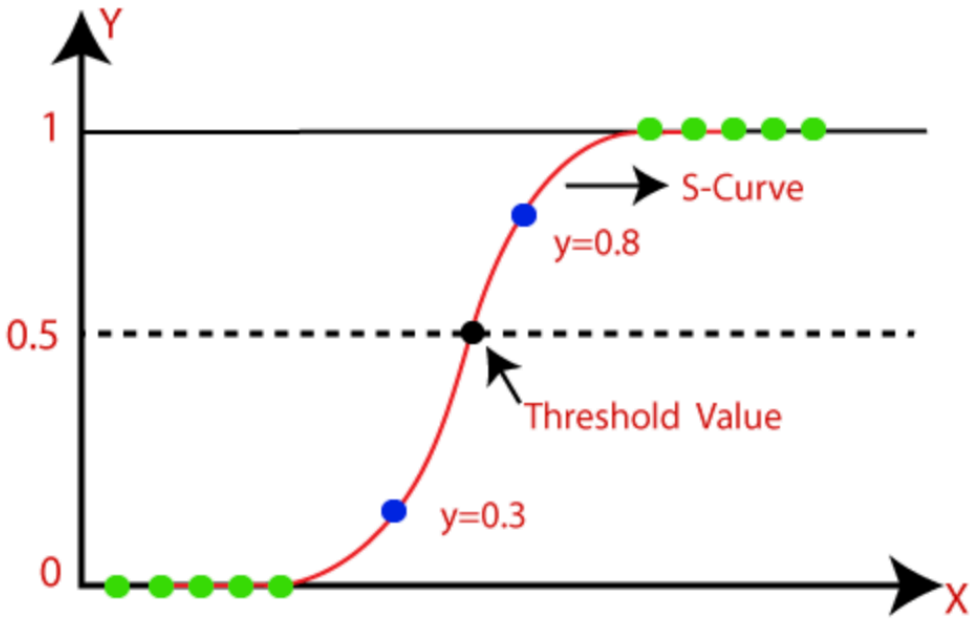
f(x) = 1 / (1 + e^(-x))

where x is the input, e is the base of the natural logarithm, and f(x) is the output of the logistic function.
The logistic function is used in logistic regression to model the relationship between the independent variables and the probability of a certain outcome. The output of the logistic function represents the probability that an observation belongs to a particular class (e.g., 1 or 0, Yes or No, True or False).
In logistic regression, the linear combination of input features (independent variables) is passed through the logistic function to produce a probability value. The equation can be written as:

p(y=1) = 1 / (1 + e^(-(b0 + b1 * x1 + b2 * x2 + ... + bn * xn)))

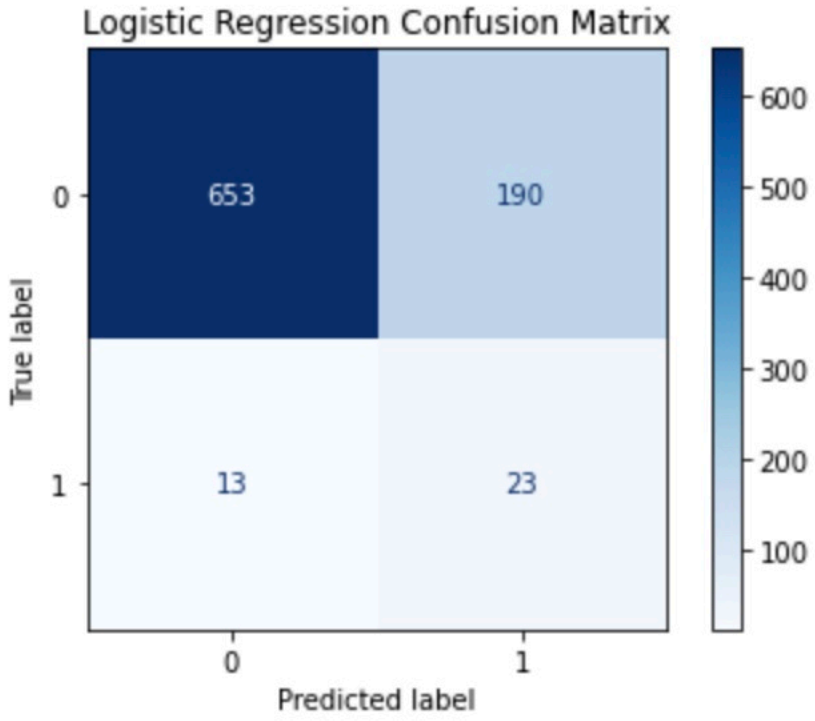
where p(y=1) is the probability of the dependent variable being equal to 1, b0 is the bias term or the y-intercept, b1, b2, ..., bn are the coefficients of the independent variables x1, x2, ..., xn and e is the base of the natural logarithm.
The logistic regression model calculates the best-fitting coefficients (b0, b1, b2, ..., bn) using maximum likelihood estimation, which aims to maximize the likelihood that the observed data comes from the modeled distribution.
By applying a threshold to the output probabilities, logistic regression can be used to classify observations into two classes. For example, if the probability output is greater than 0.5, the observation can be classified as class 1; otherwise, it is classified as class 0.
Logistic regression is widely used in various applications, such as spam detection, medical diagnosis, and marketing campaign optimization. Its ability to provide probabilities and classify new data using continuous and discrete datasets makes it an essential machine-learning algorithm.

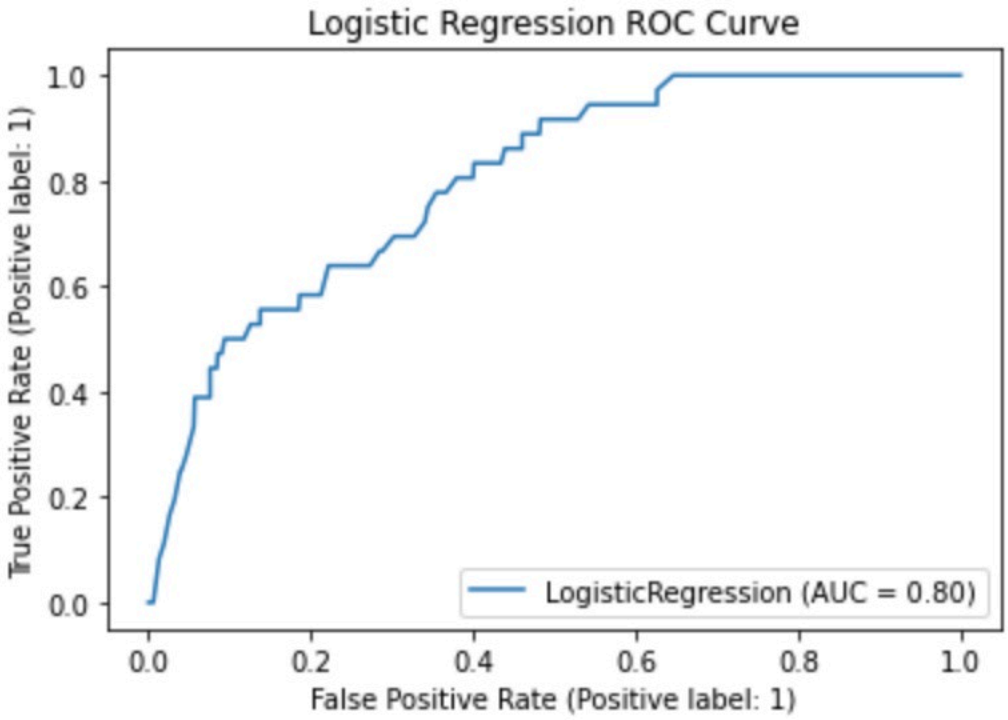




The result of our project for Logistic Regression

```
Logistic Regression Accuracy: 0.7690557451649602
[[653 190]
 [ 13  23]]
              precision    recall  f1-score   support

           0       0.98      0.77      0.87       843
           1       0.11      0.64      0.18        36

    accuracy                           0.77       879
   macro avg       0.54      0.71      0.53       879
weighted avg       0.94      0.77      0.84       879
```

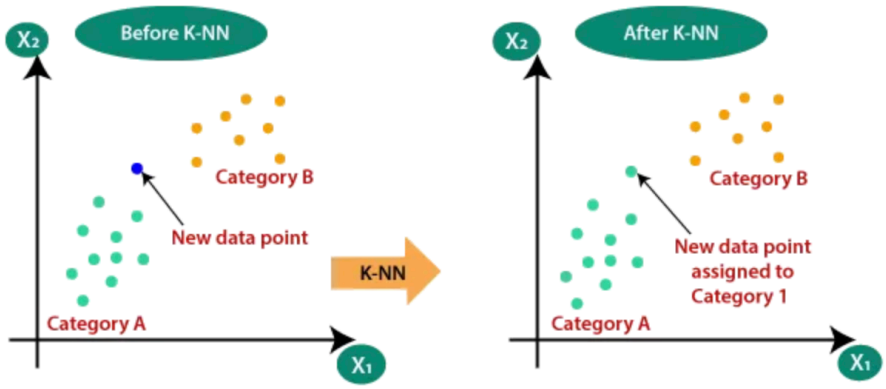




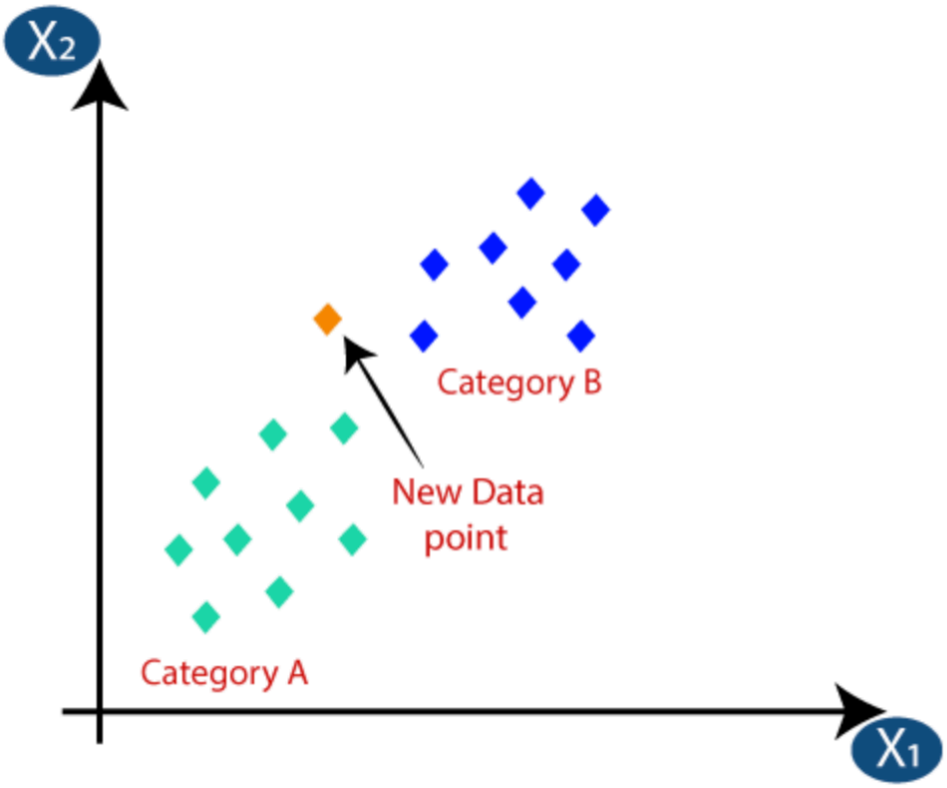
4.5. K-Nearest Neighbors

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and the class that holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

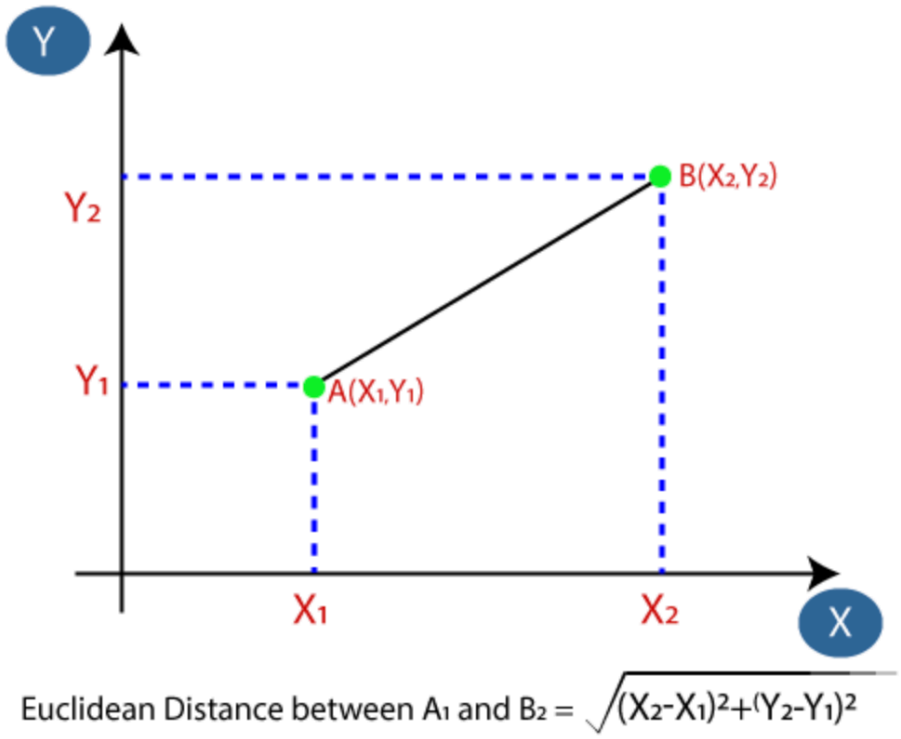
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories? To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



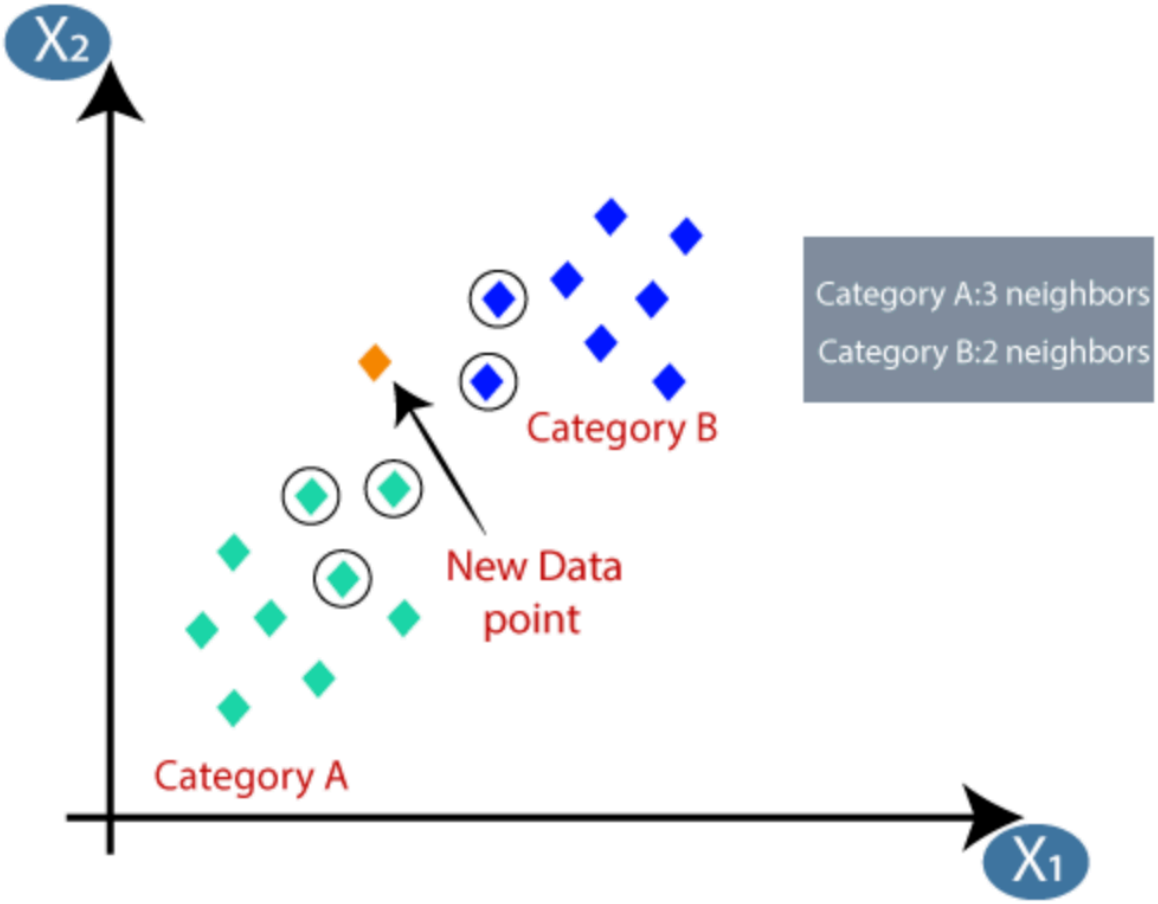
Suppose we have a new data point, and we need to put it in the required category. Consider the below image:



Firstly, we will choose the number of neighbors, so we will choose the k=5.
Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



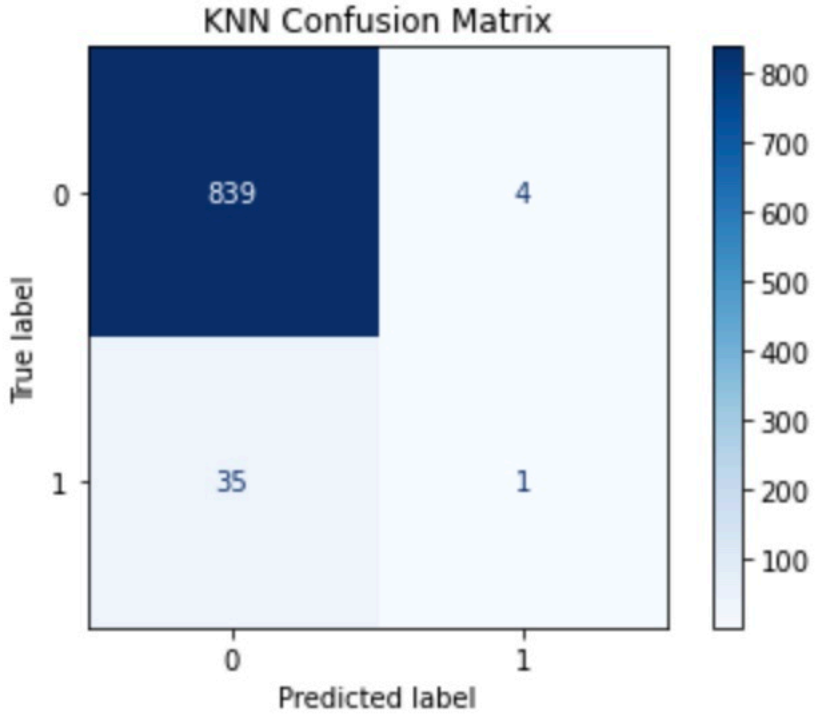
As we can see the 3 nearest neighbors are from Category A, hence this new data point must belong to Category A.

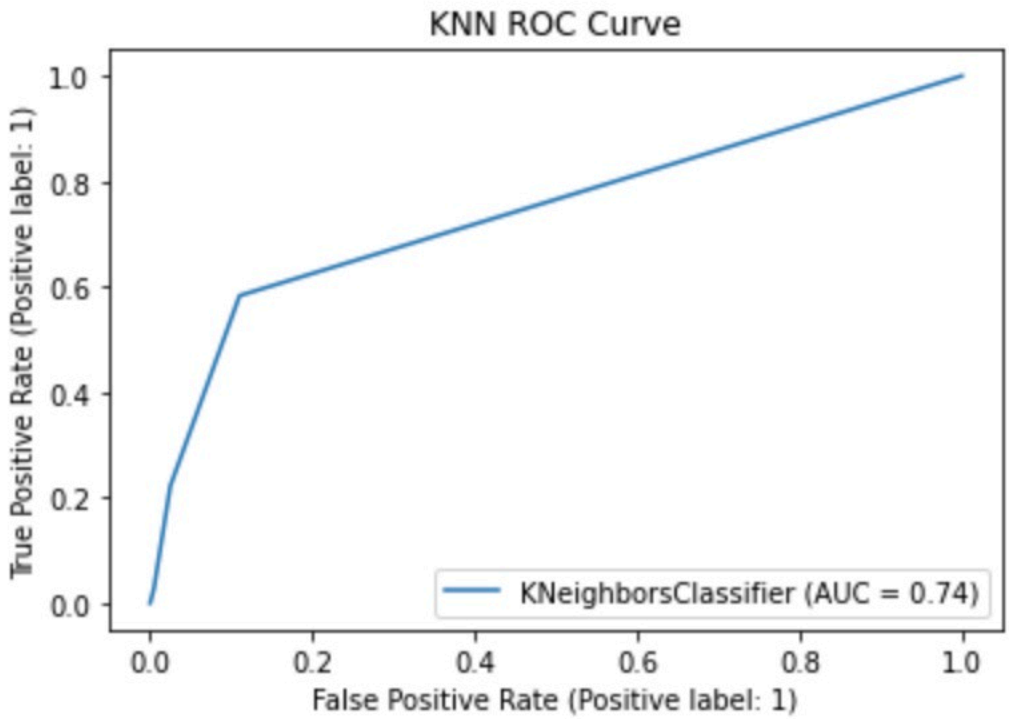
Selecting the value of K in the K-NN algorithm:

1. There is no way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
2. A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model. Large values for K are good.

The result of our project for KNN

```
K-Nearest Neighbors Accuracy: 0.9556313993174061
[[839   4]
 [ 35   1]]
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       843
           1       0.20      0.03      0.05        36

    accuracy                           0.96       879
   macro avg       0.58      0.51      0.51       879
weighted avg       0.93      0.96      0.94       879
```





4.6. Support vector Machine

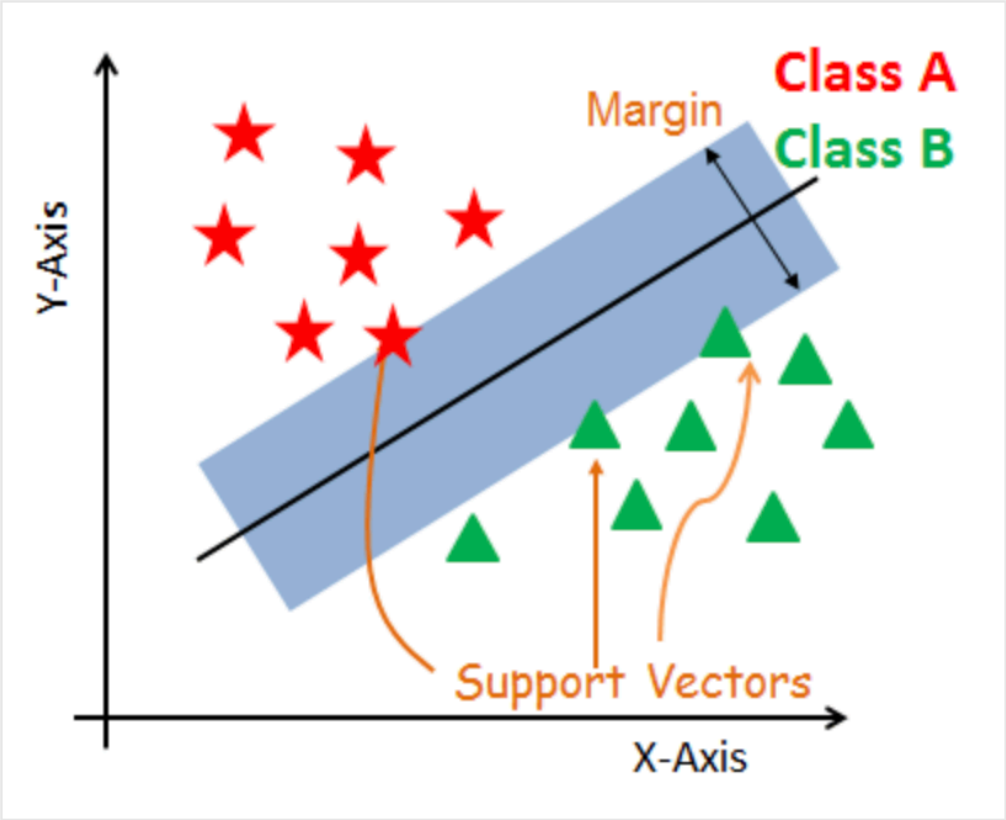
Support Vector Machine (SVM) is a widely used supervised learning algorithm, predominantly employed for classification tasks, but also applicable to regression problems in machine learning. The objective of the SVM algorithm is to determine the optimal decision boundary or hyperplane that separates an n-dimensional space into distinct classes. This enables the accurate categorization of new data points in the future.

SVM selects the most critical points or vectors that facilitate the construction of the hyperplane. These critical points are referred to as support vectors, which give the algorithm its name, Support Vector Machine.

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

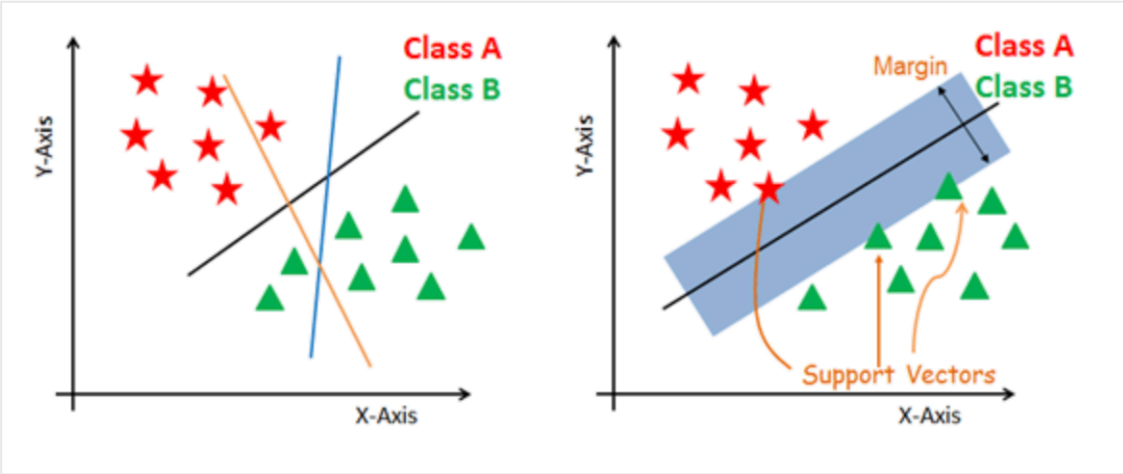
A hyperplane is a decision plane that separates a set of objects having different class memberships.

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.



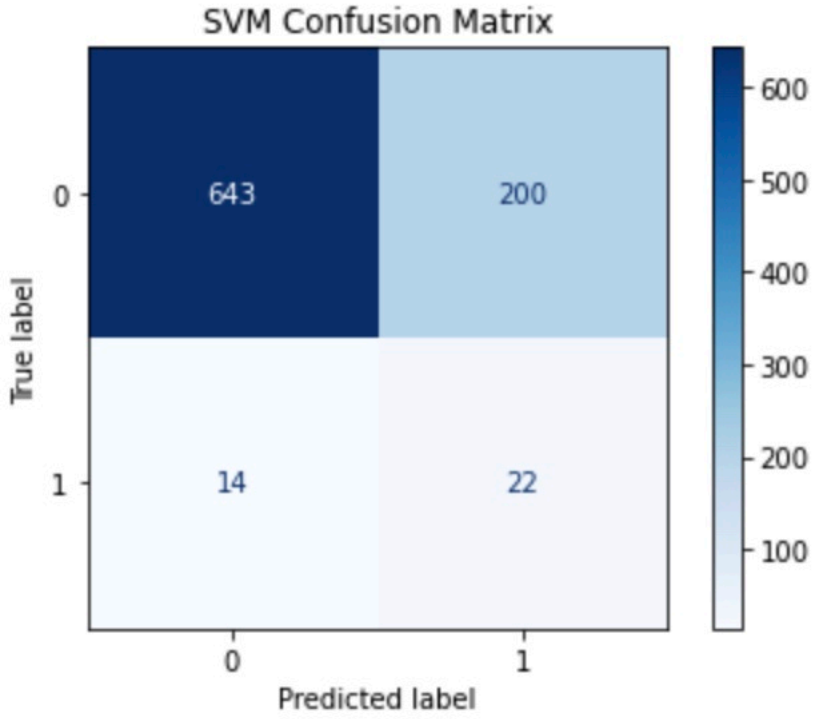
The primary goal of SVM is to optimally separate the given dataset. The margin is defined as the distance between the nearest points of each class. The objective is to choose a hyperplane that maximizes the margin between support vectors in the dataset. SVM identifies the maximum margin hyperplane through the following steps:

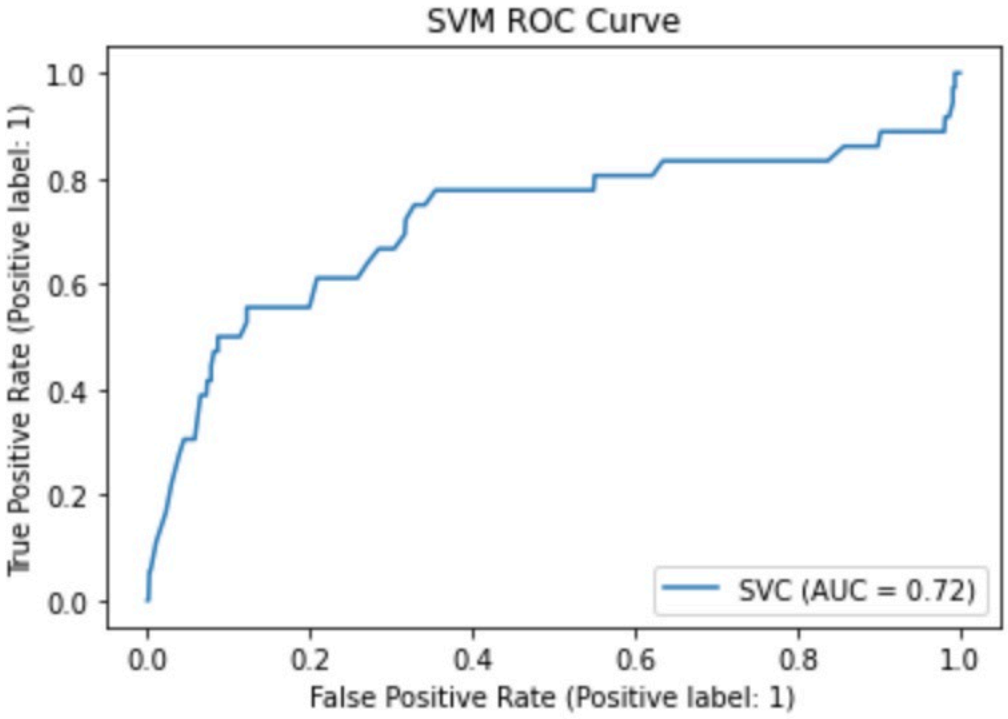
1. Generate hyperplanes that best separate the classes. The left-hand side figure illustrates three hyperplanes (black, blue, and orange). The blue and orange hyperplanes have higher classification errors, while the black hyperplane correctly separates the two classes.
2. Choose the hyperplane that maximizes the distance from the nearest data points of each class, as depicted in the right-hand side figure.



The result of our project for SVM

```
Support Vector Machine Accuracy: 0.7565415244596132
[[643 200]
 [ 14  22]]
              precision    recall  f1-score   support

           0       0.98      0.76      0.86       843
           1       0.10      0.61      0.17        36

    accuracy                           0.76       879
   macro avg       0.54      0.69      0.51       879
weighted avg       0.94      0.76      0.83       879
```





Cross Validation Scores

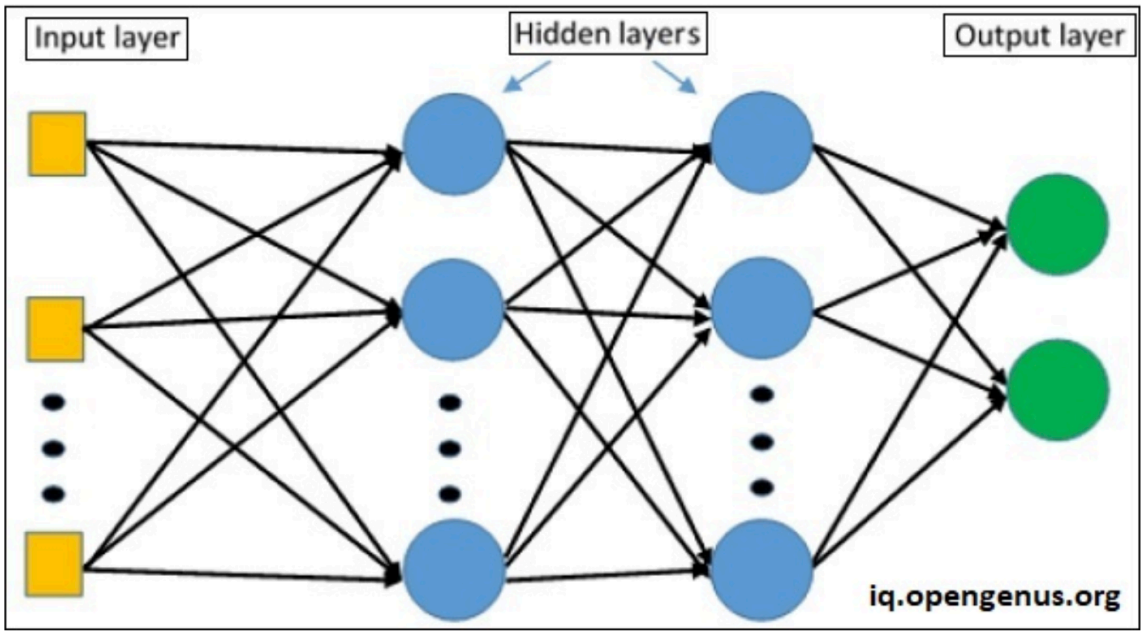
```
✓ #Cross Validation ···

Logistic Regression Accuracy: 0.7898311380699745
K-Nearest Neighbors Accuracy: 0.7882058856948075
Support Vector Machine Accuracy: 0.7979619885852133
```

4.7. MLP Classifier

Multi-Layer Perceptron (MLP) is a type of artificial neural network that is widely used for various machine learning tasks such as classification and regression. It is called a multi-layered perceptron because it has many layers of nodes (known as artificial neurons) that connect to each other. In this article, we will explore the main features and components of MLP, its architectural and training processes, and its programs in various fields. In the decades that followed, MLP was largely overshadowed by other neural network models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). However, in recent years, MLP has seen a resurgence of interest due to its simplicity, versatility, and effectiveness in solving a variety of complex problems.

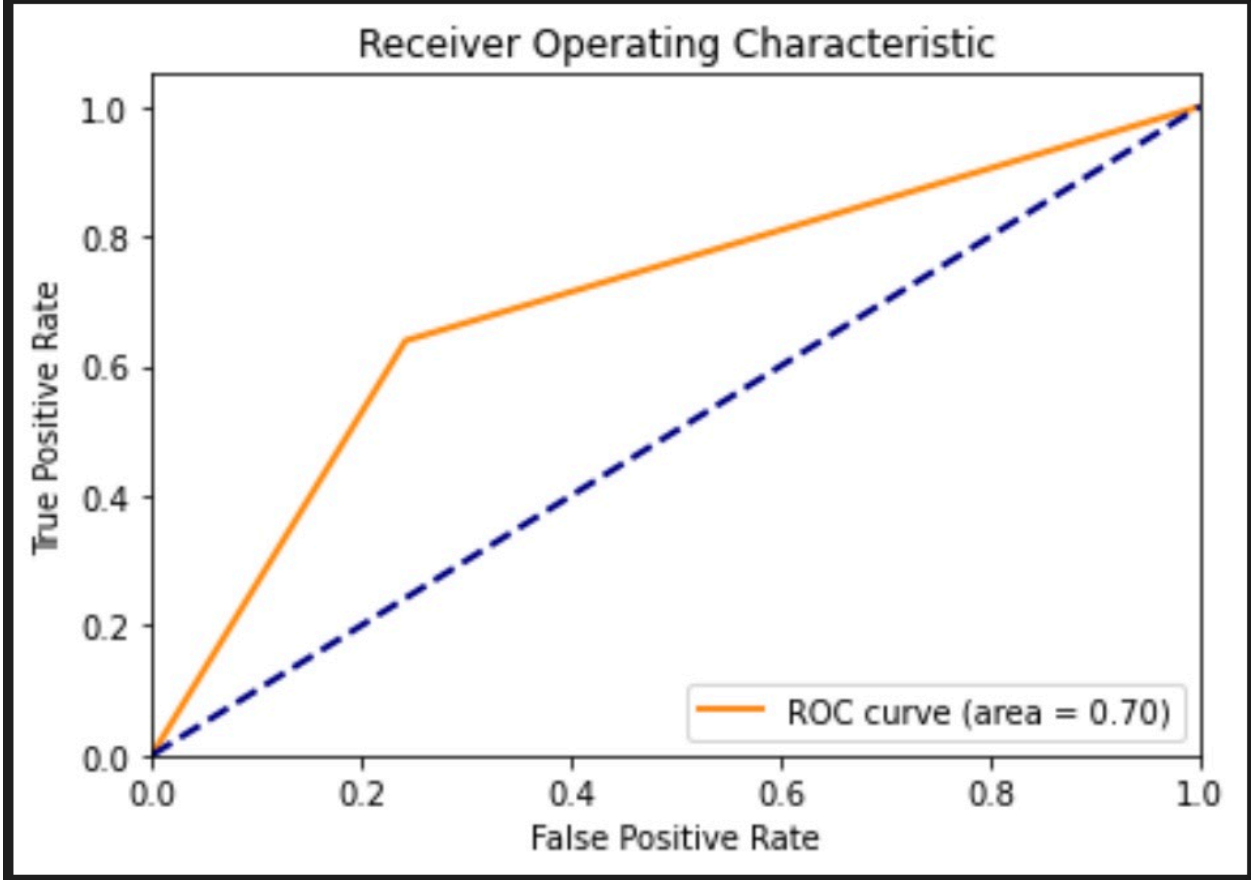
The architecture of an MLP comprises three main components: the input layer, hidden layer(s), and output layer.
Input Layer: Receives input data and forwards it to the hidden layer(s). Contains nodes equal to the number of input features.
Hidden Layer(s): Transform input data for the output layer. Composed of nodes connected through weights and biases, with adjustable layer and node counts. Common activation functions include sigmoid, tanh, and ReLU.
Output Layer: Produces final output using a transformed representation from hidden layer(s). Has nodes corresponding to classes or continuous values, depending on the task. Employs task-specific activation functions like SoftMax for classification and linear for regression.

Overall, the architecture of MLP plays a crucial role in its ability to solve complex problems. By having multiple hidden layers and nodes, MLP can learn complex non-linear relationships between the input and output data. The number of the hidden layer(s) and the number of nodes in each layer, as well as the choice of activation functions, can be adjusted to improve performance for specific tasks.



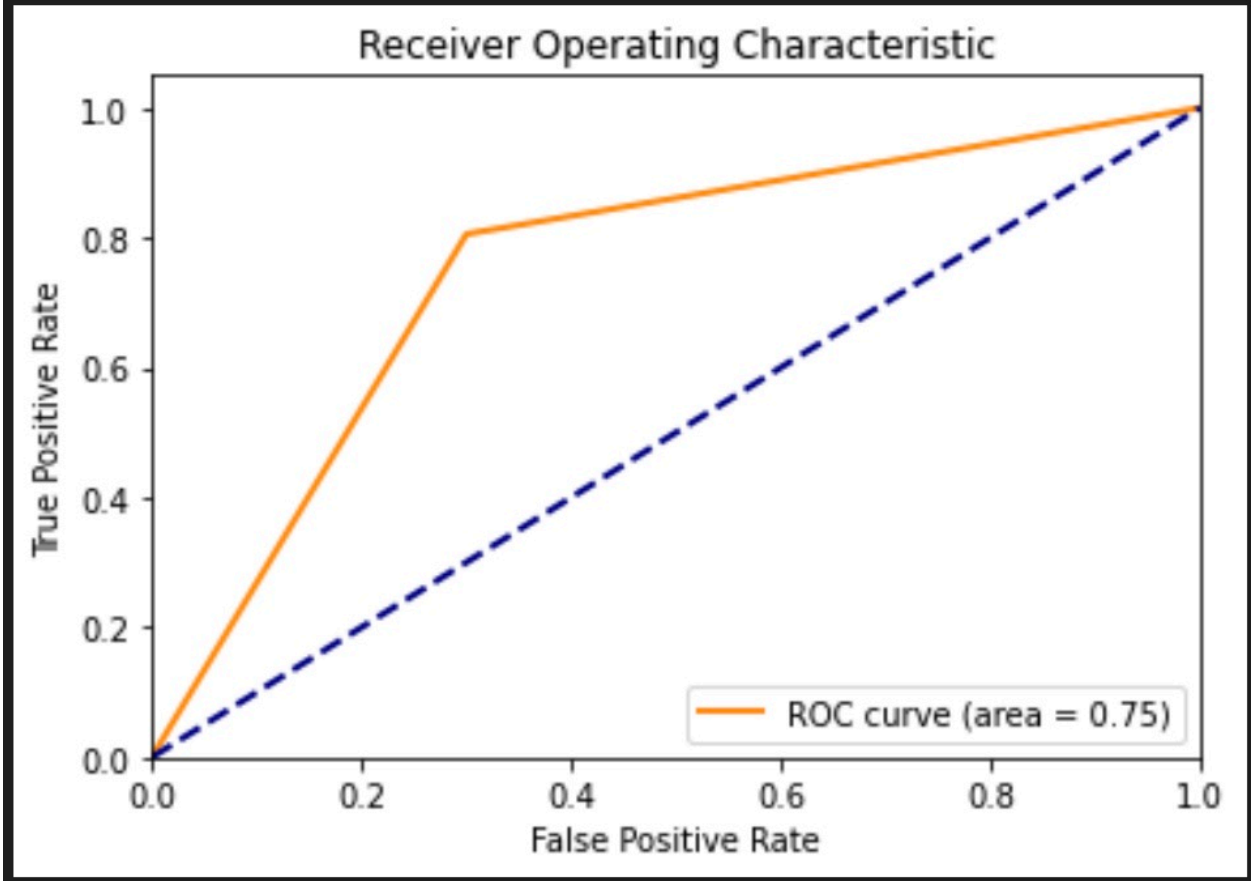
The result of the MLP Classifier without RFE

```
Accuracy: 0.753
Precision: 0.101
Recall: 0.639
F1-score: 0.175
```



The result of the MLP Classifier with RFE

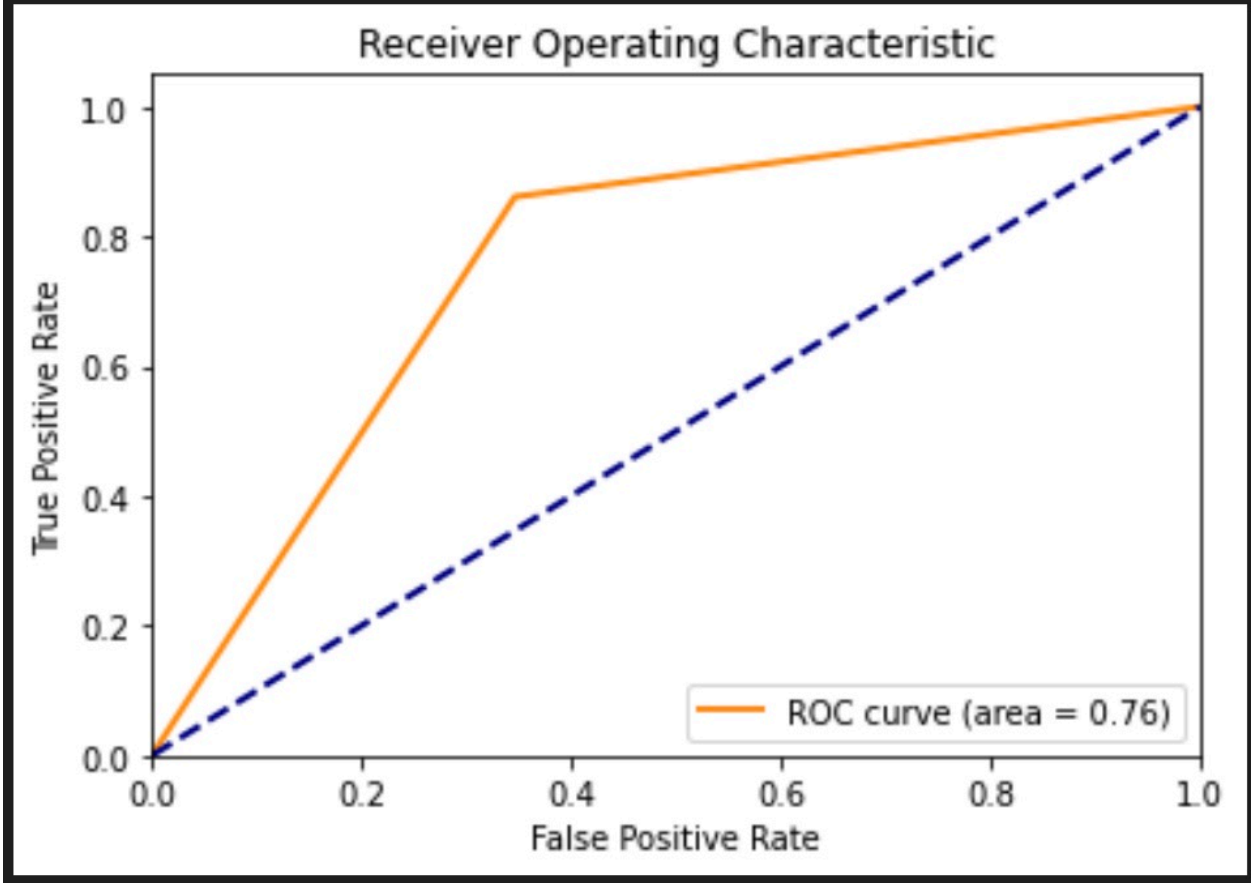
```
Accuracy: 0.704
Precision: 0.103
Recall: 0.806
F1-score: 0.182
```



The result of the MLP Classifier with PCA

```
Accuracy: 0.662
Precision: 0.096
Recall: 0.861
F1-score: 0.173
```

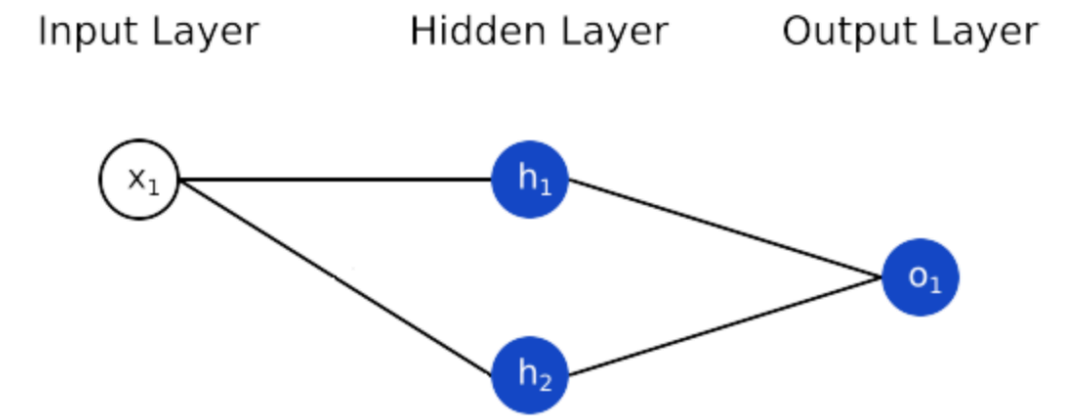
```
✓ # ROC AUC Metric ···
```



4.8. Keras

Keras is a neural network Application Programming Interface (API) for Python that is tightly integrated with TensorFlow, which is used to build machine learning models. Keras' models offer

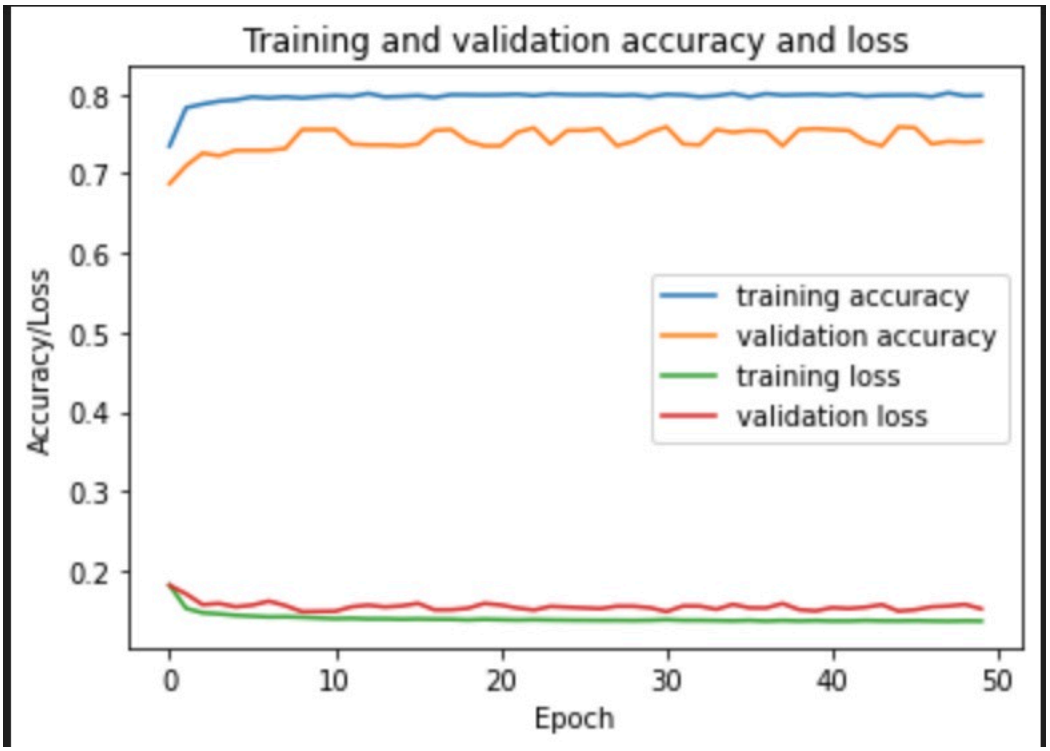
a simple, user-friendly way to define a neural network, which will then be built for you by TensorFlow.
Keras, on the other hand, is a high-level API that runs on top of TensorFlow. Keras simplifies the implementation of complex neural networks with its easy-to-use framework.
Models are the core entity you'll be working with when using Keras. The models are used to define TensorFlow neural networks by specifying the attributes, functions, and layers you want.
Keras offers several APIs you can use to define your neural network, including:
Sequential API, which lets you create a model layer by layer for most problems. It's straightforward (just a simple list of layers), but it's limited to single-input, single-output stacks of layers.
Functional API, which is a full-featured API that supports arbitrary model architectures. It's more flexible and complex than the sequential API.
Model Subclassing, which lets you implement everything from scratch. Suitable for research and highly complex use cases, but rarely used in practice.
The Sequential API is a framework for creating models based on instances of the sequential () class. The model has one input variable, a hidden layer with two neurons, and an output layer with one binary output. Additional layers can be created and added to the model.

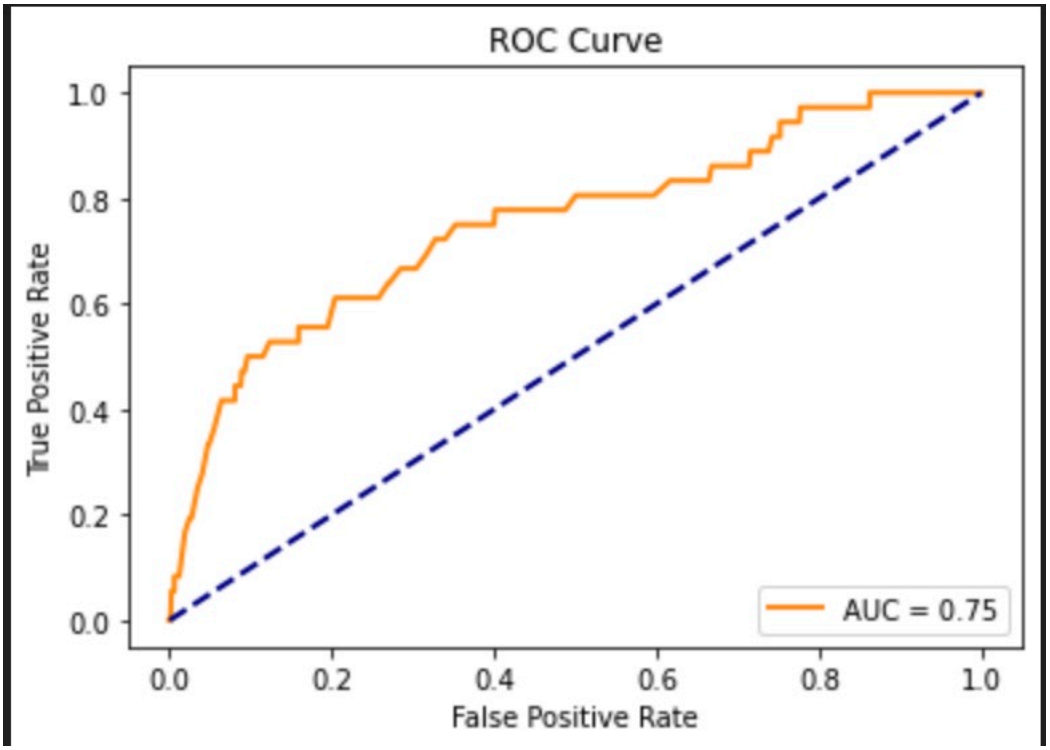


It allows us to develop models in a layer-by-layer fashion. However, it does not allow us to create models with numerous inputs or outputs. It works well for simple layer stacks with only one input and output tensor.

The result of the Keras

```
Accuracy: 0.741
Precision: 0.093
Recall: 0.611
F1-score: 0.162
```


Training and validation accuracy and loss
Accuracy/Loss
Epoch
training accuracy
validation accuracy
training loss
validation loss
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50


ROC Curve
True Positive Rate
False Positive Rate
AUC = 0.75
0.0
0.2
0.4
0.6
0.8
1.0

Optimization of the Model Selection

One always applies multiple relevant algorithms based on the problem and selects the best model based on the best performance metrics shown by the models. But this is not the end. One can increase the model performance using hyperparameters. Thus, finding the optimal hyperparameters would help us achieve the best-performing model. In this article, we will learn about Hyperparameters, Grid Search, Cross-Validation, GridSearchCV, and the tuning of Hyperparameters in Python. Hyperparameters for a model can be chosen using several techniques such as Random Search, Grid Search, Manual Search, Bayesian optimization, etc.

In this project, we are using GridSearchCV and Cross-Validation:

GridSearchCV is a technique for finding the optimal parameter values from a given set of parameters in a grid. It's essentially a cross-validation technique. The model as well as the parameters must be entered. After extracting the best parameter values, predictions are made.

GridSearchCV is the process of performing hyperparameter tuning to determine the optimal values for a given model.
GridSearchCV is also known as Grid Search cross-validation: an internal cross-validation technique is used to calculate the score for each combination of parameters on the grid.

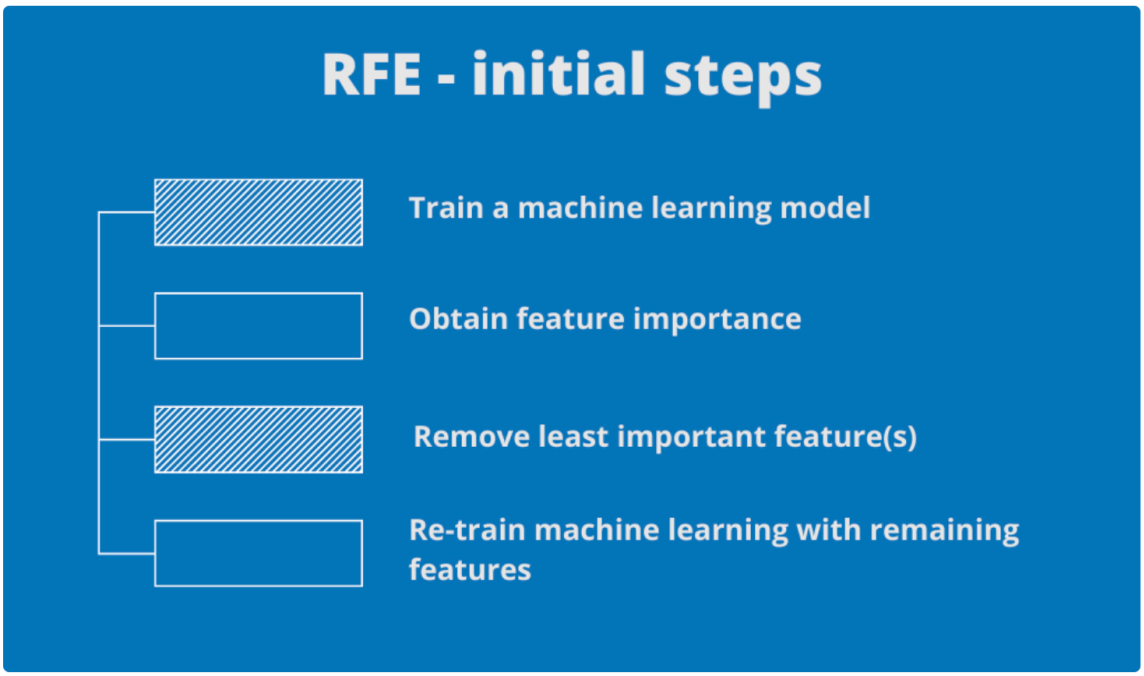
In GridSearchCV, along with Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training the model with data, we divide the data into two parts – train data and test data. In cross-validation, the process divides the train data further into two parts – the train data and the validation data.
The most popular type of Cross-validation is K-fold Cross-Validation. It is an iterative process that divides the train data into k partitions. Each iteration keeps one partition for testing and the remaining k-1 partitions for training the model. The next iteration will set the next partition as test data and the remaining k-1 as train data and so on. In each iteration, it will record the performance of the model and at the end give the average of all the performance.

PCA Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.
So, to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

Recursive feature elimination (RFE) is a technique that selects important features iteratively by training a model on all features, evaluating their importance, and then removing the least significant ones.
RFE involves the following steps:
Train an estimator using all features.
Obtain feature importance, typically from linear regression coefficients (coef_) or decision tree importance (feature_importances_) in Scikit-learn.
Remove the least important feature or group of features.
Train a new model using the remaining features and repeat the process.
This iterative approach helps to identify the most relevant features for the given machine learning task.
Steps for RFE

1. Train a machine-learning model
2. Derive feature importance.
3. Remove the least important feature(s)
4. Re-train the machine learning model on the remaining features

In this project, we use Scikit-Learn implementation.

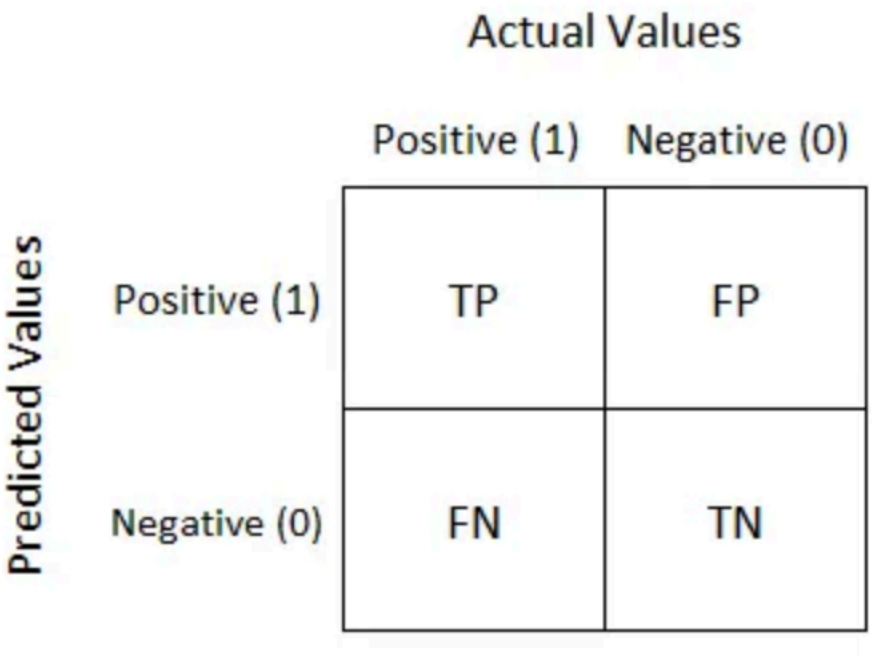


Evaluation Metrics

To assess its performance on test data, every Machine Learning model uses metrics. Metrics are like Loss Functions; except they are used to monitor test data. Accuracy, Binary Accuracy, Categorical Accuracy, and other forms of accuracy measures exist. Probabilistic measures such as binary cross-entropy, categorical cross-entropy, and others are also provided.
Different kinds of evaluation metrics available are as follows.
Classification Accuracy, Confusion Matrix, The area under Curve and ROC, F1 Score, Recall

Confusion Matrix

The performance of machine learning algorithms is typically evaluated by a confusion matrix as illustrated in Figure 1 (for a 2-class problem).



It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curves.
True Positive:
Interpretation: You predicted positive and it's true.
True Negative:
Interpretation: You predicted negative and it's true.
False Positive: (Type 1 Error)
Interpretation: You predicted positive and it's false.
False Negative: (Type 2 Error)

Interpretation: You predicted negative and it's false.
We describe predicted values as Positive and Negative and actual values as True and False.

Recall

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

The above equation can be explained by saying, from all the positive classes, how many we predicted correctly.
Recall should be high as possible.

Precision

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

The above equation can be explained by saying, from all the classes we have predicted as positive, how many are positive?
Precision should be high as possible.

Accuracy
From all the classes (positive and negative), how many of them we have predicted correctly.
Accuracy should be high as possible.

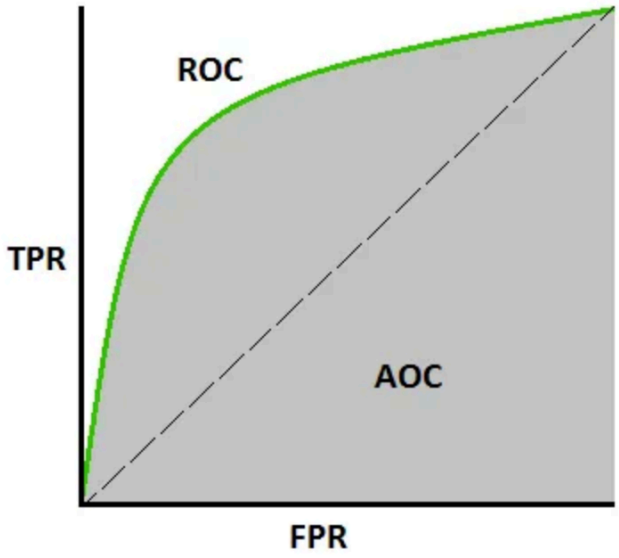
F-measure

$$F\text{ - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

It is difficult to compare two models with low precision and high recall or vice versa. So, to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time.

AUC and ROC

AUC - ROC curve is a performance measurement for classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.
The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.



An excellent model has AUC near to the 1 which means it has a good measure of separability. A poor model has an AUC near 0 which means it has the worst measure of separability. In fact, it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means the model has no class separation capacity whatsoever.

5. Results

In the stroke prediction project, the MLP model is the best choice due to its highest recall score. The recall score is essential in stroke prediction as it measures the model's ability to accurately identify patients at risk of having a stroke, minimizing false negatives. A high recall score is crucial in medical applications, as missed positive cases can lead to severe health consequences. The MLP model's superior recall performance ensures that more patients receive necessary preventive measures and timely treatment, prioritizing the detection of patients at risk of stroke.
In the stroke prediction project, the MLP model is the best choice due to its highest recall score. The recall score is essential in stroke prediction as it measures the model's ability to accurately identify patients at risk of having a stroke, minimizing false negatives. A high recall score is crucial in medical applications, as missed positive cases can lead to severe health consequences. The MLP model's superior recall performance ensures that more patients receive necessary preventive measures and timely treatment, prioritizing the detection of patients at risk of stroke.

Based on the evaluation of multiple machine learning models, we have observed the following scores for each model:

Model Name	Precision	F1 Score	ROC	Accuracy	Recall
Random Forest	0.17	0.23	0.76	0.89	0.39
XGBoost	0.16	0.23	0.78	0.88	0.42
Logistic Regression	0.11	0.18	0.80	0.77	0.64
KNN	0.20	0.05	0.74	0.95	0.03
SVM	0.10	0.17	0.72	0.75	0.61
MLP	0.10	0.18	0.75	0.70	0.81
Keras	0.09	0.16	0.75	0.74	0.61

6. Summary and Conclusions

Considering the problem statement of stroke prediction, recall is the most important evaluation metric. A high recall score ensures that the model can accurately identify a greater percentage of patients at risk of a stroke. Among all the tested models, the MLP model achieves the highest recall score of 0.81, making it the best choice for this problem.

Stroke is a life-threatening medical illness that should be treated as soon as possible to avoid further complications. The development of an ML model could aid in the early detection of stroke and the subsequent mitigation of its severe consequences. The effectiveness of several ML algorithms in properly predicting stroke based on several physiological variables is investigated in this study. The future scope of this study is that using a larger dataset and machine learning models, such as AdaBoost, SVM, and Bagging, the framework models may be enhanced. This will enhance the dependability of the framework and the framework's presentation. In exchange for just providing some basic information, the machine learning architecture may help the public in determining the likelihood of a stroke occurring in an adult patient. In an ideal world, it would help patients obtain early treatment for strokes and rebuild their lives after the event.

References

1. https://www.sciencedirect.com/science/article/pii/S235291481930019X
2. https://medium.com/geekculture/stroke-prediction-d26c15f9d1
3. https://towardsdatascience.com/5-data-science-projects-in-healthcare-that-will-get-you-hired-81003cadf2f3
4. Logistic regression in machine learning," [Online]. Google Scholar
5. G. Sailasya and G. L. A. Kumari, "Analyzing the performance of stroke prediction using ML classification algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, pp. 539–545, 2021 Google Scholar
6. "Documentation for random forest classification from sci-kit-learn," org. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7. https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148
8. https://towardsdatascience.com/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10
9. https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5
10. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
11. https://nicolo-albanese.medium.com/tips-on-principal-component-analysis-7116265971ad
12. https://medium.com/dataman-in-ai/sampling-techniques-for-extremely-imbalanced-data-part-i-under-sampling-a8dbc3d8d6d8
13. https://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html
14. https://towardsdatascience.com/powerful-feature-selection-with-recursive-feature-elimination-rfe-of-sklearn-23efb2cdb54e
15. https://www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=A%20confusion%20matrix%20represents%20the,by%20model%20as%20other%20class.