Breast Cancer Prediction Using Machine Learning

Rahul Kushwaha

W1629050

UNDER THE GUIDANCE OF

Dr. Behnam Dezfouli

Abstract

Breast cancer has become one of the most frequent and feared cancers across the globe, affecting roughly ten percent of all women internationally. Even though treatment exists in virtually all first world as well as some third-world countries, the primary issue lies in our inability to correctly identify its prevalence at the very early stages. Consequently, many cases slip by undetected. Fortunately, machine learning methods such as data mining and classification have opened new opportunities to effectively predict and identify cancers–including breast cancer–early. The major goal is to evaluate each algorithm's efficiency and effectiveness in terms of accuracy, sensitivity, specificity, and false positive rate to determine whether the data classification was right.

Contents

Overview

This undertaking is associated with COEN-240 Machine Learning. The present paper begins by providing a brief overview of the project and outlining its goals. The specified dataset will then be established and prepared. The findings will be discussed. The report will conclude with some last observations.

2.1 Introduction

About 30% of all women will have breast cancer at some point in their lives, making it the most prevalent kind of cancer in women. The fatality rate from breast cancer has decreased by 39% since 1989 because to significant advancements in early detection. From microscopic pictures, one may see numerical properties such naked nucleoli, mitoses, and plain chromatin. Later, results from the FNA are examined in conjunction with different imaging data to determine the likelihood that the patient has a malignant breast cancer tumor. All things considered; early prognosis is still a crucial part of the follow-up procedure. Reducing the amount of False positive and false negative decisions can be accomplished via data mining techniques or categorization. As they are most frequently employed in various types of data analysis, supervised learning models (algorithms that learn from labelled data) will be the main models utilized and evaluated. In this study, the Wisconsin Breast Cancer Dataset was subjected to many classification models. The findings are then evaluated using a variety of performance criteria to compare the algorithms and determine which model is most appropriate for cancer prediction.

2.2 Aim of the project

This study aims to develop machine learning models that can determine if a breast cancer cell is benign or malignant. To find patterns in the dataset and produce a more reliable analysis, data will be modified. As previously stated, the resulting accuracy, sensitivity, and specificity, among other characteristics, will be taken into consideration when choosing the best model. These measures will be defined further on. To identify and categorize the characteristics in a picture of a cancer cell nucleus, we can apply machine learning. The classification of a sample as benign ("B") or malignant ("M") would be useful.

The machine learning models we'll be using in this study aim to produce a classifier with a high degree of accuracy and a low incidence of false-negatives (high sensitivity) and false positives (high specificity).

Methods and Analysis

3.1 Data Preprocessing and Exploratory Data Analysis

3.1.1 Dataset Acquisition and Description

Dr. William H. Wolberg, a physician at the University of Wisconsin Hospital in Madison, Wisconsin, created the Breast Cancer Wisconsin Dataset, which is used in this work (https://www.kaggle.com/roustekbio/breastcancercsv). This information was compiled by the University of Wisconsin in 1992 using the findings of 699 patient biopsies.

The characteristics of the cell nuclei are described by the features in the dataset. The following is included in this:

1) Sample code number id number
2) Clump Thickness 1 - 10
3) Uniformity of Cell Size 1 - 10
4) Uniformity of Cell Shape 1 - 10
5) Marginal Adhesion 1 - 10
6) Single Epithelial Cell Size 1 - 10
7) Bare Nuclei 1 - 10
8) Bland Chromatin 1 - 10
9) 9) Normal Nucleoli 1 – 10
10) 10) Mitoses 1-10

3.1.2 Data Preprocessing

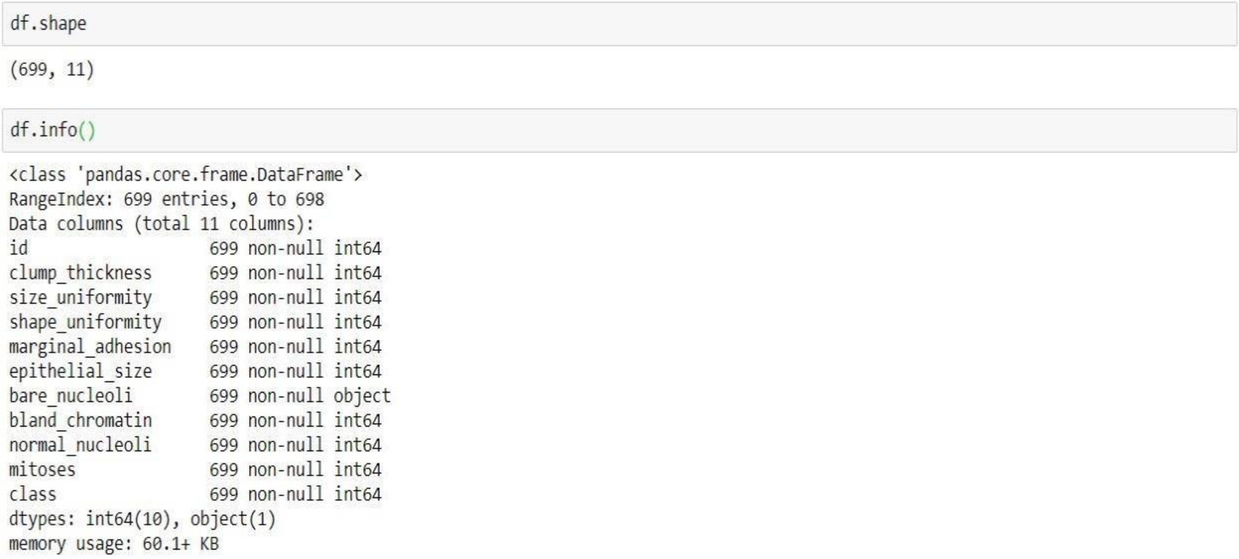
We discovered that the supplied dataset comprises 699 observations (samples) with 11 variables (features).

Note: Because the feature/variable 'id' does not contribute to the overall machine learning (i.e., it does not offer any meaningful insight into the nature of differentiated cells), the column has been omitted.

The initial collected data included a column, "bare nucleoli," that was made up of 16 question marks ('?'s) to denote 16 missing values/samples. These 16 samples were all removed from the dataset. Then, the remaining 683 samples were handled as a different dataset. The "K Nearest Neighbors" technique was then used for this subset as part of the machine learning. Finally, the values for the missing features were predicted using the trained machine.

Since a trained machine considers all the features corresponding to all the samples before predicting the output, using one to predict the missing values is preferable to simply replacing them with mean, median, and mode values of the bare nucleoli. This anticipated result will consider the complete dataset, and hence the distribution, rather than just the values contained in one single feature (as would be the case of mean, medium, and mode).

The number of neighbors was set to 16 based on the minimum Mean Absolute Error, and the distance/similarity function was set to 'Minkowski' (to account for any outliers in the dataset). The distance function and the correct number of neighbors will be addressed in depth later.

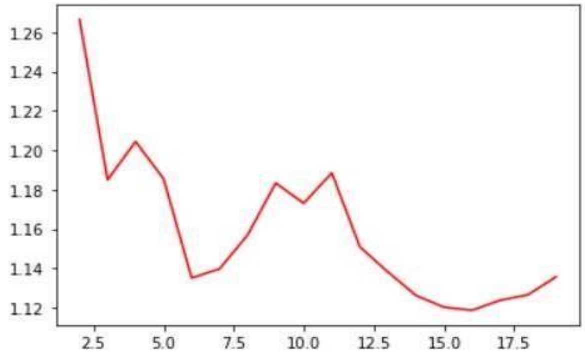


```
df.shape
```

```
(699, 11)
```

```
df.info()
```

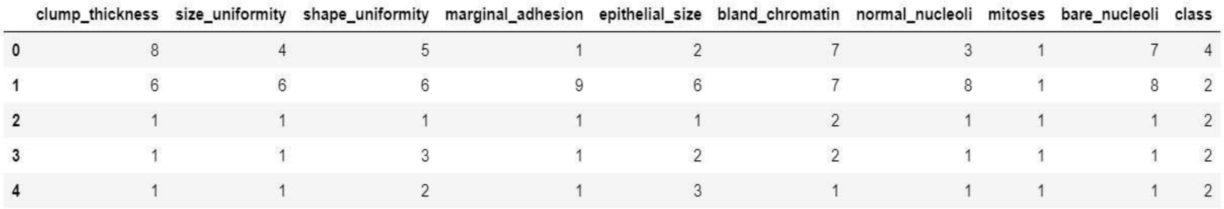
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
id                   699 non-null int64
clump_thickness      699 non-null int64
size_uniformity      699 non-null int64
shape_uniformity     699 non-null int64
marginal_adhesion    699 non-null int64
epithelial_size      699 non-null int64
bare_nucleoli        699 non-null object
bland_chromatin      699 non-null int64
normal_nucleoli      699 non-null int64
mitoses              699 non-null int64
class                699 non-null int64
dtypes: int64(10), object(1)
memory usage: 60.1+ KB
```



```
Best k parameter is  16
```

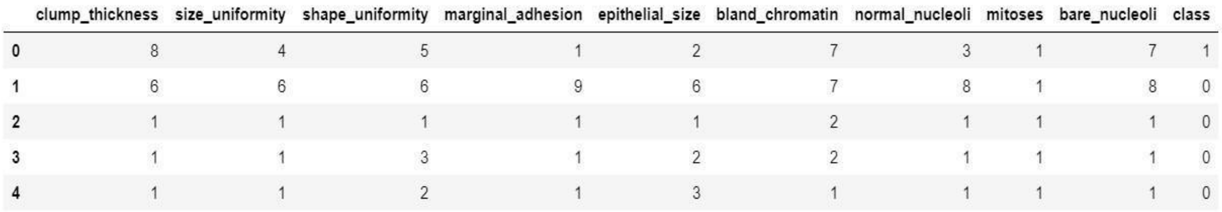
The anticipated values were then rounded up and added to the extracted samples. These extracted samples were then combined with the remaining 683 samples to produce the old dataset with the new anticipated values.

The dataset's column 'class' has two distinct classes (binary classification): '2' and '4', where 2 denotes a benign tumor (i.e., non-cancerous) and 4 represents a malignant tumor (i.e., cancerous).



	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bland_chromatin	normal_nucleoli	mitoses	bare_nucleoli	class
0	8	4	5	1	2	7	3	1	7	4
1	6	6	6	9	6	7	8	1	8	2
2	1	1	1	1	1	2	1	1	1	2
3	1	1	3	1	2	2	1	1	1	2
4	1	1	2	1	3	1	1	1	1	2

These two distinct classes were then encoded to binary numbers (0 and 1), where the former denotes a benign tumor, and the latter denotes a malignant one.

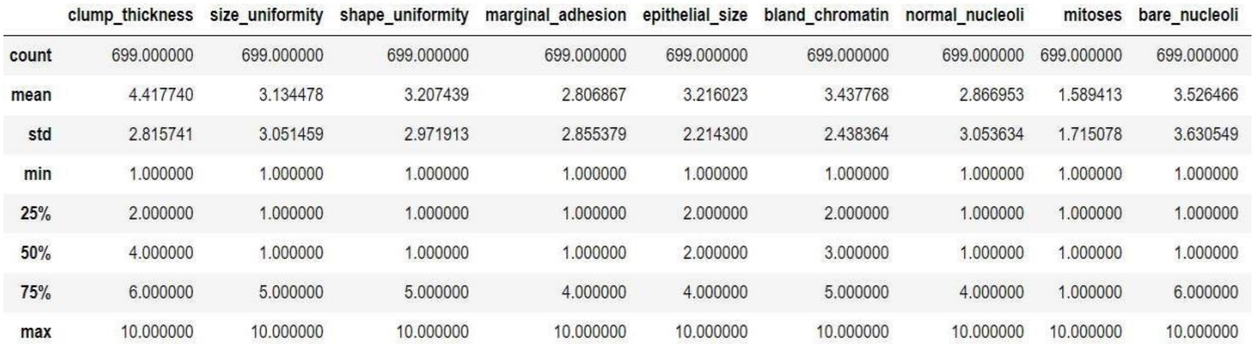


	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bland_chromatin	normal_nucleoli	mitoses	bare_nucleoli	class
0	8	4	5	1	2	7	3	1	7	1
1	6	6	6	9	6	7	8	1	8	0
2	1	1	1	1	1	2	1	1	1	0
3	1	1	3	1	2	2	1	1	1	0
4	1	1	2	1	3	1	1	1	1	0

3.1.3 Exploratory Data Analysis

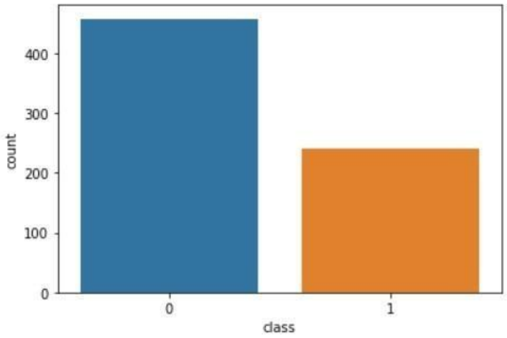
Exploratory data analysis is a crucial procedure that entails doing early investigations on data in order to find patterns, identify anomalies, test hypotheses, and validate assumptions with the use of summary statistics and graphical representations. Understanding the data first and attempting to glean as many insights from it as possible is a smart strategy. Understanding the data thoroughly before attempting to draw as many conclusions as possible from it is a good approach. Before going hands-on with the data, EDA is all about making sense of it.

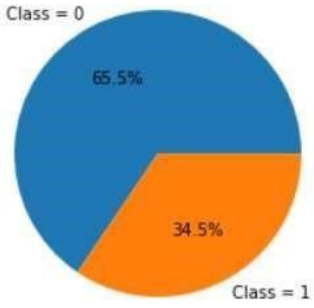
For each feature in the dataset, I first determined the mean, variance, standard deviation, maximum, and minimum values.



	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bland_chromatin	normal_nucleoli	mitoses	bare_nucleoli
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413	3.526466
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078	3.630549
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000	6.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

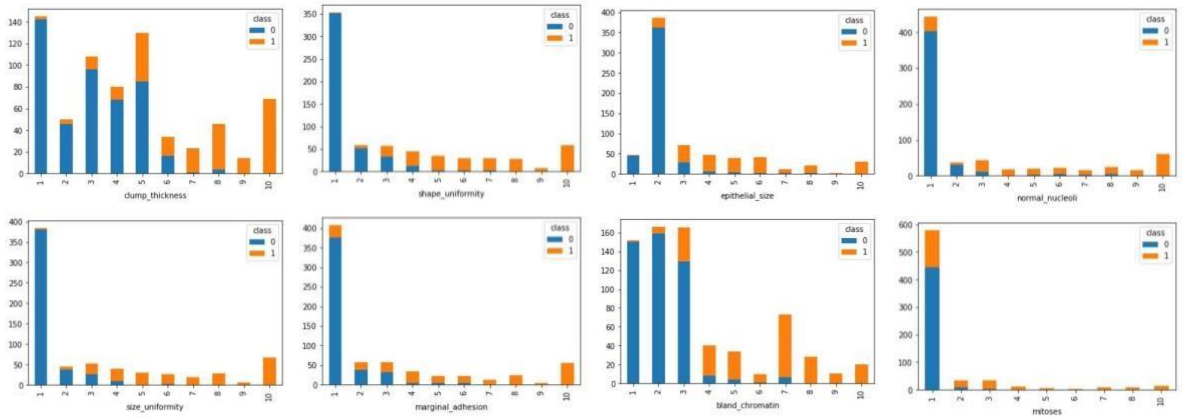
In order to determine if the provided data is balanced or not, I also plotted the count plot and pie chart for the feature "class." Data is seen as balanced if the distribution of the two classes is around 50% for each, while the distribution of the two classes is regarded as imbalanced or skewed if it is roughly 90-10.





The binary classifications included in the dataset may be quantitatively compared using both the count plot and the pie chart. The remaining 35.5% of the samples are categorized as malignant, leaving 65.5% of the samples as benign. The distribution is believed to be balanced because it is 65-35 (slightly unbalanced), thus no up-sampling, downsampling, or cost-sensitive learning is done. The data are assumed to be balanced for the purposes of the rest of the analysis in this report.

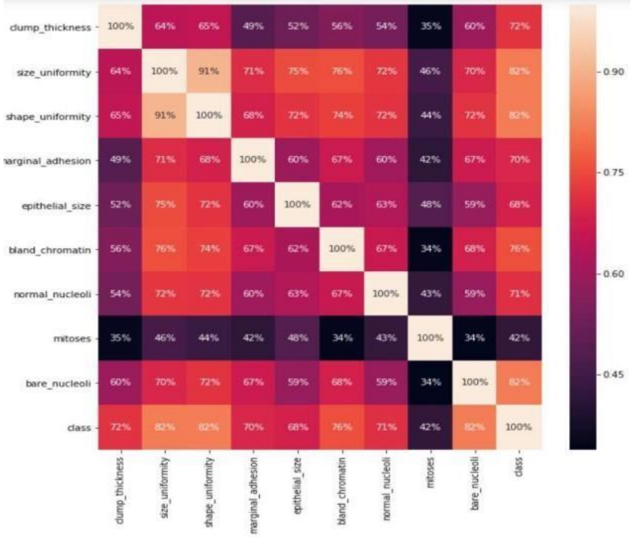
Cross tab plots were created for each characteristic in the supplied dataset in order to conduct a thorough quantitative analysis.



According to the value (1–10) of a certain trait, these crosstab plots show the proportion of tumors that are categorized as benign or malignant. Every graph makes it apparent that a cell is labeled as malignant if a characteristic has a value higher than a certain threshold. Additionally, different thresholds for various feature values can be seen.

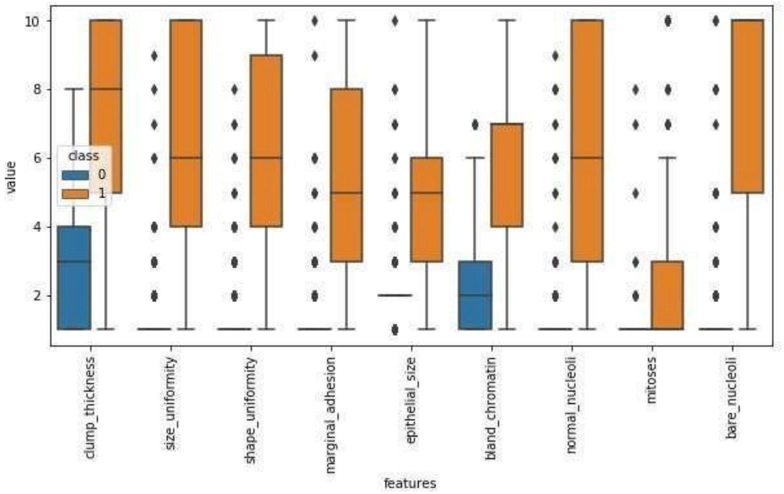
When it comes to categorizing a tumor, some characteristics are more sensitive than others. For instance, a cell is often categorized as malignant if the value of shape uniformity/size uniformity is reported to be more than 5. When the number for mitoses hits 3, the tumor is more likely to be malignant, however when the value for clump thickness reaches 8, there is still a small possibility that the cell won't be labeled as malignant. Different characteristics affect categorization in different ways, and this effect can be connected to correlation. Each feature's relationships to other features and to the class to which each sample is being mapped are unique.

A statistical technique used to assess the degree of link between two quantitative variables is correlation analysis. A high correlation indicates a significant association between two or more variables, whereas a low correlation indicates little relationship between the variables. In other words, it involves analyzing the strength of that association using the statistical information that is now accessible. A correlation matrix may therefore be used to assess how much a trait can influence the other or the class. A heat map has been used to depict the same.



Size uniformity, shape uniformity, and naked nucleoli are properties that are significantly connected to class, according to the heat map above, whereas mitoses is the trait that is least correlated to class.

Finding any outliers in the dataset was another issue that needed to be addressed. This was accomplished using the box plot shown below -



There were a few outliers in the dataset, according to the box plot. However, because the dataset was too tiny and each feature only had a value between 1 and 10, no operations were made to account for these outliers in the dataset.

3.2 Modelling

3.2.1 Model creation

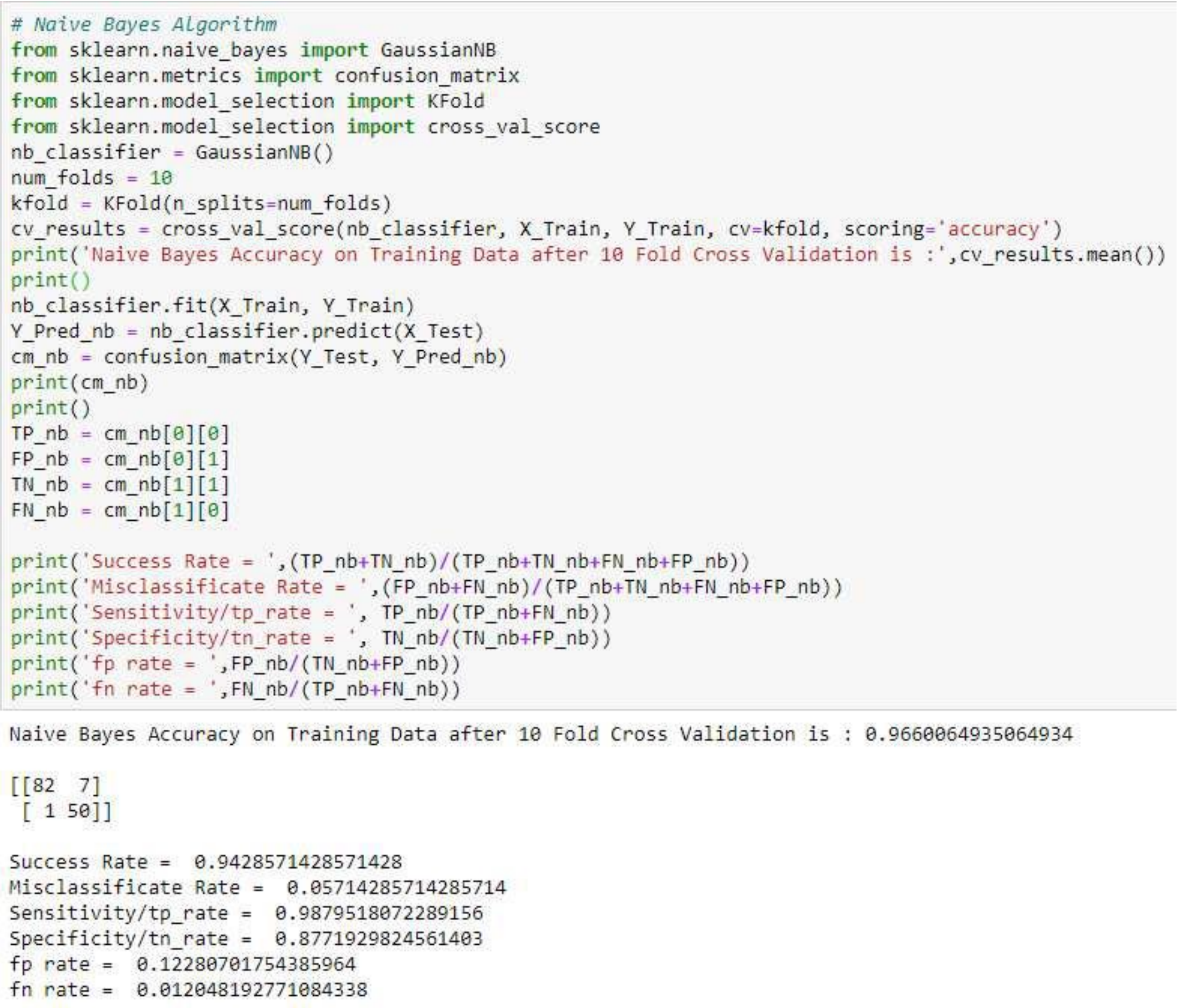
In machine learning, data are often divided into training and testing sets (and occasionally three sets: train, validate, and test), with our model being fitted to the training set before making predictions about the test set. The real dataset that we utilize to train the model includes a training dataset. This data is seen by and used to train the model. Contrarily, test data is a sample of data that is used to conduct an objective study of a final model fit on the training dataset. I divided the dataset into training and test halves using an 80%-20% ratio for this report (the first 559 instances for training while the next 140 instances for testing the model).

3.2.2 Naïve Bayes Model

Naive Bayes classifiers are a group of classification algorithms based on Bayes' Theorem, which calculates the probability of a future event using previous information. The Naive Bayes classifier assumes that input variables are independent of each other and that all options contribute separately to the likelihood of the target variable. This assumption is why it is referred to as "naive." However, in real-world datasets, the feature variables often have dependencies, which is a drawback of the Naive Bayes classifier. Despite this drawback, Naive Bayes classifiers perform well on large datasets and often outperform more complex classifiers. The formula for Naive Bayes theorem is as follows: [insert formula here]. In this formula, P(C|A) represents the posterior probability, which is the probability of a hypothesis (C) being true given some evidence (A). P(C) is the prior probability, the probability of the hypothesis being true. P(A) is the probability of the evidence, regardless of the hypothesis. P(A|C) is the probability of the evidence when the hypothesis is true.

Naive Bayes algorithm is commonly used for binary and multi-category classification tasks and has the advantage of being trainable on small datasets. This flexibility is particularly beneficial when working with limited data. Overall, Naive Bayes classifiers provide a useful approach for classification tasks, leveraging Bayes' Theorem to estimate probabilities and make predictions. While the assumption of feature independence may not always hold true in real-world scenarios, Naive Bayes classifiers still prove valuable in various applications.

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$



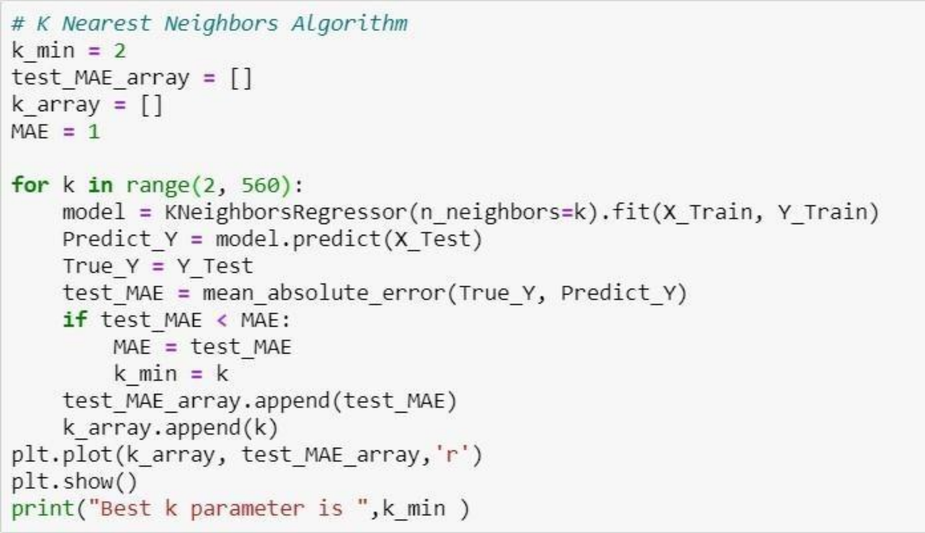
```
# Naive Bayes Algorithm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
nb_classifier = GaussianNB()
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(nb_classifier, X_Train, Y_Train, cv=kfold, scoring='accuracy')
print('Naive Bayes Accuracy on Training Data after 10 Fold Cross Validation is :',cv_results.mean())
print()
nb_classifier.fit(X_Train, Y_Train)
Y_Pred_nb = nb_classifier.predict(X_Test)
cm_nb = confusion_matrix(Y_Test, Y_Pred_nb)
print(cm_nb)
print()
TP_nb = cm_nb[0][0]
FP_nb = cm_nb[0][1]
TN_nb = cm_nb[1][1]
FN_nb = cm_nb[1][0]

print('Success Rate = ',(TP_nb+TN_nb)/(TP_nb+TN_nb+FN_nb+FP_nb))
print('Misclassificate Rate = ',(FP_nb+FN_nb)/(TP_nb+TN_nb+FN_nb+FP_nb))
print('Sensitivity/tp_rate = ', TP_nb/(TP_nb+FN_nb))
print('Specificity/tn_rate = ', TN_nb/(TN_nb+FP_nb))
print('fp rate = ',FP_nb/(TN_nb+FP_nb))
print('fn rate = ',FN_nb/(TP_nb+FN_nb))
```

```
Naive Bayes Accuracy on Training Data after 10 Fold Cross Validation is : 0.9660064935064934

[[82  7]
 [ 1 50]]

Success Rate =  0.9428571428571428
Misclassificate Rate =  0.05714285714285714
Sensitivity/tp_rate =  0.9879518072289156
Specificity/tn_rate =  0.8771929824561403
fp rate =  0.12280701754385964
fn rate =  0.012048192771084338
```

3.2.3 K-Nearest Neighbors Model

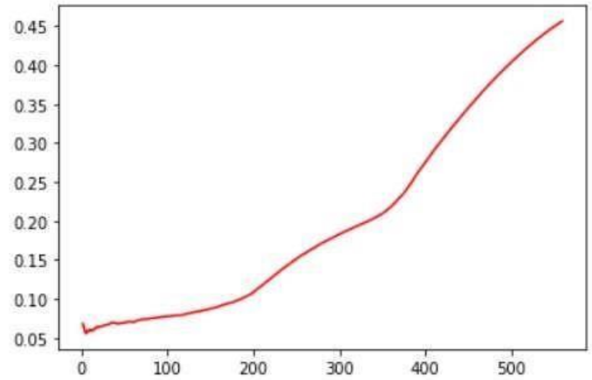
One of the simplest machine learning techniques is the K-nearest neighbor's algorithm. It has just provided evidence for the idea that items that are "near" every alternative might also share traits. Therefore, if it can identify the distinctive qualities of one of the items, it can also anticipate those of its closest neighbor. KNN is an improvisational version of the closest neighbor method. It is primarily predicated on the idea that any new instance will be classified by the majority vote of its "k" neighbors, where "k" is a positive number, occasionally with a small amount of variation. One of the most popular simple and straightforward data processing methods is KNN. Due to the requirement that the coaching examples be in the memory during run-time, it is known as Memory-Based Classification. After dealing with continuous attributes, a similarity/distance function is used to determine how to distinguish between the attributes. The distance/similarity function, which is chosen based on the applications and kind of data, is the main problem with the KNN method. The careful selection of a suitable distance function is an important step in the use of KNN.

As a result, the categorization of the test data point's closest neighbors is dependent upon it. The number of neighbors in the KNN model for this dataset is calculated with the aid of MAE (Mean Absolute Error). When a test data set is used as the input, the model's mean absolute error is defined as the average of the absolute values of each prediction error over all instances. The prediction error is the discrepancy between the actual value and the value that was expected in that particular situation. The machine is trained for various numbers of neighbors, and the mean absolute error is determined and compared. The number of neighbors with the lowest observed mean absolute error is utilized to train the model.



```
# K Nearest Neighbors Algorithm
k_min = 2
test_MAE_array = []
k_array = []
MAE = 1

for k in range(2, 560):
    model = KNeighborsRegressor(n_neighbors=k).fit(X_Train, Y_Train)
    Predict_Y = model.predict(X_Test)
    True_Y = Y_Test
    test_MAE = mean_absolute_error(True_Y, Predict_Y)
    if test_MAE < MAE:
        MAE = test_MAE
        k_min = k
    test_MAE_array.append(test_MAE)
    k_array.append(k)
plt.plot(k_array, test_MAE_array,'r')
plt.show()
print("Best k parameter is ",k_min )
```

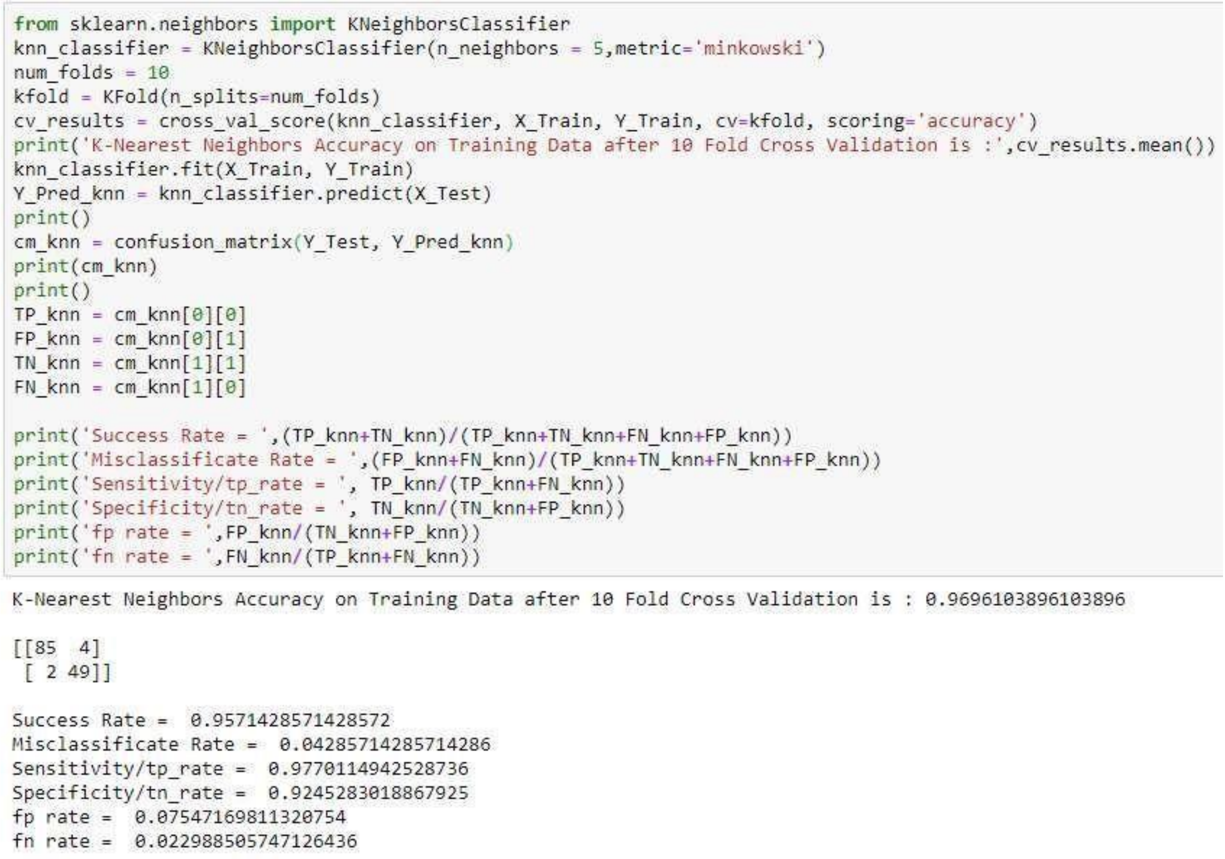
The above code snippet displays the mean absolute error value that is achieved each time the machine is trained with a varied number of neighbors. There are anywhere from 2 to 560 neighbors (2 being the value just greater than 1 and 560 being the maximum number of samples present in training data). Here is the graph that was created:



```
Best k parameter is  5
```

The graph unmistakably demonstrates that the least MAE is seen for several neighbors of 5. To train the model, we thus employ 5 neighbors. When dealing with outlier-filled data, Minkowski distance is favored because it is more resilient than Euclidean distance, which is susceptible to outliers.

Because there are multiple outliers, as can be seen in the scatter plots, the similarity/distance function employed in this situation is the Minkowski Distance.



```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5,metric='minkowski')
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(knn_classifier, X_Train, Y_Train, cv=kfold, scoring='accuracy')
print('K-Nearest Neighbors Accuracy on Training Data after 10 Fold Cross Validation is :',cv_results.mean())
knn_classifier.fit(X_Train, Y_Train)
Y_Pred_knn = knn_classifier.predict(X_Test)
print()
cm_knn = confusion_matrix(Y_Test, Y_Pred_knn)
print(cm_knn)
print()
TP_knn = cm_knn[0][0]
FP_knn = cm_knn[0][1]
TN_knn = cm_knn[1][1]
FN_knn = cm_knn[1][0]

print('Success Rate = ',(TP_knn+TN_knn)/(TP_knn+TN_knn+FN_knn+FP_knn))
print('Misclassificate Rate = ',(FP_knn+FN_knn)/(TP_knn+TN_knn+FN_knn+FP_knn))
print('Sensitivity/tp_rate = ', TP_knn/(TP_knn+FN_knn))
print('Specificity/tn_rate = ', TN_knn/(TN_knn+FP_knn))
print('fp rate = ',FP_knn/(TN_knn+FP_knn))
print('fn rate = ',FN_knn/(TP_knn+FN_knn))
```

```
K-Nearest Neighbors Accuracy on Training Data after 10 Fold Cross Validation is : 0.9696103896103896

[[85  4]
 [ 2 49]]

Success Rate =  0.9571428571428572
Misclassificate Rate =  0.04285714285714286
Sensitivity/tp_rate =  0.9770114942528736
Specificity/tn_rate =  0.9245283018867925
fp rate =  0.07547169811320754
fn rate =  0.022988505747126436
```

3.2.4 Logistic Regression Model

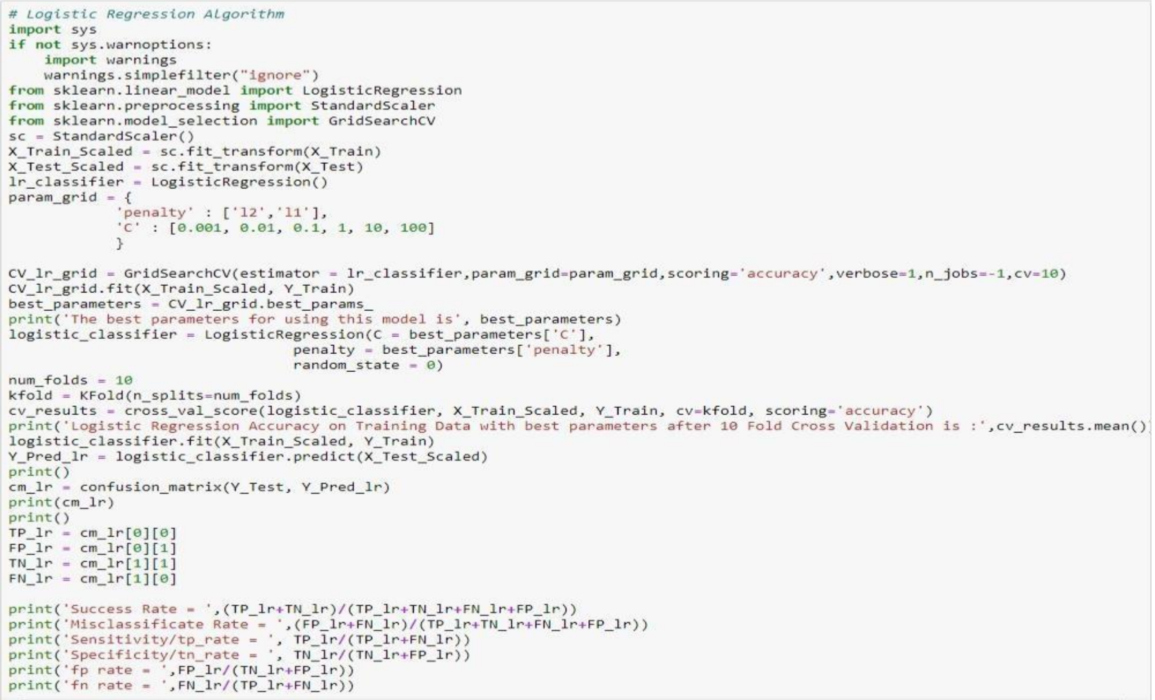
A supervised machine learning method used in classification tasks is logistic regression (for predictions based on training data). Like linear regression, logistic regression employs an equation, but its result is a categorical variable as opposed to other regression models' values. The independent variables can be used to forecast binary outcomes. Given that the logistic function is the primary methodology employed, this process is known as logistic regression. The independent variables, which make up a linear equation, can predict the outcome. The anticipated result is not constrained; it may range from a negative infinity to a positive infinity. However, a class variable is the output that is needed (i.e., yes or no, 1 or 0). The relationship between the independent variables, the input, and the variable quantity, the output, is measured via logistic regression.

To maximize testing accuracy, numerous parameters in logistic regression can be modified or improved. The following variables were utilized in this report to ensure the highest level of testing accuracy: –

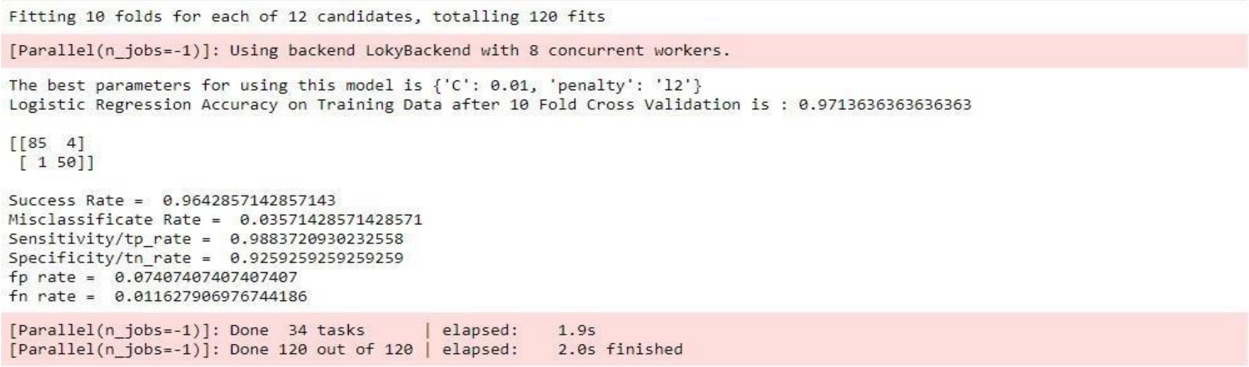
- Parameter C – The inverse of regularization strength is a regularization parameter (or lambda). The trade-off between enabling the model to become as complicated as it likes and attempting to keep it simple is controlled by lambda (). By giving large values to the weights for each parameter, the model will have the potential to become more complicated (over fit), for instance, if is very low or 0. On the other hand, if we raise the value of, the model will inevitably underfit since it will be too straightforward. For big values of C, we low the power of regularization which implies the model is allowed to increase its complexity, and therefore, over fit the data. In order to achieve the best fit, parameter C has to be tuned/optimized.
- Penalty – When used with logistic regression, the attribute "Penalty" specifies the norm that is applied when penalizing. In this study, two different kinds of penalties—l1 and l2—were used. L1 stands for Lasso Regression, while L2 stands for Ridge Regression. The main distinction between these techniques is that Lasso removes some

features entirely by shrinking the coefficient of the less significant features to zero, making it a good tool for feature selection when there are a lot of features.

GridSearchCV was used to improve the above- mentioned parameters, and the best results were achieved. The model's testing accuracy was then determined using the optimal parameter settings.



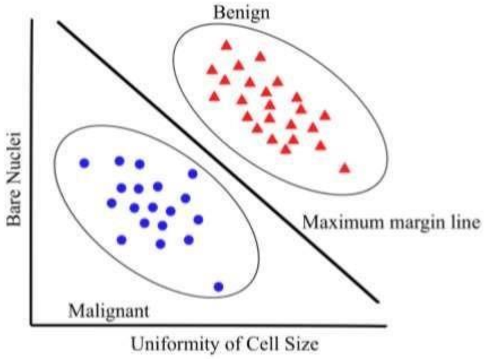
```
# Logistic Regression Algorithm
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
sc = StandardScaler()
X_Train_Scaled = sc.fit_transform(X_Train)
X_Test_Scaled = sc.fit_transform(X_Test)
lr_classifier = LogisticRegression()
param_grid = {
            'penalty' : ['l2','l1'],
            'C' : [0.001, 0.01, 0.1, 1, 10, 100]
            }

CV_lr_grid = GridSearchCV(estimator = lr_classifier,param_grid=param_grid,scoring='accuracy',verbose=1,n_jobs=-1,cv=10)
CV_lr_grid.fit(X_Train_Scaled, Y_Train)
best_parameters = CV_lr_grid.best_params_
print('The best parameters for using this model is', best_parameters)
logistic_classifier = LogisticRegression(C = best_parameters['C'],
                                penalty = best_parameters['penalty'],
                                random_state = 0)
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(logistic_classifier, X_Train_Scaled, Y_Train, cv=kfold, scoring='accuracy')
print('Logistic Regression Accuracy on Training Data with best parameters after 10 Fold Cross Validation is :',cv_results.mean()
logistic_classifier.fit(X_Train_Scaled, Y_Train)
Y_Pred_lr = logistic_classifier.predict(X_Test_Scaled)
print()
cm_lr = confusion_matrix(Y_Test, Y_Pred_lr)
print(cm_lr)
print()
TP_lr = cm_lr[0][0]
FP_lr = cm_lr[0][1]
TN_lr = cm_lr[1][1]
FN_lr = cm_lr[1][0]

print('Success Rate = ',(TP_lr+TN_lr)/(TP_lr+TN_lr+FN_lr+FP_lr))
print('Misclassificate Rate = ',(FP_lr+FN_lr)/(TP_lr+TN_lr+FN_lr+FP_lr))
print('Sensitivity/tp_rate = ', TP_lr/(TP_lr+FN_lr))
print('Specificity/tn_rate = ', TN_lr/(TN_lr+FP_lr))
print('fp rate = ',FP_lr/(TN_lr+FP_lr))
print('fn rate = ',FN_lr/(TP_lr+FN_lr))
```



```
Fitting 10 folds for each of 12 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

The best parameters for using this model is {'C': 0.01, 'penalty': 'l2'}
Logistic Regression Accuracy on Training Data after 10 Fold Cross Validation is : 0.9713636363636363

[[85  4]
 [ 1 50]]

Success Rate =  0.9642857142857143
Misclassificate Rate =  0.03571428571428571
Sensitivity/tp_rate =  0.9883720930232558
Specificity/tn_rate =  0.9259259259259259
fp rate =  0.07407407407407407
fn rate =  0.011627906976744186

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    2.0s finished
```

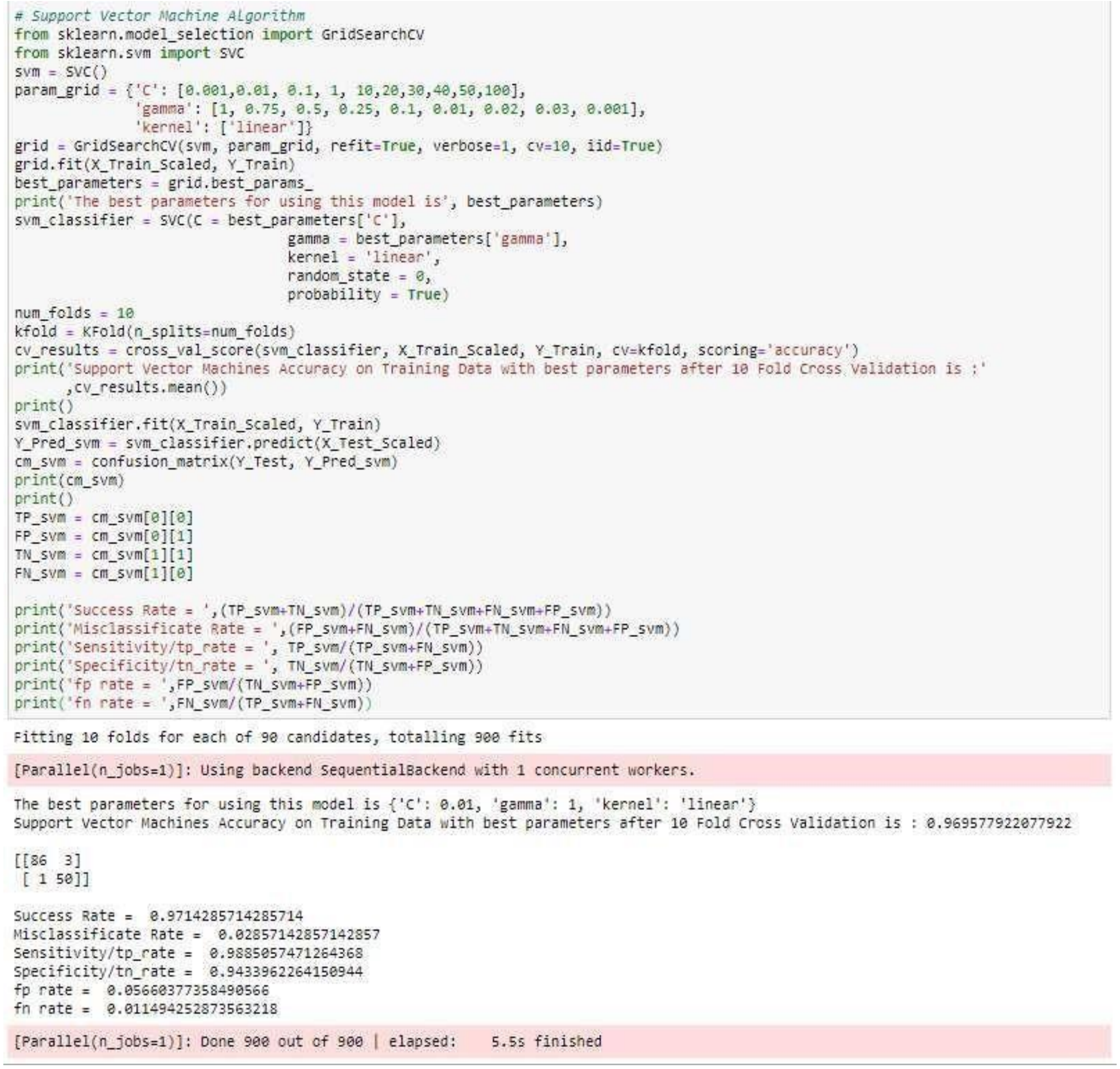
3.2.5 Support Vector Machines Model

A supervised machine learning algorithmic method called the Support Vector Machine (SVM) may be utilized to solve any classification or regression problems. But it's mostly applied to categorization problems. According to this algorithmic rule, each data point is represented as a point in n-dimensional space, where n is the number of features each feature possesses and each feature's value corresponds to a certain location. Then, classification is carried out by locating the hyperplane that separates the two classes. Researchers frequently have a tendency to plot each piece of knowledge as an extent in an n-dimensional space, with each feature's value being determined by a particular coordinate. Next, do classification by locating the hyper-plane that clearly distinguishes the two groups. Although it is a non-probabilistic binary linear classifier, it is frequently modified such that it can also do non-linear and probabilistic classification, making it a flexible algorithmic program. Unsupervised learning is necessary when data are unlabeled since supervised learning cannot be done because there is no natural grouping of the data. New data is then mapped to these created groups. Finding the separating hyperplane that maximizes the margin of the training data is the main goal of SVM.



There are several factors whose tweaking or optimization can significantly impact the testing accuracy, even in the case of Support Vector Machines. The following are a few Support Vector Machines parameters that have been adjusted or modified for this project: –

- Parameter C – Regularization Parameter – Same as defined for Logistic Regression.
- Sigma - The RBF kernel's gamma parameter controls the scope of a single training instance. When gamma is high, the SVM decision boundary will only consider points that are close to the boundary, essentially ignoring points that are farther away. A decision boundary will take points farther from it into consideration when gamma is low, in contrast. Because of this, decision boundaries with high gamma values tend to be highly flexible, while those with low gamma values tend to be more linear.
- Kernel - A collection of mathematical operations known as the kernel are used by SVM algorithms. Data is inputted into the kernel, which then transforms it into the desired form. Different kernel functions are used by various SVM algorithms. There are several forms of these functions. For instance, linear, nonlinear, polynomial, sigmoid, and radial basis function (RBF). The inner product between two locations in an appropriate feature space is returned by the kernel functions. Thus, even in very high-dimensional areas, with low computing expense, by establishing a notion of similarity. When the data can be divided using a single line and is linearly separable, the linear kernel is utilized. It is one of the most often utilized kernels. Only the "Linear" kernel has been utilized in this project.



```
# Support Vector Machine Algorithm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
svm = SVC()
param_grid = {'C': [0.001,0.01, 0.1, 1, 10,20,30,40,50,100],
              'gamma': [1, 0.75, 0.5, 0.25, 0.1, 0.01, 0.02, 0.03, 0.001],
              'kernel': ['linear']}
grid = GridSearchCV(svm, param_grid, refit=True, verbose=1, cv=10, iid=True)
grid.fit(X_Train_Scaled, Y_Train)
best_parameters = grid.best_params_
print('The best parameters for using this model is', best_parameters)
svm_classifier = SVC(C = best_parameters['C'],
                                gamma = best_parameters['gamma'],
                                kernel = 'linear',
                                random_state = 0,
                                probability = True)
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(svm_classifier, X_Train_Scaled, Y_Train, cv=kfold, scoring='accuracy')
print('Support Vector Machines Accuracy on Training Data with best parameters after 10 Fold Cross Validation is :'
      ,cv_results.mean())
print()
svm_classifier.fit(X_Train_Scaled, Y_Train)
Y_Pred_svm = svm_classifier.predict(X_Test_Scaled)
cm_svm = confusion_matrix(Y_Test, Y_Pred_svm)
print(cm_svm)
print()
TP_svm = cm_svm[0][0]
FP_svm = cm_svm[0][1]
TN_svm = cm_svm[1][1]
FN_svm = cm_svm[1][0]

print('Success Rate = ',(TP_svm+TN_svm)/(TP_svm+TN_svm+FN_svm+FP_svm))
print('Misclassificate Rate = ',(FP_svm+FN_svm)/(TP_svm+TN_svm+FN_svm+FP_svm))
print('Sensitivity/tp_rate = ', TP_svm/(TP_svm+FN_svm))
print('Specificity/tn_rate = ', TN_svm/(TN_svm+FP_svm))
print('fp rate = ',FP_svm/(TN_svm+FP_svm))
print('fn rate = ',FN_svm/(TP_svm+FN_svm))
```

```
Fitting 10 folds for each of 90 candidates, totalling 900 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

The best parameters for using this model is {'C': 0.01, 'gamma': 1, 'kernel': 'linear'}
Support Vector Machines Accuracy on Training Data with best parameters after 10 Fold Cross Validation is : 0.969577922077922

[[86  3]
 [ 1 50]]

Success Rate =  0.9714285714285714
Misclassificate Rate =  0.02857142857142857
Sensitivity/tp_rate =  0.9885057471264368
Specificity/tn_rate =  0.9433962264150944
fp rate =  0.05660377358490566
fn rate =  0.011494252873563218

[Parallel(n_jobs=1)]: Done 900 out of 900 | elapsed:    5.5s finished
```

3.2.6 Decision Tree Model

Decision Trees in Machine Learning are predictive models that map object attributes to values. They recursively partition data into classes, similar to a flowchart. Each non-leaf node represents a test on an attribute, branches show outcomes, and leaf nodes represent decisions. The root node is the best predictor. Decision Trees handle numerical and categorical data, allowing decision-makers to choose the best alternative based on information gain. Popular methods include ID3, C4.5, C5.0, and CART, which use entropy-based measures.

Information Gain: When a decision tree node is used to divide training instances into smaller subsets, the entropy (uncertainty) changes. Information gain quantifies this entropy change and helps determine the most informative attribute for classification.

Entropy: Entropy measures the uncertainty or impurity of a random variable. It reflects the amount of information contained in a collection of examples. Higher entropy indicates greater uncertainty.

Gini Index: The Gini Index is a metric used to assess how often a randomly selected element would be incorrectly classified. Attributes with lower Gini Index values are preferred as they contribute to better classification accuracy.

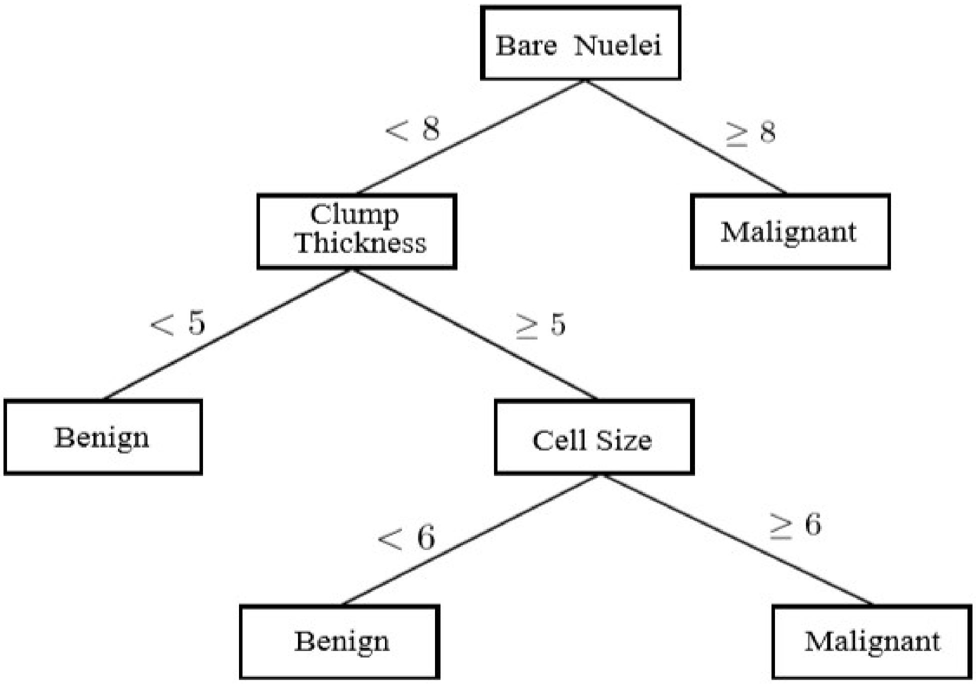
The different types of decision tree algorithms are as follows:

1. ID3 Decision Tree: This algorithm utilizes Information Gain to determine the attribute that should be used for classifying the current subset of data. Information gain is calculated recursively for each level of the tree.

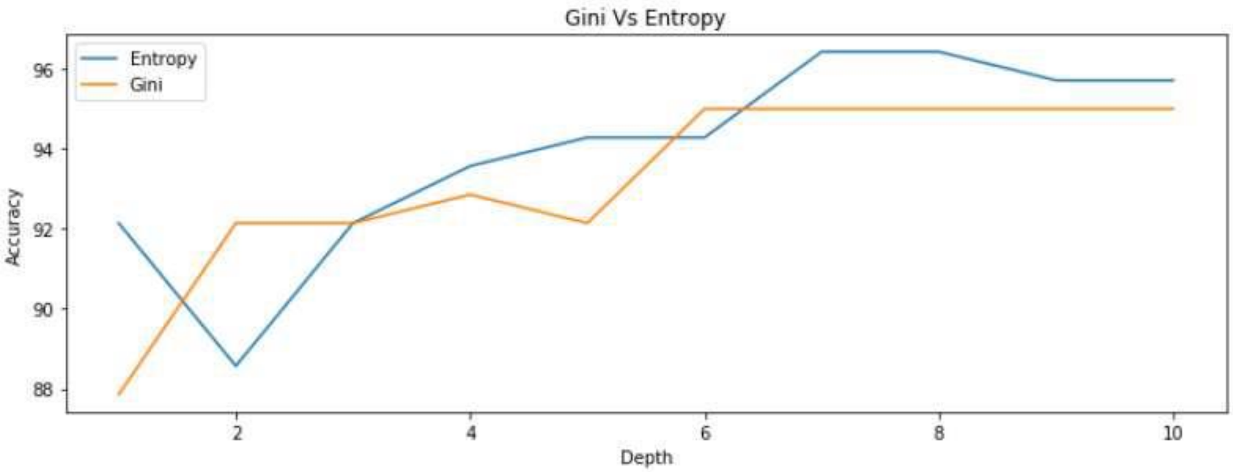
2. C4.5: C4.5 is an improvement over the ID3 algorithm. It can use either Information Gain or Gain Ratio to select the attribute for classification. Unlike ID3, C4.5 can handle continuous and missing attribute values.

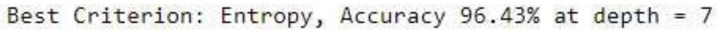
3. Classification and Regression Tree (CART): CART is a versatile learning algorithm that can generate both regression trees and classification trees based on the dependent variable.

To illustrate the functioning of Decision Trees in the context of Breast Cancer diagnosis and prognosis, an example of a decision tree structure for predicting whether a person has malignant or benign Breast Cancer is provided.



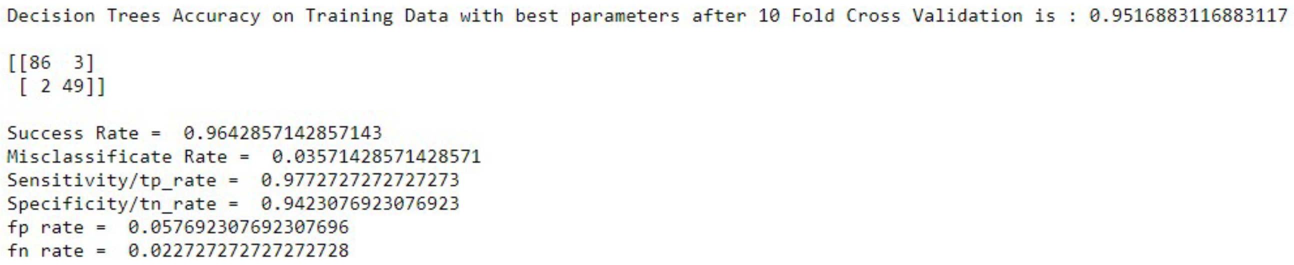
In this project the default Decision Tree Classifier (CART) has been used for which the maximum depth and criterion (Entropy or Gini) were tuned in order to achieve the best testing accuracy.
The following graph shows the variation in accuracy as and when the maximum depth of decision tree is increased, for both Entropy and Gini –



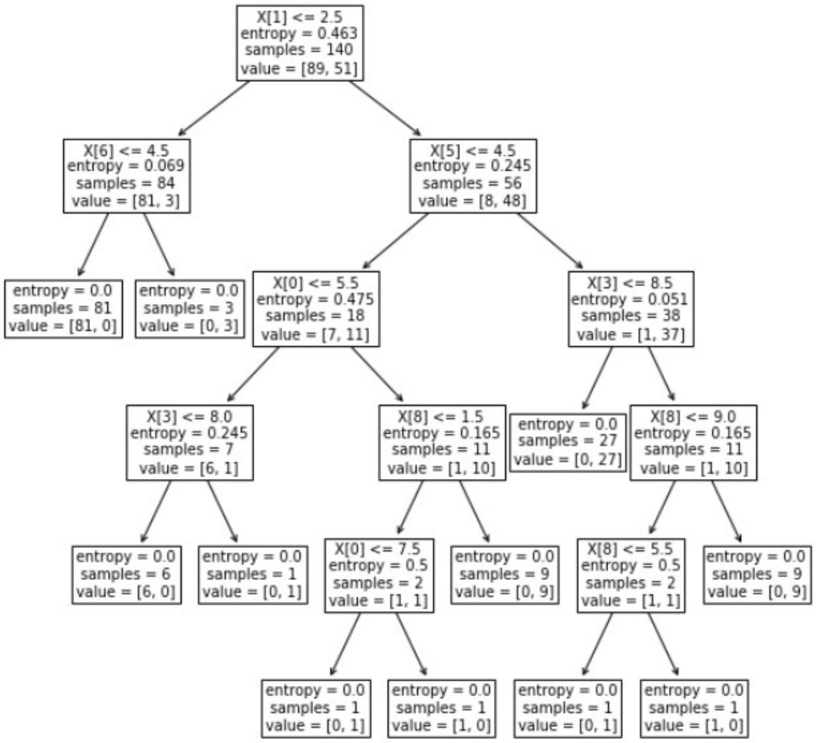


```
Best Criterion: Entropy, Accuracy 96.43% at depth = 7
```

From the graph it can be clearly seen that Entropy for a Maximum Depth of 7 gives the best training accuracy. The classifier is then trained with these best parameters and testing accuracy is calculated.



```
Decision Trees Accuracy on Training Data with best parameters after 10 Fold Cross Validation is : 0.9516883116883117

[[86  3]
 [ 2 49]]

Success Rate =  0.9642857142857143
Misclassificate Rate =  0.03571428571428571
Sensitivity/tp_rate =  0.9772727272727273
Specificity/tn_rate =  0.9423076923076923
fp rate =  0.057692307692307696
fn rate =  0.022727272727272728
```



3.2.7 Random Forest Model

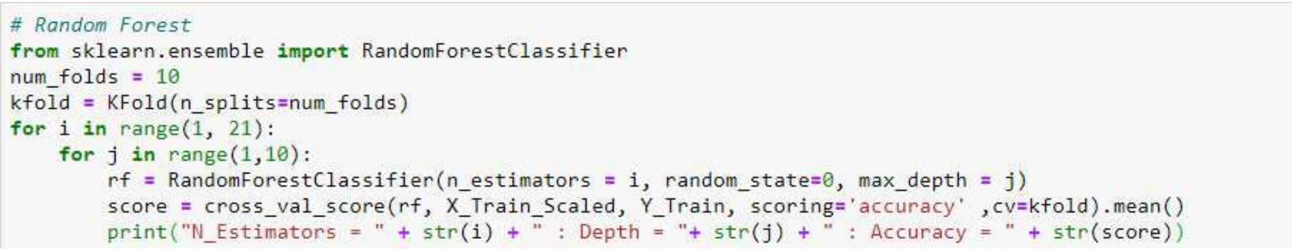
Random forests are a popular machine learning approach that improves upon the limitations of decision trees by leveraging ensemble methods. It involves constructing a forest of decision trees with randomness, which leads to better prediction performance and reduced instability. Random forests split data into sub-samples, train decision tree classifiers on each sub-sample, and average the predictions. This technique reduces variance and bias. It is a flexible and easy-to-use algorithm that produces excellent results without hyper-parameter tuning. Random forests are widely used for both classification and regression tasks. They are highly accurate, robust, and less prone to overfitting. They can handle missing values by using median replacements or computing proximity-weighted averages. Feature importance can be determined to identify the most influential features. However, generating predictions with random forests is slower due to the involvement of multiple decision trees. Interpreting the model is also more challenging compared to a single decision tree.
Random Forest and Decision Trees have some notable differences:

- Random Forest is a collection of multiple decision trees.
- Decision trees with excessive depth can be prone to overfitting, but Random Forest mitigates this issue by creating trees on random subsets of the data.
- Decision trees are computationally faster compared to Random Forest.
- While Decision Trees are easily interpretable and can be converted into rules, Random Forest is more challenging to interpret.

Random forests offer a useful feature selection indicator. Scikit-learn provides a variable with the model that shows the relative importance of each feature in the prediction. This score helps identify the most important features and exclude the least important ones. Random forest uses Gini importance or mean decrease in impurity (MDI) to calculate feature importance. Gini importance measures the decrease in model fit or accuracy when a variable is dropped. The larger the decrease, the more significant the variable. Optimizing the number of estimators and maximum depth of a tree can further improve results.

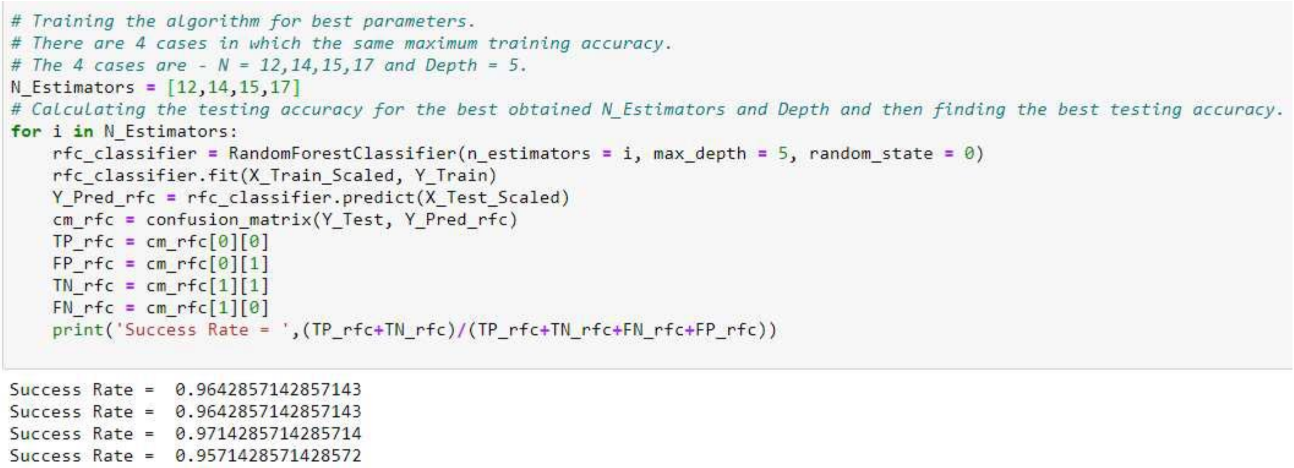
□ Number of Estimators defines the number of trees in the forest.
□ Maximum Depth defines the maximum depth each tree can go to or it can also be described as the length of
the longest path from the tree root to a leaf. The root node is considered to have a depth of 0.

The training process involved adjusting the depth and number of trees in the forest to optimize the training accuracy and find the best parameters. Once the best parameter combination was determined, the model was trained accordingly. Finally, the testing accuracy of the trained model was evaluated.



```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
num_folds = 10
kfold = KFold(n_splits=num_folds)
for i in range(1, 21):
    for j in range(1,10):
        rf = RandomForestClassifier(n_estimators = i, random_state=0, max_depth = j)
        score = cross_val_score(rf, X_Train_Scaled, Y_Train, scoring='accuracy' ,cv=kfold).mean()
        print("N_Estimators = " + str(i) + " : Depth = "+ str(j) + " : Accuracy = " + str(score))
```

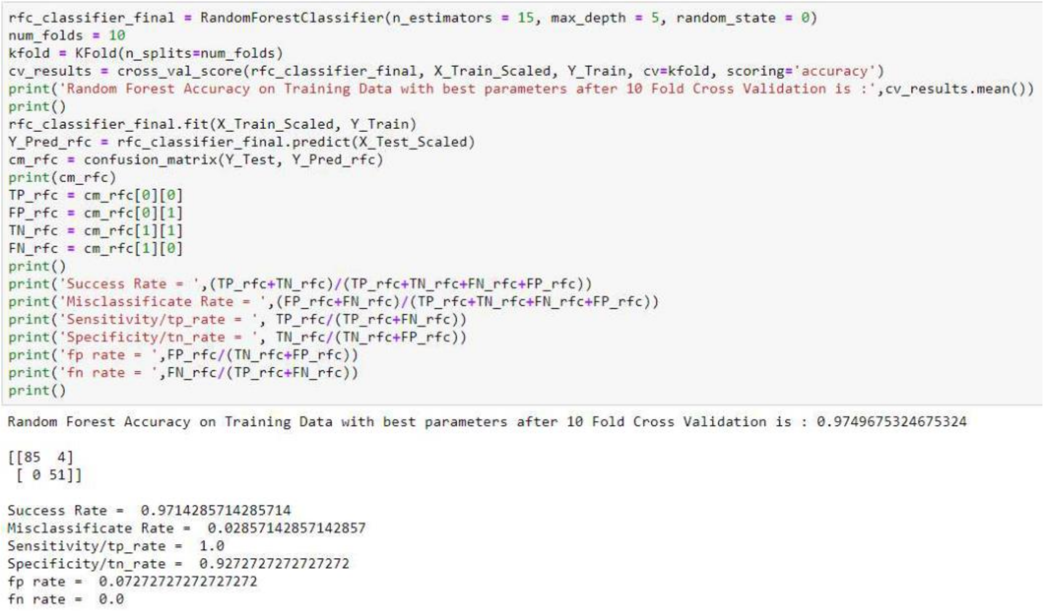
The provided code snippet involved calculating the training accuracy using various combinations of n_estimators and max_depth. The best training accuracies were determined through observation. It was observed that the highest training accuracy was achieved with max_depth = 5. However, for this particular depth, there were four different values of n_estimators (12, 14, 15, and 17) that yielded the same training accuracy. Subsequently, the model was trained four times, each time using a different n_estimators value while keeping max_depth constant, and the corresponding testing accuracies were recorded.



```
# Training the algorithm for best parameters.
# There are 4 cases in which the same maximum training accuracy.
# The 4 cases are - N = 12,14,15,17 and Depth = 5.
N_Estimators = [12,14,15,17]
# Calculating the testing accuracy for the best obtained N_Estimators and Depth and then finding the best testing accuracy.
for i in N_Estimators:
    rfc_classifier = RandomForestClassifier(n_estimators = i, max_depth = 5, random_state = 0)
    rfc_classifier.fit(X_Train_Scaled, Y_Train)
    Y_Pred_rfc = rfc_classifier.predict(X_Test_Scaled)
    cm_rfc = confusion_matrix(Y_Test, Y_Pred_rfc)
    TP_rfc = cm_rfc[0][0]
    FP_rfc = cm_rfc[0][1]
    TN_rfc = cm_rfc[1][1]
    FN_rfc = cm_rfc[1][0]
    print('Success Rate = ',(TP_rfc+TN_rfc)/(TP_rfc+TN_rfc+FN_rfc+FP_rfc))
```

```
Success Rate =  0.9642857142857143
Success Rate =  0.9642857142857143
Success Rate =  0.9714285714285714
Success Rate =  0.9571428571428572
```

It was observed that for n_estimators = 15 and max_depth = 5, the best testing accuracy was obtained.

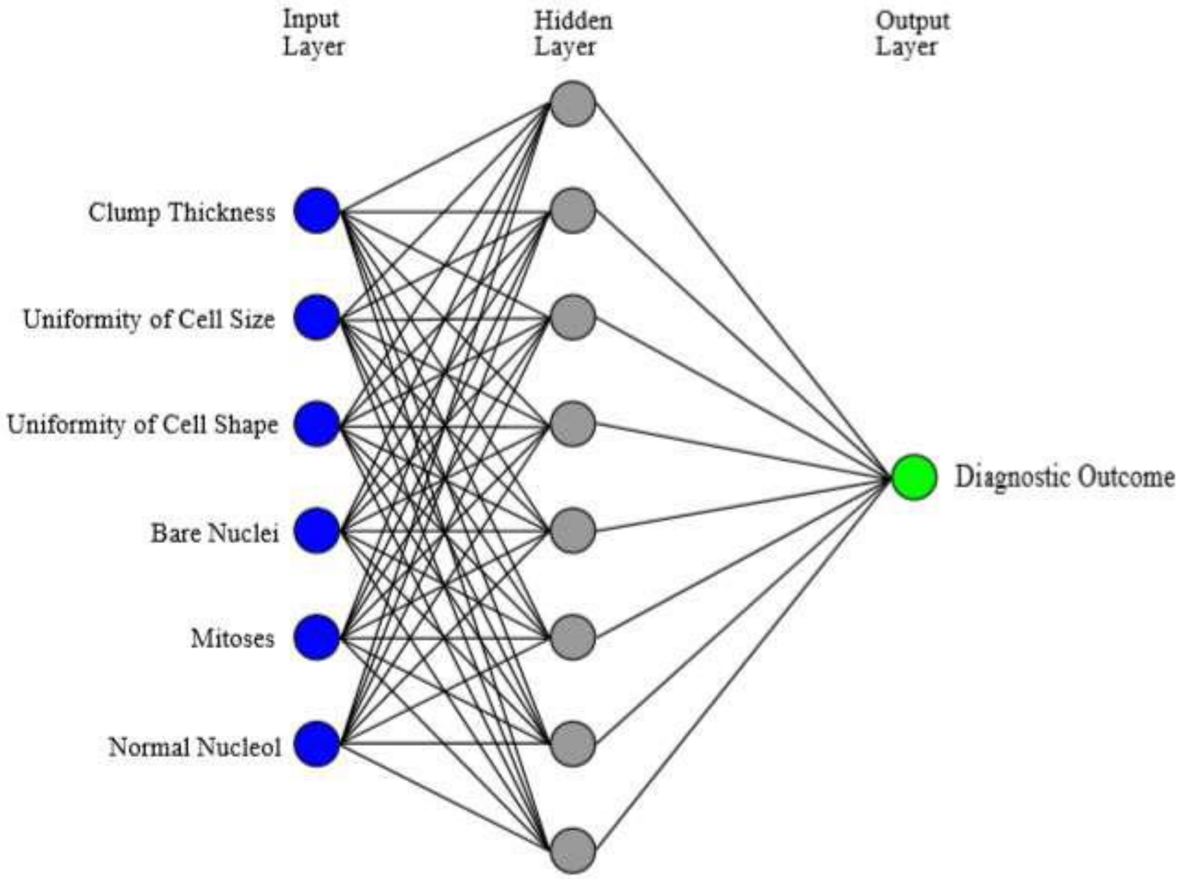


```
rfc_classifier_final = RandomForestClassifier(n_estimators = 15, max_depth = 5, random_state = 0)
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(rfc_classifier_final, X_Train_Scaled, Y_Train, cv=kfold, scoring='accuracy')
print('Random Forest Accuracy on Training Data with best parameters after 10 Fold Cross Validation is :',cv_results.mean())
print()
rfc_classifier_final.fit(X_Train_Scaled, Y_Train)
Y_Pred_rfc = rfc_classifier_final.predict(X_Test_Scaled)
cm_rfc = confusion_matrix(Y_Test, Y_Pred_rfc)
print(cm_rfc)
TP_rfc = cm_rfc[0][0]
FP_rfc = cm_rfc[0][1]
TN_rfc = cm_rfc[1][1]
FN_rfc = cm_rfc[1][0]
print()
print('Success Rate = ',(TP_rfc+TN_rfc)/(TP_rfc+TN_rfc+FN_rfc+FP_rfc))
print('Misclassificate Rate = ',(FP_rfc+FN_rfc)/(TP_rfc+TN_rfc+FN_rfc+FP_rfc))
print('Sensitivity/tp_rate = ', TP_rfc/(TP_rfc+FN_rfc))
print('Specificity/tn_rate = ', TN_rfc/(TN_rfc+FP_rfc))
print('fp rate = ',FP_rfc/(TN_rfc+FP_rfc))
print('fn rate = ',FN_rfc/(TP_rfc+FN_rfc))
print()
```

```
Random Forest Accuracy on Training Data with best parameters after 10 Fold Cross Validation is : 0.9749675324675324

[[85  4]
 [ 0 51]]

Success Rate =  0.9714285714285714
Misclassificate Rate =  0.02857142857142857
Sensitivity/tp_rate =  1.0
Specificity/tn_rate =  0.9272727272727272
fp rate =  0.07272727272727272
fn rate =  0.0
```

3.2.8 Artificial Neural Network Model

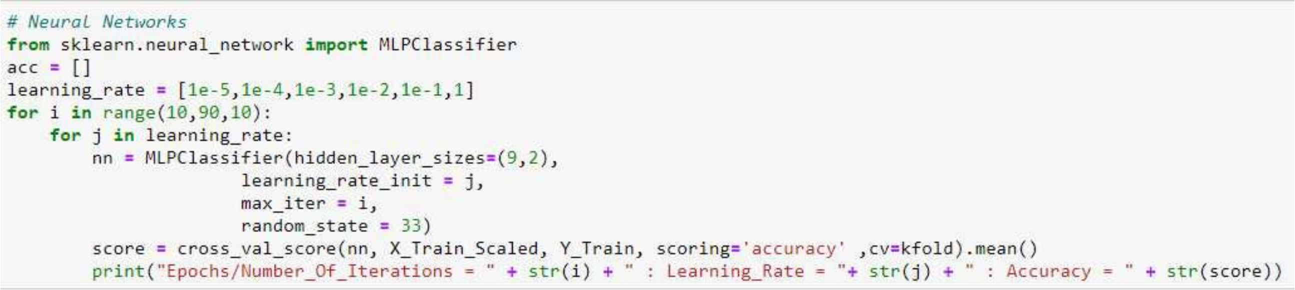
Artificial Neural Networks (ANN) are mathematical algorithms inspired by the structure and function of biological neurons. They consist of interconnected nodes (neurons) and edges (synapses). Input data is processed through weighted synapses and calculations are performed in neurons, which can transmit information to other neurons or represent the output. Neural Networks learn the weights of connections through training data, enabling them to make predictions or classifications for new input data. Neural Networks are capable of handling complex models without the need for intricate feature engineering. Multilayer Neural Networks, such as the Multi-Layer Perceptron (MLP), can address nonlinear problems and provide solutions by utilizing hidden layers for input transformation. An MLP typically includes an input layer, hidden layers, and an output layer. Neurons in the hidden layers perform attribute transformations, and the weights of the neurons represent the features in the transformed domain. The output layer determines the final output pattern based on these features. ANN can be trained to predict outcomes by adjusting the weights based on input data.



In this project, an ANN model is trained for binary classification using nine input features and two output classes. The learning rate and maximum iterations (epochs) are optimized to achieve the best training accuracy.

The learning rate determines how quickly the network adapts its beliefs based on new information. A higher learning rate can lead to faster changes, but it may also result in abrupt or unstable outcomes. Finding an appropriate learning rate is crucial for convergence and efficient training.

Epochs or maximum iterations refer to the number of times the algorithm processes the entire dataset. Each epoch involves a forward pass and a backward pass through all the training examples. It helps the network learn from the entire dataset and refine its predictions over time.

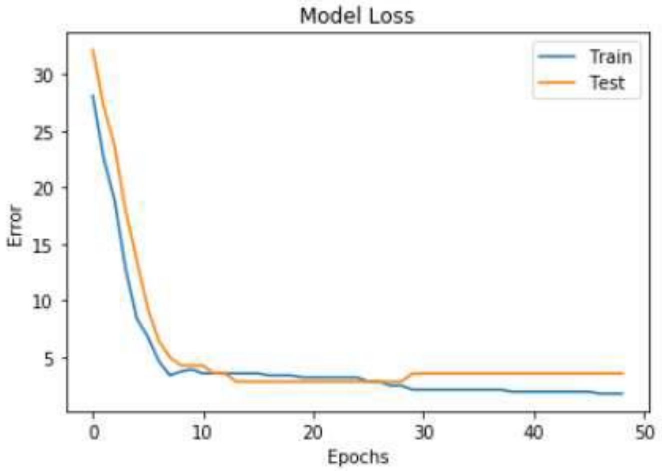


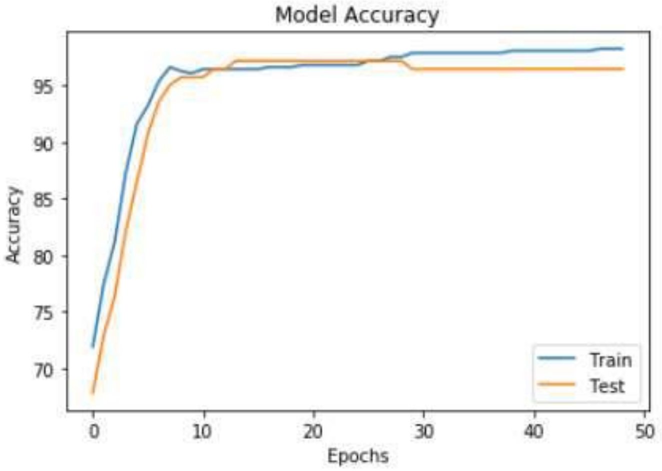
```python
# Neural Networks
from sklearn.neural_network import MLPClassifier
acc = []
learning_rate = [1e-5,1e-4,1e-3,1e-2,1e-1,1]
for i in range(10,90,10):
    for j in learning_rate:
        nn = MLPClassifier(hidden_layer_sizes=(9,2),
                           learning_rate_init = j,
                           max_iter = i,
                           random_state = 33)
        score = cross_val_score(nn, X_Train_Scaled, Y_Train, scoring='accuracy' ,cv=kfold).mean()
        print("Epochs/Number_Of_Iterations = " + str(i) + " : Learning_Rate = "+ str(j) + " : Accuracy = " + str(score))
```

The provided code snippet displays the training accuracy for different combinations of learning rate and number of epochs. Upon examination, it was found that the highest training accuracy was achieved when the learning rate was set to 1e-2. For this learning rate, the number of epochs that yielded the best accuracy were 40 and 50.

However, it is important to consider the possibility of overfitting when increasing the number of epochs with a constant learning rate. Overfitting can result in poor generalization of the model, leading to lower testing accuracy. To address this, the number of epochs was varied within a specific range (up to 50 epochs) while keeping the learning rate at 1e-2. This allowed for the determination of the best testing accuracy.

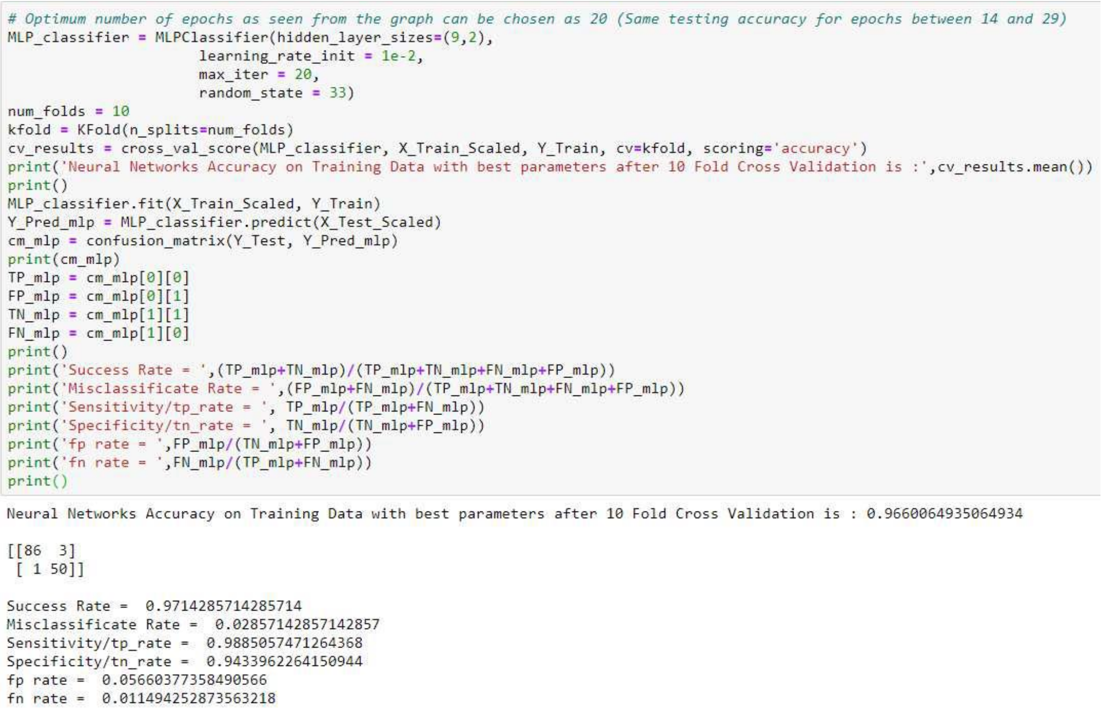
The obtained results are visualized through graphs showcasing the variations of training and testing accuracy, as well as training and testing error.





The graphs show that as the number of epochs increases, both training and testing accuracy generally improve. However, there is a point where the testing accuracy begins to decline while the training accuracy continues to increase. This indicates that the model is overfitting the data, resulting in higher training accuracy but lower testing accuracy.

The optimal number of epochs is the point where the training accuracy starts to degrade. In this model, any value between 14 and 29 epochs yields the same testing accuracy, indicating that the model reaches its optimal performance within that range.



```
# Optimum number of epochs as seen from the graph can be chosen as 20 (Same testing accuracy for epochs between 14 and 29)
MLP_classifier = MLPClassifier(hidden_layer_sizes=(9,2),
                      learning_rate_init = 1e-2,
                      max_iter = 20,
                      random_state = 33)
num_folds = 10
kfold = KFold(n_splits=num_folds)
cv_results = cross_val_score(MLP_classifier, X_Train_Scaled, Y_Train, cv=kfold, scoring='accuracy')
print('Neural Networks Accuracy on Training Data with best parameters after 10 Fold Cross Validation is :',cv_results.mean())
print()
MLP_classifier.fit(X_Train_Scaled, Y_Train)
Y_Pred_mlp = MLP_classifier.predict(X_Test_Scaled)
cm_mlp = confusion_matrix(Y_Test, Y_Pred_mlp)
print(cm_mlp)
TP_mlp = cm_mlp[0][0]
FP_mlp = cm_mlp[0][1]
TN_mlp = cm_mlp[1][1]
FN_mlp = cm_mlp[1][0]
print()
print('Success Rate = ',(TP_mlp+TN_mlp)/(TP_mlp+TN_mlp+FN_mlp+FP_mlp))
print('Misclassificate Rate = ',(FP_mlp+FN_mlp)/(TP_mlp+TN_mlp+FN_mlp+FP_mlp))
print('Sensitivity/tp_rate = ', TP_mlp/(TP_mlp+FN_mlp))
print('Specificity/tn_rate = ', TN_mlp/(TN_mlp+FP_mlp))
print('fp rate = ',FP_mlp/(TN_mlp+FP_mlp))
print('fn rate = ',FN_mlp/(TP_mlp+FN_mlp))
print()
```

```
Neural Networks Accuracy on Training Data with best parameters after 10 Fold Cross Validation is : 0.9660064935064934

[[86  3]
 [ 1 50]]

Success Rate =  0.9714285714285714
Misclassificate Rate =  0.02857142857142857
Sensitivity/tp_rate =  0.9885057471264368
Specificity/tn_rate =  0.9433962264150944
fp rate =  0.05660377358490566
fn rate =  0.011494252873563218
```

Result Analysis

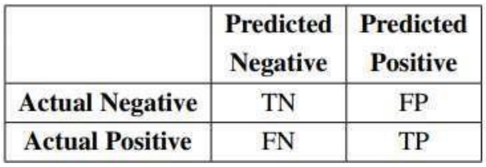
Following the application and implementation of machine learning models, the next stage is to determine the model's effectiveness, or how it performed on the datasets. This is done by applying the models to the previously created test dataset.

4.1 Performance Metrics

The performance of the Machine Learning algorithms has been assessed using a variety of performance indicators. Performance indicators related to classifications are covered in this report because it honestly addresses classification issues. If the target variable for predicting Breast Cancer is 1 (malignant), it is a positive instance, indicating that the patient has Breast cancer. The patient does not have cancer if the target variable is 0 (benign), which is a negative instance.

4.1.1 Confusion Matrix

The effectiveness of a classification system is summarized using a method known as a confusion matrix. By comparing the proportion of positive occurrences that are correctly or mistakenly categorized with that of negative instances, it is maybe the simplest technique to control how well a classification model performs. As seen below, a confusion matrix has rows that indicate real labels and columns that represent expected labels.



	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

True Positives (TP): When a patient has problems (in this example, breast cancer) and is also identified by the model as having complications, this is referred to as an instance where both the predicted and actual classes are true (1).

True Negatives (TN): The instances where the predicted class and actual class are both False (0) that is, when a patient does not have complications and is also identified by the model as not having complications are known as true negatives.

False Negative (FN): When a patient is categorized by the model as not having problems even if they do, this is an instance where the predicted class is False (0) but the actual class is True (1).

False Positive (FP): False positives are situations where the predicted class is True (1) but the actual class is False (0); in other words, when the model classifies a patient as having complications even though they do not.

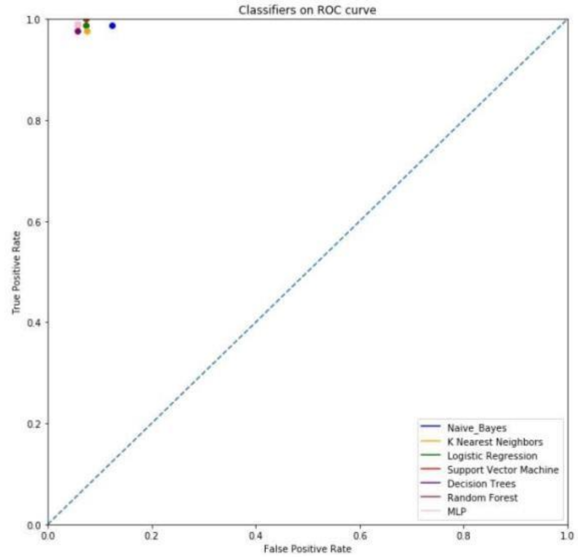
Accuracy/Success Rate: One of the criteria used to evaluate classification models is accuracy. The percentage of a forecast that is accurate. It calculates the proportion of accurate forecasts among all the model's predictions. The accuracy formula is (TP+TN)/ (TP+TN +FP+FN).

Sensitivity/TP Rate: Sensitivity has a relationship with the classifier's ability to recognize favorable results. It is a measurement of how many patients, out of those who have complications, are labeled as having those complications. The following is the formula for specificity Precision =TP/(TP+FN).

Specificity/TN Rate: Specificity is correlated with a classifier's ability to detect undesirable results. It is a measurement of the proportion of patients who are labeled as not having complications among those who did not. Specificity can be calculated by TN/(TN+FP).

4.1.2 Comparing Classifiers based on ROC Curves

Receiver Operating Characteristics are referred to as ROC. The TP rate (sensitivity) and FP rate (1-specificity) are plotted on the ROC graph's y and x axes, respectively. The relative tradeoffs between benefits and costs are therefore displayed by a ROC curve. With the aid of ROC analysis, it is possible to choose potentially optimal models and eliminate less-than-ideal ones without first specifying the cost context or the class distribution. A direct and natural connection can be made between cost-benefit analysis and diagnostic decision-making through ROC analysis. Classifiers with high sensitivity and specificity in the top left corner of the ROC curve are favored (classifier having the highest sensitivity and specificity is the best model for that dataset and hence is used for prediction). Here is the ROC curve for the classifiers mentioned previously:



One confusion matrix, or one ROC point, is produced when a classifier algorithm is applied to the test set. By thresholding the classifier in relation to its complexity, we may produce a ROC curve. A new point in ROC space results from each level of complexity in a hypothesis class. By comparing ROC curves in the same ROC space for the two learning schemes, it is possible to compare two distinct learning schemes on the same domain. These are the ROC curves for several classifiers:

Interpreting the ROC Curve - It is evident from the ROC curve that all trained classifiers are in the top left corner, indicating that they all produce positive results for the dataset. However, since every classifier is in the upper left corner, their relative placements must be taken into account. The classifier with the lowest false positive rate, the support vector machine, is located farthest to the left on the ROC curve. There isn't a classifier that has the highest true positive and false negative rates relative to other classifiers. Therefore, while selecting the optimum model, there is a trade-off between TP and FP rate. False negative rates for breast cancer prediction should be as low as feasible.

The genuine positive rate should thus be as high as feasible. Looking at the ROC curve does not allow for any definitive conclusions. Therefore, in order to choose the optimal classifier for the provided dataset, we must take into account the sensitivity and specificity numerical values.

4.2 Model Performances

Naïve Bayes (NB), K-Nearest Neighbors Classifier (KNN) and Logistic Regression (LR), Support Vector Machines (SVM), Decision Trees (DT), Random Forests and Multi-Layer Perceptron (MLP) have been applied on the dataset classification techniques that were utilized to analyze the dataset. Accuracy, Sensitivity, and Specificity are used to evaluate each experiment's algorithm performance. The findings of several measures used by the algorithms to predict breast cancer are shown in the table below:

Algorithm	Accuracy	Misclassification Rate	Sensitivity	Specificity	FP rate	FN Rate
Naïve Bayes	0.943	0.057	0.988	0.877	0.123	0.012
K Nearest Neighbors	0.957	0.043	0.977	0.924	0.075	0.022
Logistic Regression	0.964	0.036	0.988	0.926	0.074	0.012
SVM	0.971	0.029	0.989	0.943	0.057	0.011
Decision Tree	0.964	0.036	0.977	0.942	0.058	0.023
Random Forest	0.971	0.029	1.0	0.927	0.073	0
ANN (MLP)	0.971	0.029	0.989	0.943	0.057	0.011

Conclusion

The Wisconsin Madison Breast Cancer Diagnosis Problem is addressed in this study as a pattern classification issue. In this research, a number of machine learning models were examined, and the best model was chosen by taking into account the model with the highest sensitivity and specificity.

SVM and MLP achieved the same accuracy, sensitivity, and specificity of 97.1%, 98.9%, and 94.3% respectively. In contrast, the Random Forest Classifier had a sensitivity of 100% (no false negatives) but a specificity of 92.7%, slightly lower than MLP and SVM.

Considering the average of sensitivity and specificity, MLP and SVM performed better with an average of 96.60% compared to the Random Forest Classifier with an average of 96.35%.

Therefore, it can be concluded that both SVM and MLP provided optimal results when considering both sensitivity and specificity. However, it is noteworthy that Random Forests achieved a remarkable sensitivity of 100% and an average of 96.35%, slightly lower than MLP and SVM.

Bibliography

[1] Breast cancer facts and figures 2003-2004 (2003). American Cancer Society.

[2] Fine Needle Aspiration Biopsy of the Breast. American Cancer Society.

[3] William H Wolberg, W Nick Street, and Olvi L Mangasarian. Breast cancer Wisconsin (diagnostic) data set.

[4] Delen, D.; Walker, G.; Kadam, A. Predicting breast cancer survivability: A comparison of three data mining methods. Artif. Intell. Med. 2005, 34, 113–127.

[5]. Funahashi, K.; Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. Neural Netw. 1993, 6, 801–806.

[6]. Razi, M.A.; Athappilly, K. A comparative predictive analysis of neural networks (NNs), nonlinear regression and classification and regression tree (CART) models. Expert Syst. Appl. 2005, 29, 65–74.

[7]. Subasi, A.; Ercelebi, E. Classification of EEG signals using neural network and logistic regression. Comput. Methods Prog. Biomed. 2005, 78, 87–99.

[8]. Breiman,L.;Friedman,J.H.;Olshen,R.A.;Stone,C.J. Classification and Regression Trees; CRC Press: Boca Raton, FL, USA, 1984.