Brain Tumor Prediction Using Machine Learning

Group 2

Ishdeep Singh Chadha
1810110087
Samved Rajvanshi
1810110209

Abstract

In today's world, brain tumor and other nervous system cancer is the 10th leading cause of death for men and women. The growth of abnormal cells in the brain leads to brain tumor. There are two main types of tumors: malignant and benign. Malignant is cancerous whereas Benign tumor is non-cancerous. Diagnosing a tumor at an early stage can save a person's life. It is actually a great challenge to find the brain tumor. Detection of Brain Tumor can be done with the help of Machine Learning. Machine Learning, in this field, has played a vital role in correctly predicting the presence of tumor inside the brain. In this report, we have used and compared the performance of some Machine Learning Algorithms: Naïve Bayes, K-Nearest Neighbours, Support Vector Machine, Logistic Regression, Neural Networks and various other algorithms, on Brain Tumor Dataset. Our main purpose is to correctly identify which algorithm gives best results in terms of both sensitivity and specificity, accuracy and false positive rate. The algorithms are coded in Python language and executed in Google Colaboratory; a cloud based Jupyter notebook environment.

Content

Introduction

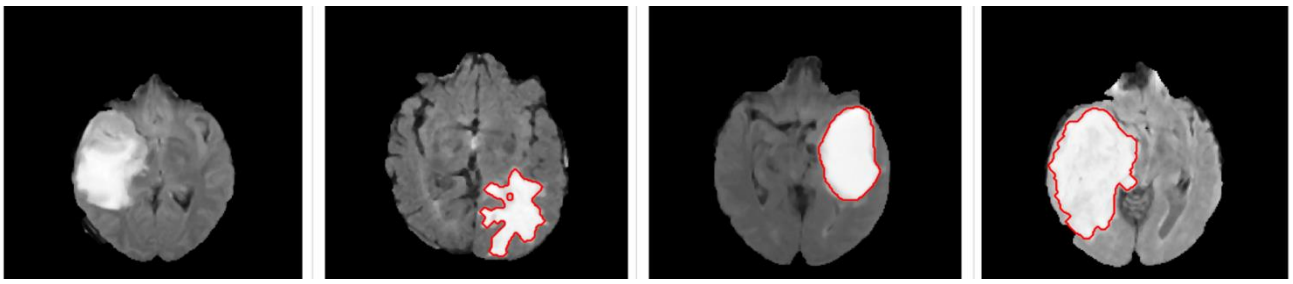
Brain tumors account for 85% to 90% of all primary central nervous system (CNS) tumors. All types of brain tumors may produce symptoms that vary depending on the part of the brain involved. These symptoms may include headaches, seizures, problems with vision, vomiting and mental changes. The headache is classically worse in the morning and goes away with vomiting. Other symptoms may include difficulty walking, speaking or with sensations. As the disease progresses, unconsciousness may occur.

The 5-year survival rate tells you what percent of people live at least 5 years after the tumor is found. The 5-year survival rate for people with a cancerous brain or CNS tumor is almost 36%. The 10-year survival rate is almost 31%. Survival rates decrease with age. The 5-year survival rate for people younger than age 15 is more than 74%. For people aged 15 to 39, the 5-year survival rate is about 71%. The 5-year survival rate for people aged 40 and over is about 21%. However, survival rates vary widely and depend on several factors.

Detection of Brain Tumor manually is a recurring activity which consumes a lot of time and also the results are not accurate. MRI helps in analysis of Brain Tumor along with CT images as well as ultrasonic or X-Rays. The advanced technology of brain MRI image data has created more opportunities for neurosurgeons and medical scientists to diagnose brain tumors. The detection of tumor size at an early stage must be detected accurately for the diagnosis purpose and which will increase the survival rate of the tumor.

The dataset used contains the features extracted from MRI scans of the brain. Further, Naïve Bayes, K-Nearest Neighbours, Logistic Regression, Support Vector Machine, Neural Networks, Decision Trees, and Trees Ensembles techniques like Bagging, Adaboost and Random Forest algorithms have been implemented on the dataset. The results obtained are then measured using various performance metrics to compare among the algorithms in order to find out the best suited model for cancer prediction.

Dataset



This is a brain tumor feature dataset including five first-order features and eight texture features with the target level (in the column Class). Within the research papers we referred, they first used deep learning techniques to extract the features related to the brain tumor from the given images, but due to our constraints we directly took the extracted features and applied machine learning algorithms to predict whether the person has a tumor based on the values of the features.

We choose this dataset for a number of reasons, firstly because the research papers we referred used some similar features compared to this data. Also seeing that all the features have numerical values we decided it would prove to be a suitable dataset for the given requirements of the project.

Independent Features:

Mean: Gives the contribution of individual pixel intensity for the entire image

Variance: Used to find how each pixel varies from the neighbouring pixel

Standard Deviation: measures the deviation of measured Values or the data from its mean.

Skewness: measures of symmetry, or more precisely, the lack of symmetry.

Kurtosis: describes the peakiness e.g., a frequency distribution

The above features are first order features which rely only on the values of individual pixels in the image, and do not express their relationship to other image pixels.

Below are the second order features or texture features:

Contrast: the difference in luminance or colour across the image

Energy: It's the rate of change in the color/brightness/magnitude of the pixels over local areas.

ASM (Angular second moment): is a measure of textural uniformity of an image

Entropy: is a statistical measure of randomness that can be used to characterize the texture of the image

Homogeneity: homogeneity expresses how similar certain elements (pixels) of the image are.

Dissimilarity: is a numerical measure of how different two data objects are.

Correlation: Correlation is the process of moving a filter mask often referred to as kernel over the image and computing the sum of products at each location (CNN alike)

Coarseness: Describes the roughness/harshness of a texture

Dependent Feature:

Class: column defines either the image has tumor or not (1 = Tumor, 0 = Non-Tumor)

Method

Exploratory Data Analysis and Data Visualization

(i) How many samples and attributes are there in the dataset?

```
In [6]: df.shape
Out[6]: (3762, 15)
```

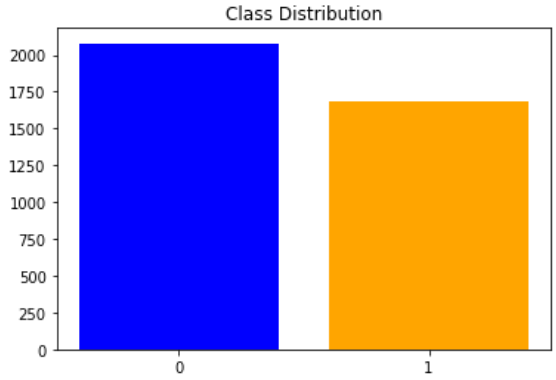
Therefore, we have 3762 samples and 15 attributes/features.

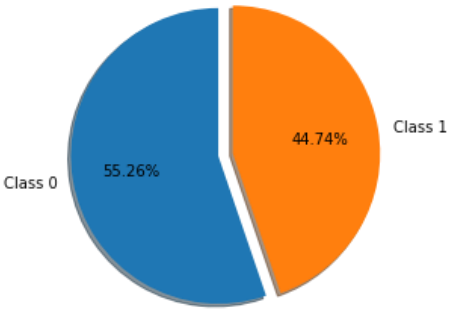
(ii) Datatype of each attribute

```
In [7]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3762 entries, 0 to 3761
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Image               3762 non-null   object
 1   Class               3762 non-null   int64
 2   Mean                3762 non-null   float64
 3   Variance            3762 non-null   float64
 4   Standard Deviation  3762 non-null   float64
 5   Entropy             3762 non-null   float64
 6   Skewness            3762 non-null   float64
 7   Kurtosis            3762 non-null   float64
 8   Contrast            3762 non-null   float64
 9   Energy              3762 non-null   float64
 10  ASM                 3762 non-null   float64
 11  Homogeneity         3762 non-null   float64
 12  Dissimilarity       3762 non-null   float64
 13  Correlation         3762 non-null   float64
 14  Coarseness          3762 non-null   float64
dtypes: float64(13), int64(1), object(1)
memory usage: 441.0+ KB
```

Here we can see that, apart from the Image attribute all other attributes have int/float data type. We thus dropped the Image attribute as it did not have any significant role in determining the Class i.e., target attribute.

(iii) Distribution of 'Class' / dependent attribute. To check if our dataset is balanced or not.





Therefore, we can say that we have a highly balanced dataset as we have 55.26% (2079) of samples classified as Class 0 i.e., Non-Tumor and 44.74% (1683) of samples classified as Class 1 i.e., Tumor.

(iv) Correlation Matrix:

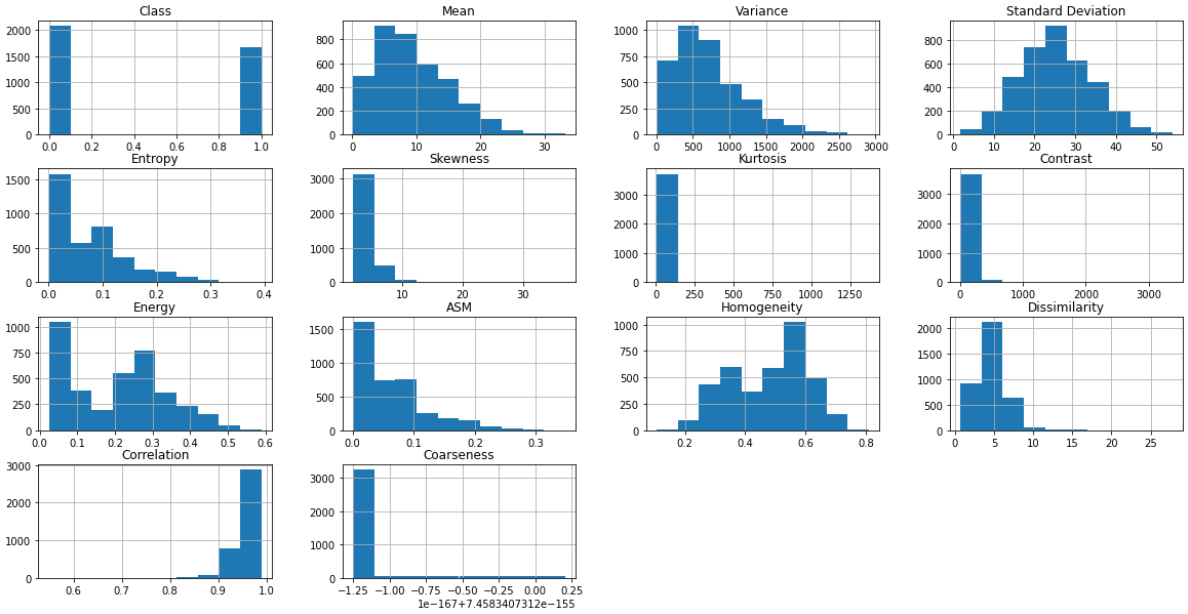
A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

In [12]:
```
### Correlation Matrix

corr_m = df.corr()
corr_m.style.background_gradient(cmap='coolwarm',axis=None).set_precision(2)
```

Out[12]:

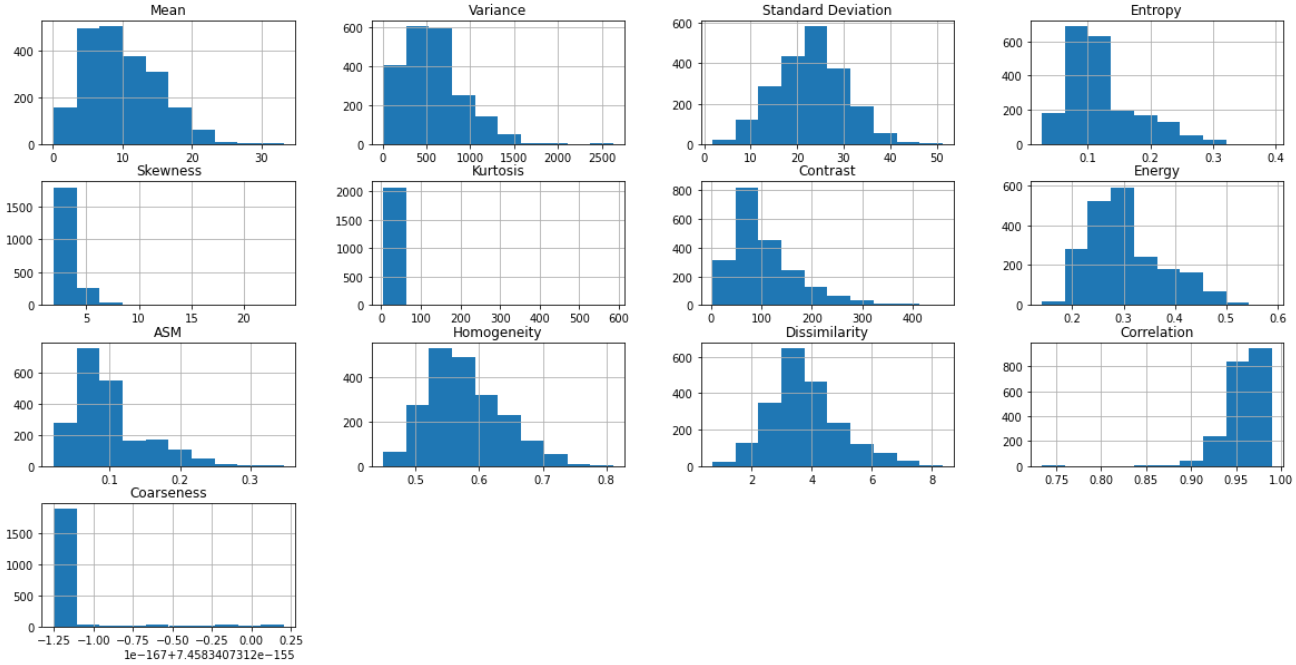
	Class	Mean	Variance	Standard Deviation	Entropy	Skewness	Kurtosis	Contrast	Energy	ASM	Homogeneity	Dissimilarity	Correlation	Coarseness
Class	1.00	-0.10	0.31	0.29	-0.78	0.40	0.24	0.21	-0.86	-0.76	-0.85	0.56	-0.11	nan
Mean	-0.10	1.00	0.78	0.79	-0.10	-0.60	-0.36	-0.05	-0.01	-0.11	0.10	-0.11	0.29	nan
Variance	0.31	0.78	1.00	0.98	-0.34	-0.35	-0.25	0.14	-0.34	-0.34	-0.29	0.24	0.29	nan
Standard Deviation	0.29	0.79	0.98	1.00	-0.35	-0.43	-0.33	0.12	-0.33	-0.34	-0.29	0.22	0.35	nan
Entropy	-0.78	-0.10	-0.34	-0.35	1.00	-0.22	-0.14	-0.14	0.97	1.00	0.85	-0.50	0.12	nan
Skewness	0.40	-0.60	-0.35	-0.43	-0.22	1.00	0.90	0.35	-0.30	-0.21	-0.47	0.51	-0.57	nan
Kurtosis	0.24	-0.36	-0.25	-0.33	-0.14	0.90	1.00	0.30	-0.17	-0.13	-0.31	0.38	-0.59	nan
Contrast	0.21	-0.05	0.14	0.12	-0.14	0.35	0.30	1.00	-0.13	-0.14	-0.27	0.76	-0.43	nan
Energy	-0.86	-0.01	-0.34	-0.33	0.97	-0.30	-0.17	-0.13	1.00	0.96	0.92	-0.55	0.12	nan
ASM	-0.76	-0.11	-0.34	-0.34	1.00	-0.21	-0.13	-0.14	0.96	1.00	0.84	-0.49	0.12	nan
Homogeneity	-0.85	0.10	-0.29	-0.29	0.85	-0.47	-0.31	-0.27	0.92	0.84	1.00	-0.75	0.20	nan
Dissimilarity	0.56	-0.11	0.24	0.22	-0.50	0.51	0.38	0.76	-0.55	-0.49	-0.75	1.00	-0.39	nan
Correlation	-0.11	0.29	0.29	0.35	0.12	-0.57	-0.59	-0.43	0.12	0.12	0.20	-0.39	1.00	nan
Coarseness	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

(v) Distribution of each attribute using histograms.

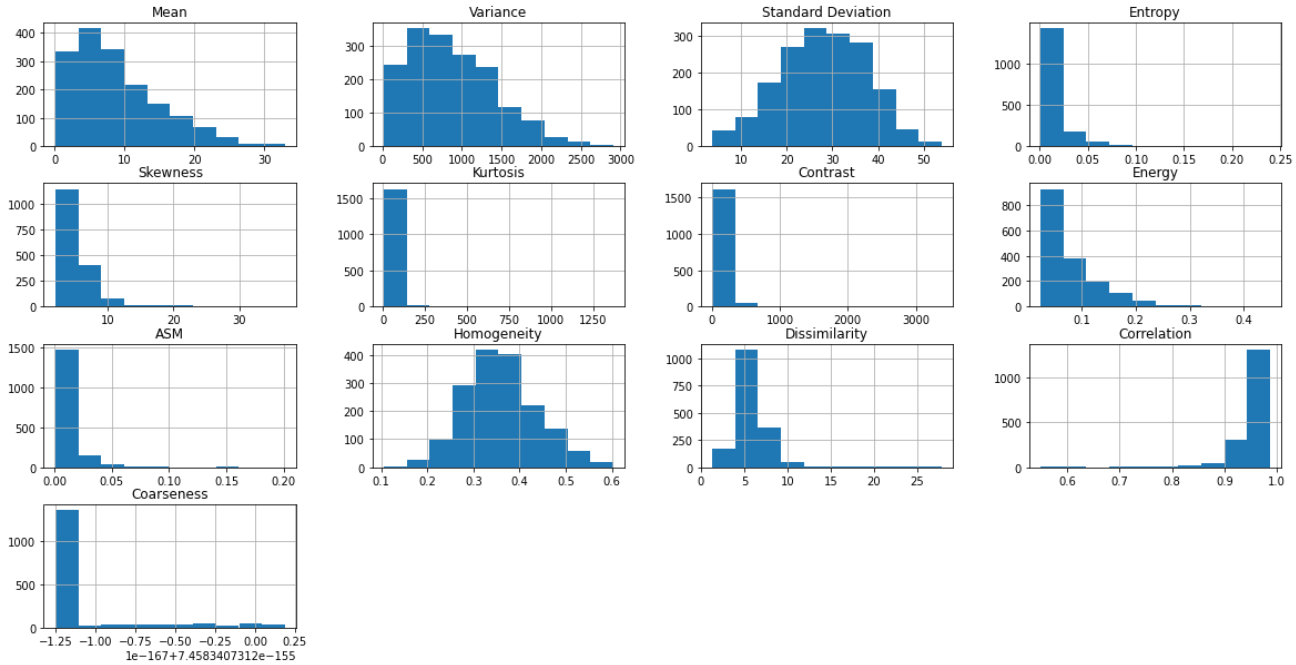


(vii) Distribution of each attribute with respect to target attribute i.e 'Class'.

For Class = 0



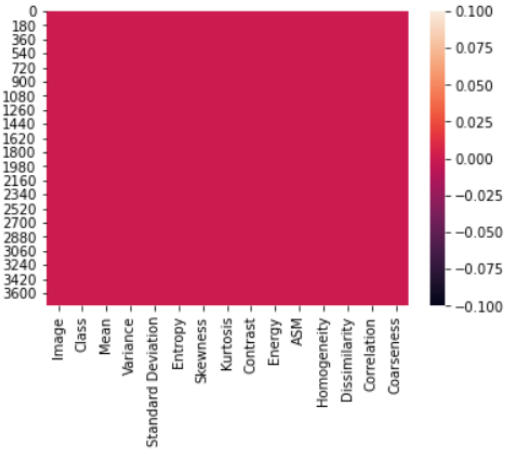
For Class = 1


Mean
Variance
Standard Deviation
Entropy
Skewness
Kurtosis
Contrast
Energy
ASM
Homogeneity
Dissimilarity
Correlation
Coarseness
1e−167+7.4583407312e−155

Data Transformation

1. Finding Missing Values

```
Class                  0
Mean                   0
Variance               0
Standard Deviation     0
Entropy                0
Skewness               0
Kurtosis               0
Contrast               0
Energy                 0
ASM                    0
Homogeneity            0
Dissimilarity          0
Correlation            0
Coarseness             0
dtype: int64
```



Thus, we do not have any null/missing values in any attribute.

2. Outlier Detection and Removal

Interquartile Range Method:

The IQR is calculated as the difference between the 75th and the 25th percentiles of the data and defines the box in a box plot.

For removal of outliers, we first defined lower and upper bounds using the below function by which we found which of the attributes had outliers meaning data points falling outside of this limit.

```
# Setting an upper and lower limit for outliers
def outlier_thresholds(dataframe, variable):
    quartile1 = dataframe[variable].quantile(0.25)
    quartile3 = dataframe[variable].quantile(0.75)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit
```

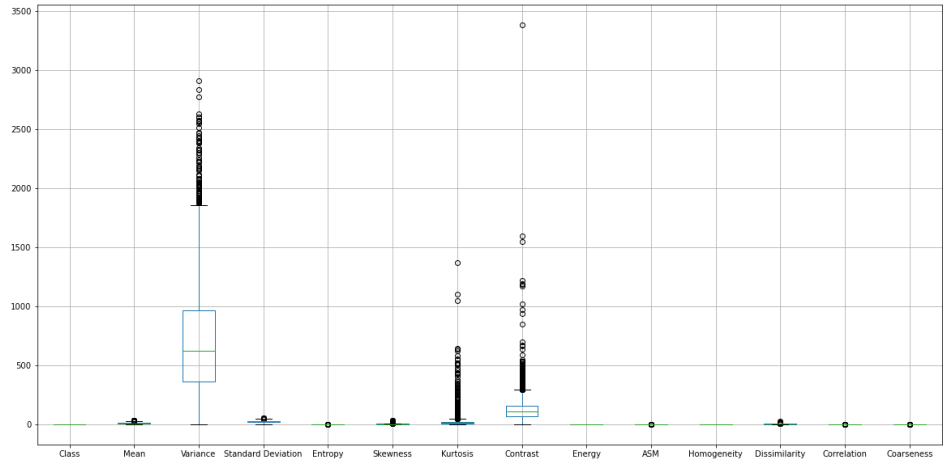
```
Class False
Mean True
Variance True
Standard Deviation True
Entropy True
Skewness True
Kurtosis True
Contrast True
Energy False
ASM True
Homogeneity False
Dissimilarity True
Correlation True
Coarseness True
```

After identifying the attributes, we replaced those data points with the value of our upper limit using the below function.

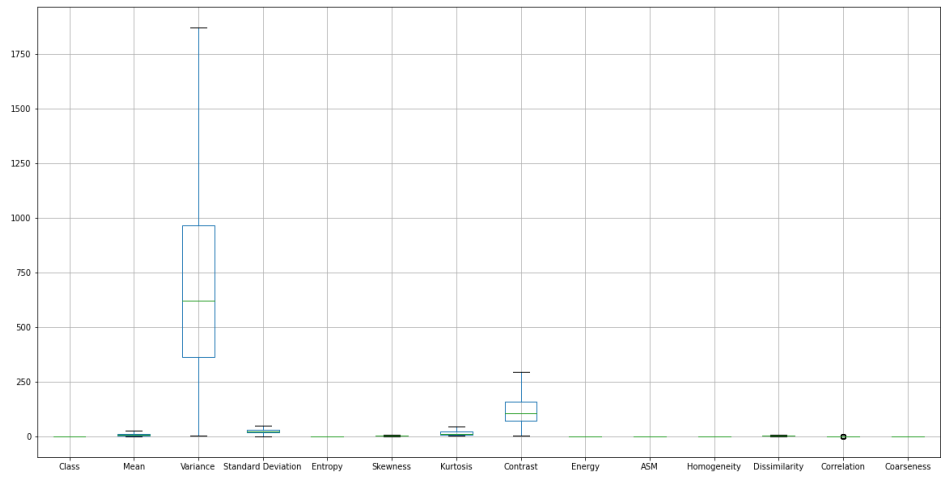
```
# Replacing outliers with upper and lower limit
def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit
```

Here we show two different boxplots of our data before and after outlier removal.

Before:



After:



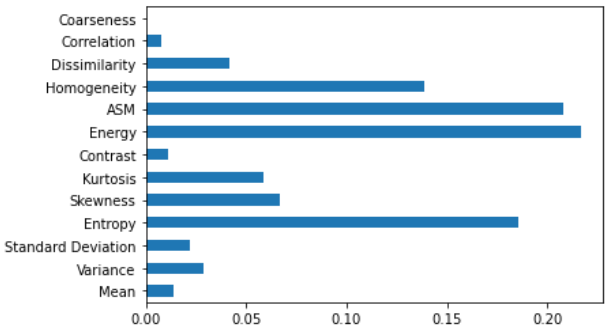
3. Dimensionality Reduction

Feature Selection and Importance:

a. Univariate Selection:

Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features. In the table below(left), Score represents the chi squared value of each attribute.

	Specs	Score
0	Mean	123.124729
1	Variance	103854.495031
2	Standard Deviation	935.344852
3	Entropy	151.972629
4	Skewness	635.713523
5	Kurtosis	9550.360760
6	Contrast	9564.673039
7	Energy	228.638330
8	ASM	123.967254
9	Homogeneity	92.254340
10	Dissimilarity	781.693602
11	Correlation	0.031753
12	Coarseness	0.000000



b. ExtraTrees Classifier:

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result. To perform feature selection using the above forest structure, during the construction of the forest, for each feature, the normalized total reduction in the mathematical criteria used in the decision of feature of split (Gini Index) is computed. This value is called the Gini Importance of the feature. To perform feature selection, each feature is ordered in descending order according to the Gini Importance of each feature and the user selects the top k features according to his/her choice.

From the bar graph seen above(right) we can see the different values of feature importance for each attribute.

Interpreting from the above two techniques, we come to the conclusion that the 'Coarseness' and 'Correlation' features do not affect the Class in any significant way. Hence, we can drop the mentioned features.

4. Feature Normalization

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data

Standard Scalar: Standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

We first used a holdout method to split our data into training (80%) and testing data (20%) and then applied the mentioned library on our training data.

Machine Learning Algorithms used and Results Obtained:

1. Naive Bayes Classifier

Naïve Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature which contributes to the target variable. So, the existence of one feature variable doesn't have an effect on the opposite feature variables. This can be why it's known as Naive. However, in real knowledge sets, the feature variables are dependent on one another therefore this can be one among the drawbacks of Naive Bayes classifier. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. The algorithm for Naive Bayes theorem is:

$$y_{NB} = arg \max_{q} P(y_q) \prod_{j} P(x_j|y_q)$$

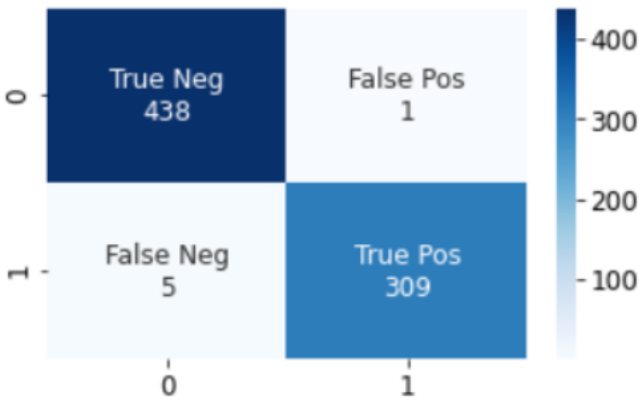
Here, y_{NB} denotes the class output by the naive Bayes classifier.

$$P(\mathbf{x}|y_q) = \frac{\text{Number of times pattern } \mathbf{x} \text{ appears with } y_q \text{ class}}{\text{Number of times } y_q \text{ appears in the data}}$$

The naive Bayes classifier is a non-parametric method. It makes use of the Bayes theorem as the base and estimates the priors $P(y_q)$ and likelihoods $P(x|y_q)$ for an unseen sample x directly from the given dataset. The data is divided into training and test sets, and then a naive Bayes classifier is applied to the test samples using a training set for estimating probability distributions.

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	439
1.0	1.00	0.98	0.99	314
accuracy			0.99	753
macro avg	0.99	0.99	0.99	753
weighted avg	0.99	0.99	0.99	753

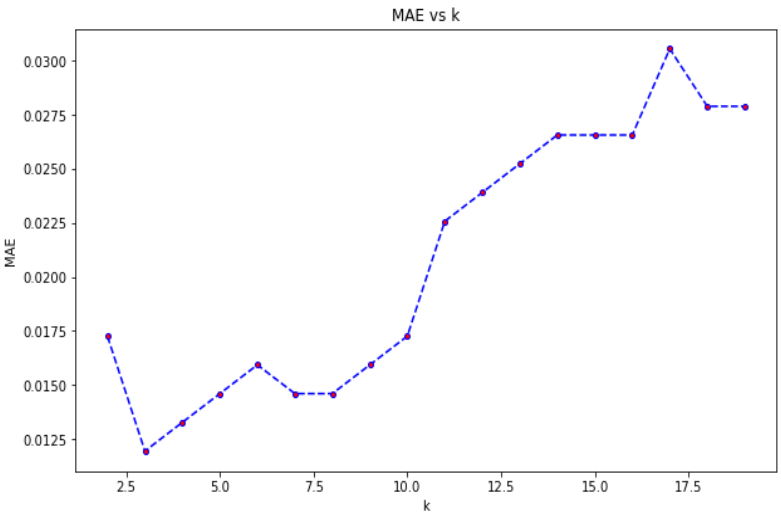
```
Naive Bayes Test Accuracy :  99.20318725099602 %
Training Accuracy:  97.04220671319376 %

TP Rate :  0.9840764331210191
TN Rate :  0.9977220956719818
FP Rate :  0.0032258064516129032
FN Rate :  0.01592356687898089
Success Rate:  0.9920318725099602
Misclassifiaction Rate:  0.0079681274900398
```



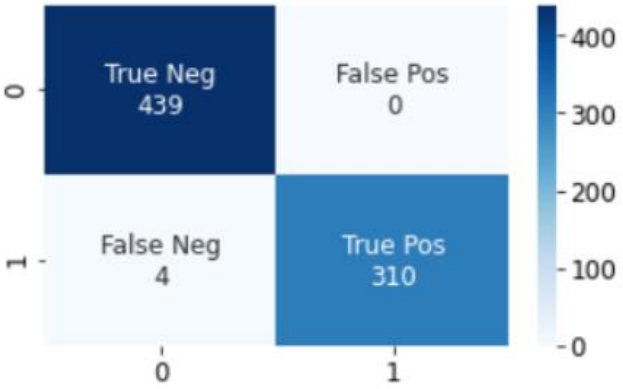
2. K-Nearest Neighbours Model

K-NN algorithm is one of the simplest classification techniques that doesn't make assumptions about the model between class membership and features. It does not involve estimation of parameters in an assumed model, thus making it a non-parametric technique. If it can recognize the characteristic options of one of the objects, it will be additionally predicted for its nearest neighbour. It is based mostly on the plan that any new instance will be classified by the majority vote of its 'k' neighbours, - wherever k is a positive number, sometimes a little variety. It is also sometimes called Memory Based Classification.

The value of k, in this model, is determined with the help of MAE (Mean Absolute Error). Given any test data-set, Mean Absolute Error of the model refers to the mean of the absolute values of each prediction error on all instances of the test data-set. Prediction error is the difference between the actual value and the predicted value. For different numbers of neighbours, the machine is trained and MAE is calculated and compared. The number of neighbours for which the least mean absolute error is observed is used for training the model.



Value of k for which MAE is minimum: 2




```
WITH K=2

TP Rate :  0.9872611464968153
TN Rate :  1.0
FP Rate :  0.0
FN Rate :  0.012738853503184714
Success Rate:  0.9946879150066401
Misclassifiaction Rate:  0.005312084993359893
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       439
         1.0       1.00      0.99      0.99       314

    accuracy                           0.99       753
   macro avg       1.00      0.99      0.99       753
weighted avg       0.99      0.99      0.99       753

KNN Test Accuracy :  99.46879150066401 %
Training Accuracy:  98.93652376204719 %
```

3. Logistic regression

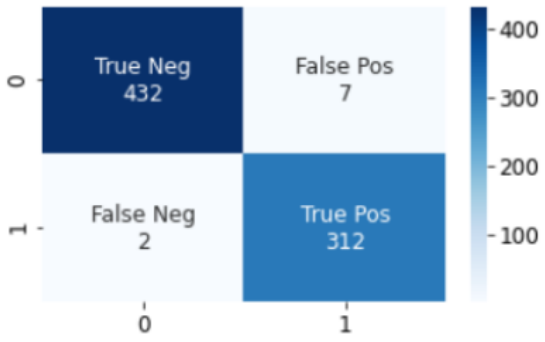
Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of the target or dependent variable is dichotomous, which means there would be only two possible classes. Logistic Regression uses a simple equation which shows the linear relation between the independent variables. We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function. The hypothesis of logistic regression tends it to limit the cost function between 0 and 1.

Logistic regression has parameters that can be optimized to achieve the maximum testing accuracy. The parameters used in this report to achieve the maximum testing accuracy are -

C: It is the regularization parameter, also defined as the inverse of regularization strength (λ). If λ is very low or equal to 0, the model will tend to overfit as it will have enough power to increase the complexity. Whereas if we increase λ, it will tend to under fit. To achieve the best results, C has to be optimized.

Penalty: Used to specify the norm used in the penalization. The two types of penalties used in this report are l1 and l2. l1 signifies Lasso Regression whereas l2 signifies Ridge Regression. Ridge regression adds "squared magnitude" of coefficient as penalty term to the loss function whereas Lasso Regression adds "absolute magnitude" of coefficient as penalty term to the loss function.

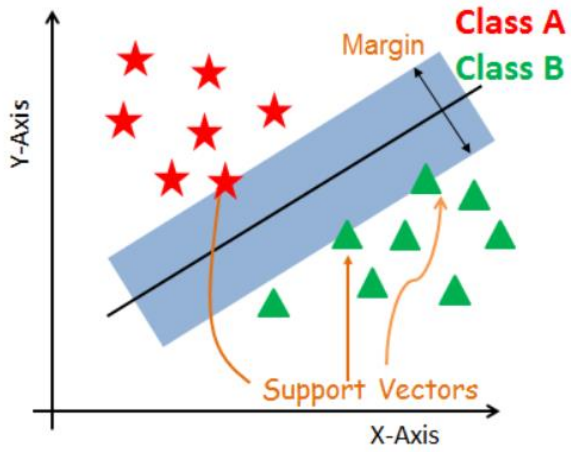
```
TP Rate :  0.9936305732484076
TN Rate :  0.9840546697038725
FP Rate :  0.0219435736677116
FN Rate :  0.006369426751592357
Success Rate:  0.9880478087649402
Misclassifiaction Rate:  0.01195219123505976
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       439
         1.0       0.98      0.99      0.99       314

    accuracy                           0.99       753
   macro avg       0.99      0.99      0.99       753
weighted avg       0.99      0.99      0.99       753

Logistic Regression Test Accuracy :  98.80478087649402 %
Training Accuracy:  98.37155201063477 %
```



4. Support Vector Machine / Support Vector Classifier

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces.



- **Kernel**: The main function of the kernel is to transform the given dataset input data into the required form. There are various types of functions such as linear, polynomial, and radial basis functions (RBF). Polynomials and RBF are useful for non-linear hyperplanes. Polynomial and RBF kernels compute the separation line in the higher dimension.
- **Regularization**: Regularization parameter in python's Scikit-learn C parameter used to maintain regularization. Here C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. This is how we can control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane.
- **Gamma**: A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes overfitting.

For choosing these hyperparameters we used GridSearchCV with 10-fold cross validation which gave us the following results:

```
classifier=SVC()

parameters = [{'C': [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7,
                     2.0,1, 10, 100, 1000], 'kernel': ['linear']},

              {'C': [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7,
                     2.0,1, 10, 100, 1000], 'kernel': ['rbf'],
               'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},

              {'C': [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7,
                     2.0,1, 10, 100, 1000], 'kernel': ['poly']}]
kfold = KFold(n_splits=10, random_state=21,shuffle=True)
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = kfold,
                           n_jobs = -1)
grid_search = grid_search.fit(X_train_scaled, y_train)
print(grid_search.best_params_)
```

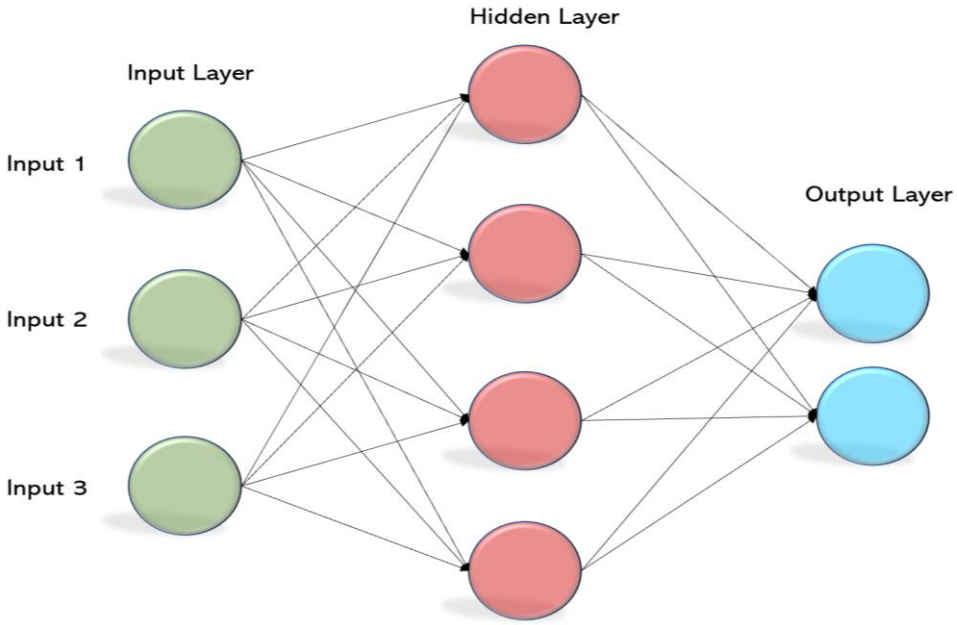
```
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
TP Rate :  0.9968152866242038
TN Rate :  0.9954441913439636
FP Rate :  0.006349206349206349
FN Rate :  0.0031847133757961785
Success Rate:  0.9960159362549801
Misclassifiaction Rate:  0.00398406374501992
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       439
         1.0       0.99      1.00      1.00       314

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753

SVC Test Accuracy :  99.60159362549801 %
Training Accuracy:  99.7341309405118 %
```

5. Building an Artificial Neural Network (ANN)

The idea of ANNs is based on the belief that working of the human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites. The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from the external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to another neuron to handle the issue or does not send it forward. Artificial neural networks (ANNs) are composed of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has

an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. When all weights are trained, the neural network can be utilized to predict the class or a quantity, if there should arise an occurrence of regression of a new input data point. With neural networks, extremely complex models can be trained and they can be utilized as a kind of black box.



ANN Architecture and Design Parameters:

(i) Number of layers used: 3 consisting of one input layer, one hidden layer and one output layer. Used hit and trial method to calculate the number of layers required

(ii) Number of neurons in each layer: Decided the number of neurons keeping in mind the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

ANN Training Protocols and Hyperparameters:

(i) Epochs: An epoch is referred to as 1 step of weight change. 1 epoch gets completed only if all the samples are fed to the network. We have used 50 epochs, keeping in mind the size of our data. First five epochs:

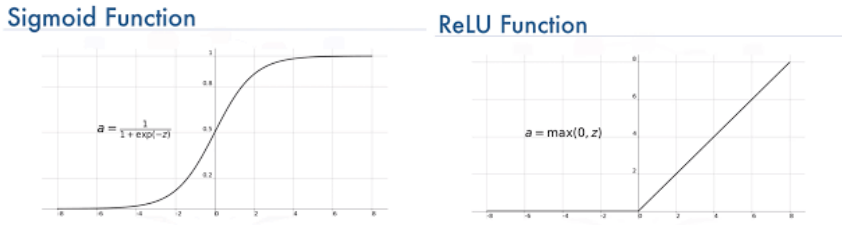
```
Epoch 1/50
38/38 [==============================] - 10s 170ms/step - loss: 0.6572 - accuracy: 0.5946 - val_loss: 0.5019 - val_accuracy: 0.
9302
Epoch 2/50
38/38 [==============================] - 0s 4ms/step - loss: 0.4892 - accuracy: 0.8752 - val_loss: 0.3685 - val_accuracy: 0.945
2
Epoch 3/50
38/38 [==============================] - 0s 3ms/step - loss: 0.3805 - accuracy: 0.9159 - val_loss: 0.2740 - val_accuracy: 0.960
1
Epoch 4/50
38/38 [==============================] - 0s 5ms/step - loss: 0.3056 - accuracy: 0.9369 - val_loss: 0.2106 - val_accuracy: 0.963
5
Epoch 5/50
38/38 [==============================] - 0s 4ms/step - loss: 0.2430 - accuracy: 0.9492 - val_loss: 0.1685 - val_accuracy: 0.965
1
```

(ii) Batch Method: Mini Batch SGD method is used as we have passed 64 samples per epoch to the network.

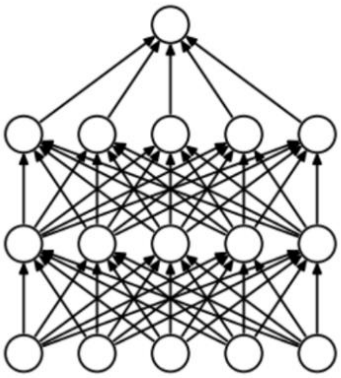
(iii) Learning Rate: refers to the step of backpropagation, when parameters are updated according to an optimization function. We have tested our model for different learning rates ranging from 0.0001 to 1 and saw which value gave the best result in terms of minimizing loss and maximizing accuracy.

(iv) Optimizer: Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function. We have used Adam optimizer as it is the most efficient and widely accepted.

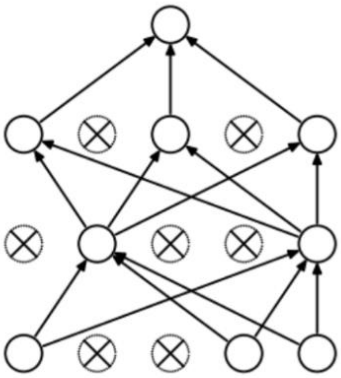
(v) Activation function: it is the function through which we pass our weighted sum, in order to have a significant output, namely as a vector of probability or a 0–1 output. We have used the 'Relu' activation function for our input and hidden layer and 'sigmoid' function for the output layer.



(vi) Dropout: refers to ignoring units (i.e., neurons) during the training phase of certain set of neurons which is chosen at random. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. Therefore, we use a dropout layer to prevent overfitting.

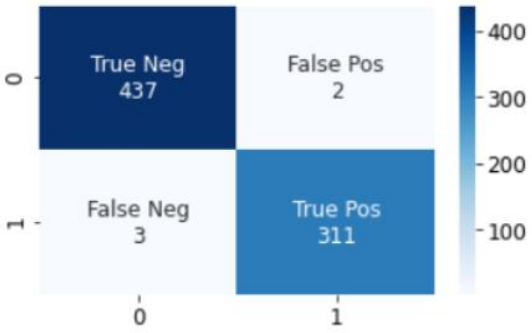


(a) Standard Neural Net

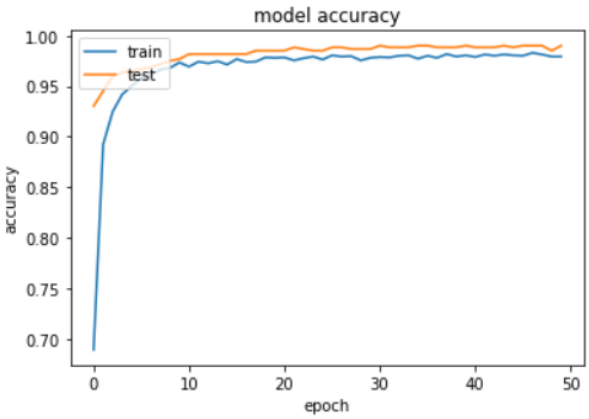


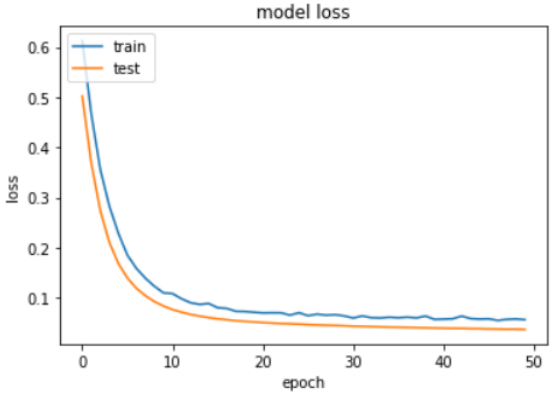
(b) After applying dropout.

```
TP Rate :  0.9904458598726115
TN Rate :  0.9954441913439636
FP Rate :  0.006389776357827476
FN Rate :  0.009554140127388535
Success Rate:  0.9933598937583001
Misclassifiaction Rate:  0.006640106241699867
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99       439
         1.0       0.99      0.99      0.99       314

    accuracy                           0.99       753
   macro avg       0.99      0.99      0.99       753
weighted avg       0.99      0.99      0.99       753

Neural Network Test Accuracy :  99.33598937583001 %
```



For our Neural Network we can see how the train and test accuracy and loss change with the increase in number of epochs:





6. Decision Trees

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree.

Hyperparameters:

- **Criterion:** The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- **Max Depth:** The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **Min Sample Split:** the minimum number of samples required to split an internal node.

Here too, using GridSearchCV and 10-fold cross validation or Pruning of Decision Tree we acquired the mentioned hyperparameters as following:

```
param_dist = {
    "criterion" : ["gini" , "entropy"] ,
    "max_depth" : [1,2,3,4,5,6,7,8,9,10,None] ,
    "min_samples_split" : [10,20,50,80,100,200,300,500]
}
kfold = KFold(n_splits=10, random_state=21,shuffle=True)
grid_dt = GridSearchCV(tree , param_grid = param_dist , cv=kfold , n_jobs=-1)
grid_dt.fit(X_train_scaled,y_train)
```

```
GridSearchCV(cv=KFold(n_splits=10, random_state=21, shuffle=True),
             estimator=DecisionTreeClassifier(random_state=1), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, None],
                         'min_samples_split': [10, 20, 50, 80, 100, 200, 300,
                                               500]})
```

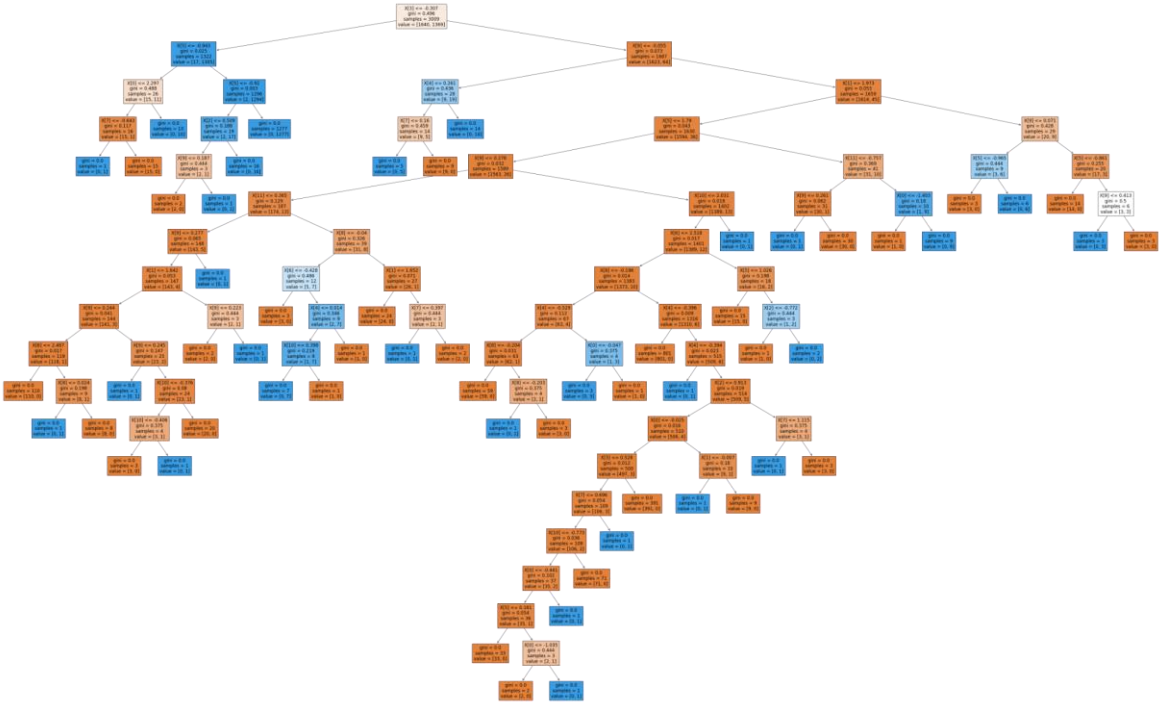
```
grid_dt.best_params_
```

```
{'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 10}
```

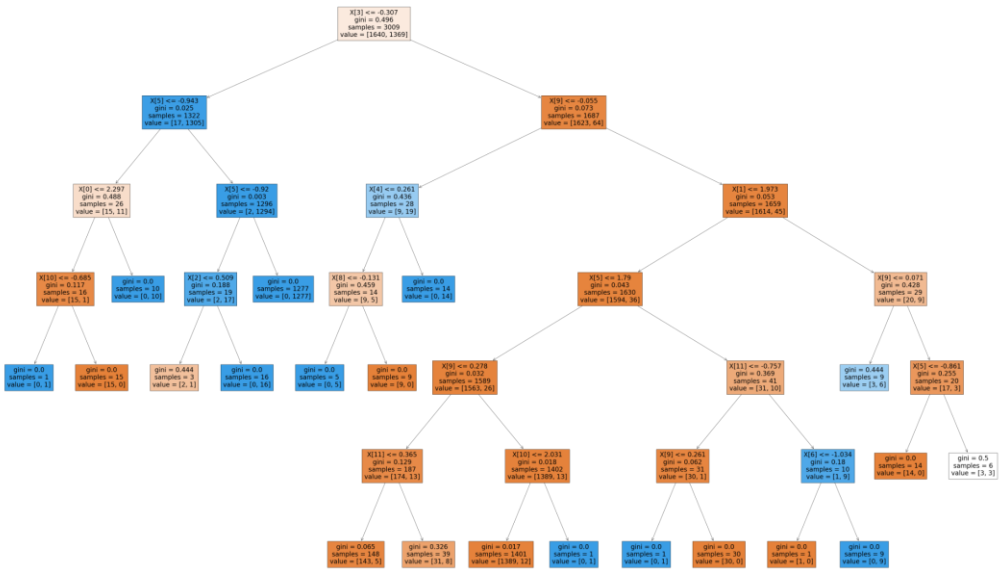
```
TP Rate :  0.9968152866242038
TN Rate :  0.9908883826879271
FP Rate :  0.012618296529968454
FN Rate :  0.0031847133757961785
Success Rate:  0.9933598937583001
Misclassifiaction Rate:  0.006640106241699867
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99       439
         1.0       0.99      1.00      0.99       314

    accuracy                           0.99       753
   macro avg       0.99      0.99      0.99       753
weighted avg       0.99      0.99      0.99       753

Decision Tree Test Accuracy :  99.33598937583001 %
Training Accuracy:  98.96975739448321 %
```

Without any hyperparameter tuning, by default, the Decision Tree function doesn't perform any pruning and allows the tree to grow as much as it can. And thus, the chance of overfitting increases.

Decision Tree without Pruning / Hyperparameter Optimization:



Decision Tree with Pruning:



7. Tree Ensemble Techniques

a) Bagging

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples (or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

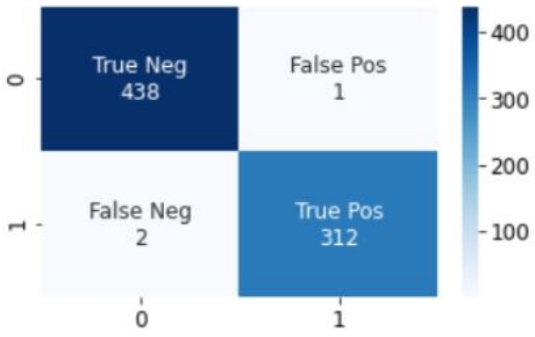
Hyperparameters:

- **n_estimators**: The number of base estimators in the ensemble.
- **max_samples**: The number of samples to draw from X to train each base estimator.
- **bootstrap:** Whether samples are drawn with replacement. If False, sampling without replacement is performed.
- **max_features:** The number of features to draw from X to train each base estimator.

Here too, using GridSearchCV and 10-fold cross validation Bagging Classifier we acquired the mentioned hyperparameters as following:

```
In [97]: parameters = {
             'n_estimators': [50,100,500],
             'max_samples': [0.1,0.2,0.4,0.7,1.0],
             'bootstrap' : [True,False],
             'max_features' : [0.1,0.2,0.4,0.7,1.0]
             }
         kfold = KFold(n_splits=10, random_state=21,shuffle=True)
         search = GridSearchCV(bag, parameters, cv=kfold,verbose = 1 , n_jobs=-1)
         search.fit(X_train_scaled,y_train)
         search.best_params_

         Fitting 10 folds for each of 150 candidates, totalling 1500 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   22.9s
         [Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   56.7s
         [Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  2.0min
         [Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  4.2min
         [Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed:  4.4min finished

Out[97]: {'bootstrap': True,
          'max_features': 0.4,
          'max_samples': 1.0,
          'n_estimators': 500}
```

```
OOB Score with hyperparameter tuning :  98.60418743768695 %

TP Rate :  0.9936305732484076
TN Rate :  0.9977220956719818
FP Rate :  0.00319488817891373 8
FN Rate :  0.006369426751592357
Success Rate:  0.9960159362549801
Misclassifiaction Rate:  0.00398406374501992
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       439
         1.0       1.00      0.99      1.00       314

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753

Bagging Test Accuracy :  99.60159362549801 %
Training Accuracy:  100.0 %
```



b) Random Forest Method

To put it simply, random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity. It does not suffer from the overfitting problem. The root node is considered to have a depth of 0.For each tree in the forest, the depth was varied and along with it, the number of trees in the forest are also varied so as to achieve the maximum training accuracy and best parameters. The model is then trained according to the best parameter and the testing accuracy is calculated.

Hyperparameters:

- **n_estimators:** The number of trees in the forest.
- **Criterion:** The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- **Bootstrap:** Whether bootstrap samples are used when building trees.
- **max_depth:** The maximum depth of the tree.
- **max_features:** The number of features to consider when looking for the best split.
- **min_sample_split:** The minimum number of samples required to split an internal node.

Again, using **GridSearchCV** and **10-fold cross validation** Random Forest Classifier, we acquired the mentioned hyperparameters as following:

```
Out[118]: GridSearchCV(cv=5,
                       estimator=RandomForestClassifier(criterion='entropy',
                                                        max_features=0.5,
                                                        min_samples_split=10,
                                                        n_estimators=50),
                       n_jobs=-1,
                       param_grid={'bootstrap': [True, False],
                                   'criterion': ['gini', 'entropy'],
                                   'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, None],
                                   'max_features': [0.1, 0.2, 0.4, 0.7, 1.0],
                                   'min_samples_split': [10, 20, 50, 80, 100, 200, 300,
                                                         500],
                                   'n_estimators': [50, 100, 200]},
                       verbose=1)
```

```
In [119]: search_rf.best_params_

Out[119]: {'bootstrap': True,
           'criterion': 'entropy',
           'max_depth': 9,
           'max_features': 0.7,
           'min_samples_split': 10,
           'n_estimators': 100}
```

```
TP Rate :  0.9968152866242038
TN Rate :  1.0
FP Rate :  0.0
FN Rate :  0.0031847133757961785
Success Rate:  0.99867197875166
Misclassifiaction Rate:  0.0013280212483399733
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       439
         1.0       1.00      1.00      1.00       314

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753

OOB Score with hyperparameter tuning :  98.47125290794284 %
Random Forest Test Accuracy :  99.867197875166 %
Training Accuracy:  99.6011964107677 %
```

c) Boosting: Adaboost

It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances. Boosting is used to reduce bias as well as the variance for supervised learning. It works on the principle where learners are grown sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones. Adaboost algorithm also works on the same principle as boosting, but there is a slight difference in working.

Hyperparameters:

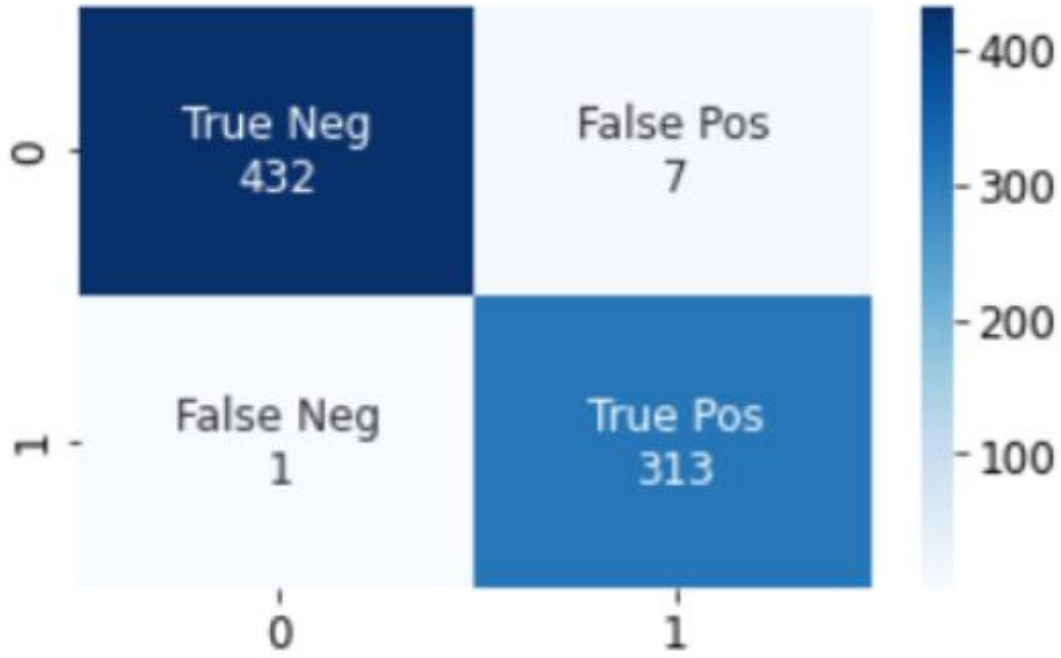
- **n_estimators:** The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.
- **learning rate:** Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the learning_rate and n_estimators parameters.
- **random_state:** Controls the random seed given at each base_estimator at each boosting iteration. Thus, it is only used when base_estimator exposes a random_state.

Again, by using **GridSearchCV** and **10-fold cross validation** Adaboost Classifier, we acquired the mentioned hyperparameters as following:

```
In [105]: ada_params = {
              'n_estimators': [50,100,200],
              'learning_rate' : [0.1,0.01,0.001,0.2,0.3,0.4,0.5],
              'random_state' : [0,20,40]
              }

          ada_cv = RandomizedSearchCV(ada, ada_params, cv=kfold , verbose=1,n_jobs=-1)
          ada_cv.fit(X_train_scaled,y_train)
          print(ada_cv.best_params_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    5.0s
{'random_state': 20, 'n_estimators': 50, 'learning_rate': 0.2}
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    5.9s finished
```

```
TP Rate :  0.9968152866242038
TN Rate :  0.9840546697038725
FP Rate :  0.021875
FN Rate :  0.0031847133757961785
Success Rate:  0.9893758300132802
Misclassifiaction Rate:  0.010624169986719787
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       439
         1.0       0.98      1.00      0.99       314

    accuracy                           0.99       753
   macro avg       0.99      0.99      0.99       753
weighted avg       0.99      0.99      0.99       753

Random Forest Test Accuracy :  98.93758300132802 %
Training Accuracy:  100.0 %
```

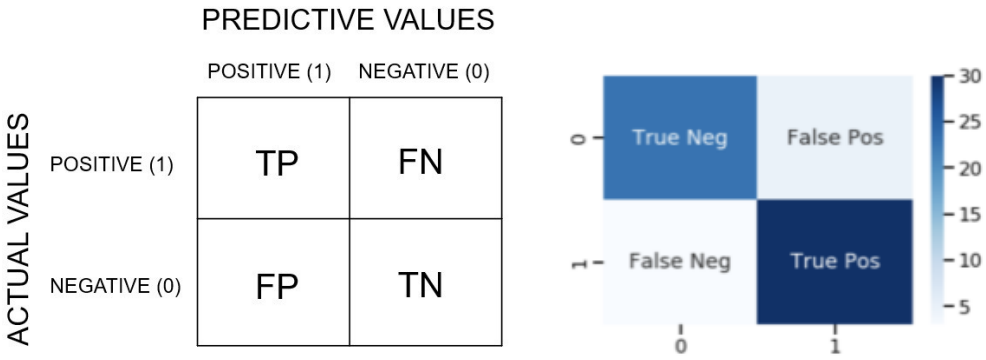


Results and Analysis:

Performance Metrics Considered:

1. **Confusion matrix (tp rate, fp rate, fn rate, tn rate)**

A confusion matrix is a technique for summarizing the performance of a classification algorithm. Calculating a confusion matrix can give a better idea of what the classification model is getting right and what types of errors it is making.



TP (True Positives): These are the occurrences where both the predictive and actual class is true.

TN (True Negatives): These are the occurrences where both the predicted class and actual class is false.

FP (False Positives): False positives are the occurrences where the predicted class is true, while the actual class is false.

FN (False Negatives): These are occurrences where the predicted class is false but actual class is true.

Accuracy: An evaluation criterion, accuracy is the fraction of prediction. It determines the number of correct predictions over the total number of predictions made by the model. The formula of accuracy is: (TP+TN)/ (TP+TN +FP+FN).

Sensitivity/TP Rate/Precision: Classifier's performance to spot positive results is related by Sensitivity. Specificity is calculated as follows: TP/(TP+FN).

Specificity/TN Rate: Classifier's performance to spot negative results is related by Specificity. It is given by: TN/(TN+FP).

Recall: It is the number of correctly classified +ve examples divided by the total number of +ve examples in the dataset. It is given by: TP/ (TP + FP)

F1 score: F1-score is the harmonic mean of precision and recall = 2(Precision)(Recall)/ (Precision + Recall)

2. **Misclassification rate:**

(FP + FN)/ (TP +TN + FP +FN), it is the error caused by misclassification of samples.

3. **ROC Curve:**

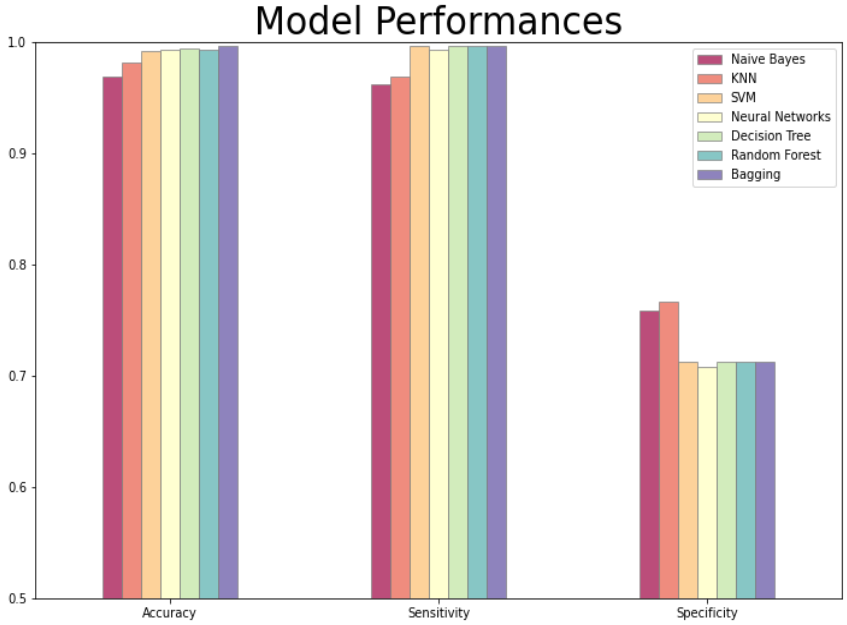
ROC stands for Receiver Operating Characteristics. ROC graph plots TP rate on the y axis and FP rate on the x axis. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. Classifiers in the upper left corner of ROC graphs are preferred as both their sensitivity and specificity are high as a classifier having the highest sensitivity and specificity is the best model for that dataset. The more the area under the curve is close to 1, the better is the algorithm for which the ROC curve is plotted.

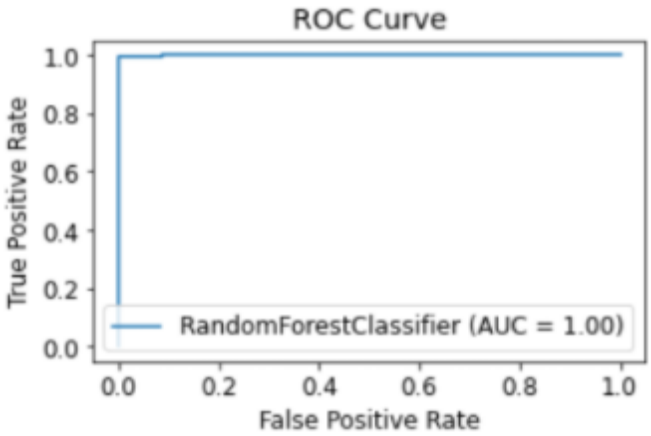
4. **OOB Score and Log Loss:**

Out-of-bag (OOB) error, on an average for a dataset D, if there are total N samples, 2N/3 samples will be independent and included in ensemble techniques. Therefore N/3 samples are left untouched, these samples can then be used for testing and help in avoiding overfitting. The testing accuracy on these samples is called OOB score.

Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value. We have used log-loss/binary cross entropy for minimizing the loss in our neural network.

Comparison of algorithms used based on accuracy, sensitivity and specificity.





Also, area under ROC curve was close to 1 for all algorithms, but was closest for Random Forest Classifier

To avoid overfitting in the various algorithms applied we opted the following techniques:

1) Hyperparameter tuning using GridSearchCV to give the best parameters.

2) K Fold Cross Validation to test the model's ability to predict new data that was not used in estimating it.

3) Regularization significantly reduces the variance of the model, without substantial increase in its bias.

For the comparison of different machine learning algorithms applied we will take into account the following parameters:

Algorithm	Test Accuracy	Sensitivity	Specificity	FP rate	FN rate
Naive Bayes	99.203%	0.984	0.997	0.003	0.015
K Nearest Neighbours	99.468%	0.987	1.000	0.000	0.012
Logistic Regression	98.804%	0.993	0.984	0.021	0.006
Support Vector Classifier	99.601%	0.997	0.995	0.006	0.003
Neural Network	99.335%	0.990	0.995	0.006	0.009
Decision Tree	99.335%	0.997	0.990	0.010	0.003
Bagging Classifier	99.601%	0.994	0.997	0.003	0.006
Random Forest	99.867%	0.997	1.000	0.000	0.003
Adaboost	98.93%	0.997	0.984	0.021	0.003

Sensitivity allows us to correctly identify those with brain tumor, whereas Specificity helps us correctly identify those without it. Both should be considered important while evaluating a model as they give us an idea of how reliable the model is.

FP rate helps in determining the cases which don't have brain tumor but have been diagnosed with it whereas FN rate helps in determining cases which have brain tumor but have been diagnosed negative. FN rate should be really low and close to 0 as a false negative diagnosis can be really serious. FP rate should also be low as possible as it can lead to unnecessary treatments. However, in brain tumor prediction. the false negative rate can be given an edge above the false positive rate.

Conclusion:

Our report evaluates the Brain Tumor Dataset using various machine learning algorithms. First the data was clearly reviewed and visualized to transform it to give optimized results. Afterwards, all the models were properly investigated and the best model was selected based on various performance metrics.

All models achieved really good test accuracies. However, sensitivity, specificity, FP rate and FN rate were the most important factors in deciding the optimal model.

If we consider the average of specificity and sensitivity, the average of almost all algorithms was more than 99%.

If we consider FP rate and FN rate, Random Forest Classifier gave the best results with a remarkable FP rate of 0 and a FN rate of 0.003.

Also, for the Random Forest classifier, the average of specificity and sensitivity was 0.9985, being the highest among all.

Taking into account all the performance metrics, Random Forest Classifier gave the optimal results among all the models. It yielded a really high test accuracy of 99.867%, the highest among all along with the best average of sensitivity and specificity, and the lowest FP and FN rate of 0 and 0.003 respectively.

Thus, it can be concluded that Random Forest Classifier is the most optimal and efficient model for our dataset as it achieved the best results when compared to all the other models. It also ensured that no overfitting occurred in the model as all the hyper-parameters were properly tuned.