Malicious Traffic Detector on a Network using Machine Learning Techniques

Team Members:

Rashi Maheshwari- 19BDS0006
Vinay Kumar Ratnala- 19BDS0124
Utpal Prajapati - 19BCE0759

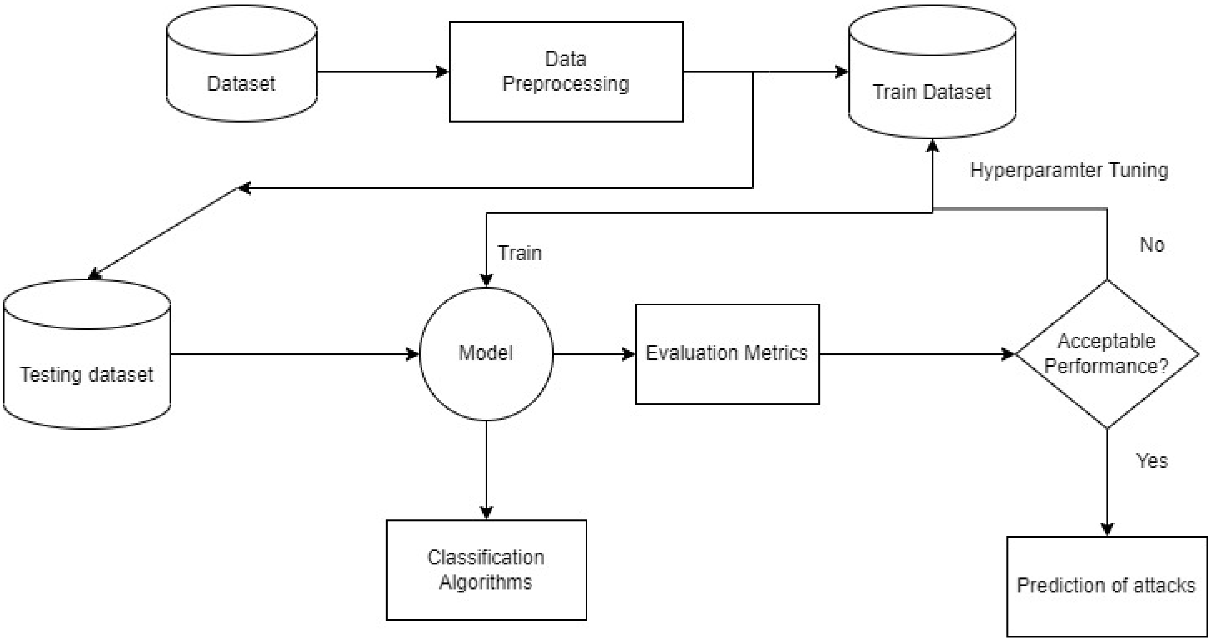
Abstract

Intrusion Detection Systems (IDS) are a type of software application which cater to the responsibility of detecting violation of policies and other malicious activities by analysis through monitoring a network or some systems. With the pandemic excessively aggravating the usage of Internet and computer systems, the number of attacks is increasing immensely, threats are being created to compromise systems or networks in a rapid manner. The CIA Triad- Confidentiality, Integrity, Availability, is severely affected as a result of Intrusions. Hence, Intrusion Detection Systems should be adopted to be aware of miscreants and prevent potential attacks. This will help protect important data and services.In this Project, we aim at developing a Network Intrusion Detection System using various Machine Learning Techniques: Logistic Regression, Decision Tree, Random Forest, Multi-Layer Perceptron, Support Vector Machine and Naïve Bayes. Analysis of network parameters such as srv_count, flag etc. are accounted for and with the help of such values the prediction is made that an attack is happening or not. All the techniques are taken into consideration and compared.

Problem Statement

Cybersecurity has become very prominent now due to our increasing dependence on technology. There has been a rise in attacks now more than ever due to a faster network, better technology and more use of the cloud. In the today’s world where everyone is connected to the Internet and everything is being done online and there is a lot of communication between the devices where the data packets are sent from one PC to another, several types of attacks have evolved which not only can harm the private data of an organisation but can also lead to the disruption of services like in the case of Denial Of Service attack. So a system which can detect such malicious packets and protect the system from loss is the need of the hour. Anomaly traffic detectors are must for any company or organisation dealing with general to protect its internal systems and data from the attacker. We plan on making a network intrusion detection system which will decide if it will allow the packets to pass or not after analysing the traffic. Our model is going to help in preventing the 4 types of attacks which are DoS, Probe, R2L, and U2R by denying the entry to the malicious packets. This would help in keeping the network safer.

Architecture Flowchart/Block Diagram:



Proposed Methods/Methodology

The primary goal is to design a plan for detecting intrusions within the system with the least possible number of features within the dataset. Based on the data from previous papers published, we can tell that only a subdivision of features in the dataset are derivative to the Intrusion Detection System. We have to cut back the dimensionality of the dataset to build an improved classifier in a justifiable amount of time. Our approach consists of 4 fundamental stages:

1. Data collection:

We will use the NSL KDD dataset provided by the University of New Brunswick (unb.ca/cic/datasets/nsl.html). The dataset is given in text format and has to be converted to csv format for processing. Each record has a total of 41 attributes for identifying the various features of the network packets and a label is named to every packet. The 42nd attribute unfolds data about labels of each record. They’re categorised as 1 for ordinary class and 4 attack-type classes which are DoS, Probe, R2L, and U2R.

2. Data Preprocessing:

In this step we will check for missing values in the dataset and drop unnecessary records. The 42nd column of the dataset has different kinds of attack names.we will classify them into 4 kinds namely DOS,Probe,R2L and U2R as 1,2,3,4 and 0 as normal case.We will use label encoders to deal with categorical columns.We will use standard scalers to scale the numeric columns to a certain range.

3. Data Visualisation and Analysis:

After preprocessing the data, we will then proceed to see how the target value was distributed.
Using python inbuilt libraries such as matplotlib and seaborn we will plot distributions of attacks and normal cases.we will compare input features to output/target column to analyze the relationship between them. We will cross validation technique and split the dataset into train and test sets randomly.

4. Building a classifier:

We will use the following algorithms for classification:

1. Random Forest
2. Decision tree
3. Support vector machine
4. Naïve bayes
5. Logistic regression
6. Multi-Layer Perceptron
7. Xg Boost
8. AdaBoost

4. Evaluation

We will perform evaluation of all the models using

1. 80-20 train test split
2. K-cross validation

Then we will calculate the accuracy, precision, recall and F1-score.

Literature Review:

References	Method	Result	Limitations
[1]	This study uses several machine learning algorithms such as j48, MLP, Random forest classifier to build an intrusion detection system to predict four kinds of attacks(DOS,U2R,R2L,Probe)	Results of all ML algorithms on the datasets are compared, out of which j48 outperformed all other algorithms with an accuracy of 93%.	The major limitation of this study is that no single ML algorithm is good at classifying all kinds of attacks.Decision Table has lowest false negative value where it has better accuracy than other models.Naive bayes is efficient in classifying normal cases.
[2]	In this study a new ML algorithm was proposed. The proposed algorithm consists of Correlation Attribute Evaluation (CAE) and combines with Area Under Roc Curve (AUC) metric to overcome the problem of effective feature selection for Bot-IoT detection by using a specific machine learning (ML) algorithm.	Decision tree, and Random Forest are 98.95% and 99% while Naive Bayes and SVM are 98.44% and 98.48%, which are very effective performance results with respective specificity metrics.	Due to the inappropriate feature selection, several ML models prone misclassify mostly malicious traffic flows.
[3]	In this study authors proposed ML algorithms such as SVM ,Decision Trees, Naive Bayes using different kernel functions.authors have tested models on many datasets for this study	Decision Trees with 89% performed better than Naive Bayes(73%) and SVM (70%) with RBF kernel	Since the number of attributes of the dataset is high.It is difficult for the classifier to provide timely results.
	This paper focuses on the ways to select	For minimal time Scatter algorithm was	Major limitation is that IDS should be

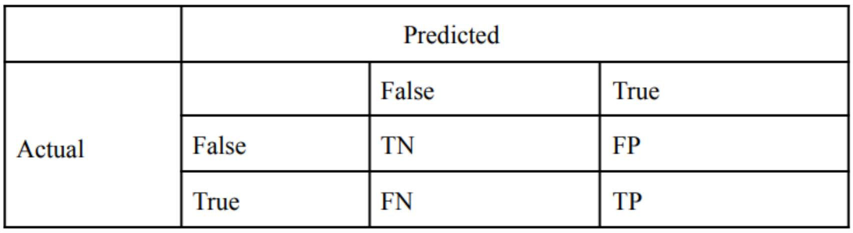
[4]	features for NIDS. They use a few feature selection techniques like Classic Guided, Meta Heuristic, Nature Inspired and Modern	proved to be better. For good results Genetic algorithms proved to be better.	updated regularly as new kinds of attacks keep evolving.
[5]	Establishing a relation on the basis of number of classes taken and accuracy seen in four models namely J48, C5.0, NB and PART Usage of MinMax normalization and Z-score standardization	Not to directly use NSL_KDDTrain+ dataset for training and testing the classifiers. MinMax normalization preferred over Z-scores on handling continuous along with categorical variables.Using J48, C5.0, Naïve Bayes and PART classifiers with attacks grouped to only one “BAD” class, shows a higher accuracy.	Not much insight into preprocessing other than MinMax and Z-Score was provided.
[6]	Recurrent Neural Network (RNN-IDS) and its comparison with J48, artificial neural network, random forest and support vector machine.NSL KDD dataset used in this study.	RNN-IDS has strong modeling ability and high accuracy in both binary and multiclass classification.	No mention of classification performance of LSTM and Bidirectional RNNs algorithm.
[7]	This paper focuses on the comparison of performance of several ML algorithms when used with different feature selection techniques.They have used 3 different feature selection techniques correlation,Information gain,PCA.They have used SVM,DT,random forests,KNN,NB	K Nearest Neighbors(KNN) has given the best accuracy of 99% followed by Decision Trees, SVM	By Observing accuracies of several algorithms when used different feature selection techniques. We can say that no particular selection technique performs well for all algorithms.

[8]	The main objective of this study is to build an IDS using ML algorithms which increases the accuracy score and decreases the False error/misclassification score.Naive Bayes and Support Vector Machine algorithms are used.	SVM outperformed NB with accuracy score of 97.2% and Misclassification rate of 2.7%	Due to excess amounts of data, false error rate will increase and accuracy of algorithms decrease.This is the major issue when the system encounters new kinds of attacks that it has never seen.
[9]	In this paper an intrusion detection tree which is a ML based security model is used and it is compared with traditional ML models.	Intrusion detection tree models achieved high accuracy compared to traditional models.	This study highlighted the limitations of using traditional models which are overcomed using Intrusion detection trees.
[10]	In this paper, we proposed EDEMA, a modular solution for early detection of network activity originating from IoT malware using ML classification techniques. Existing IoT malware were distributed among multiple categories based on their targeted software vulnerabilities. Later, steps for the ML classifier operation and the features used for classification were listed.	Accuracy, Precision, Recall and F1 scores for various classifiers were compared. KNN has given best Accuracy of 94.44% followed by Random Forest, Gaussian Naive Bayes.	If the feature distributions under the two conditions are not easily distinguishable, it may impair the detection performance.

Performance Metrics

For any classification problems the metrics that matter are the following:

Confusion Matrix



	Predicted		
Actual		False	True
	False	TN	FP
	True	FN	TP

1. Precision: TP/(TP + FP)
2. F1 score: 2*TP/(2*TP + FP + FN)
3. Recall: TP/(TP + FN)
4. ROC curve area: A graph that is drawn to plot the false positive rate along the x axis and the true positive rate along the y axis.
5. Reliability Curve: A graph to plot the observed relative frequency along the y axis versus the predicted relative frequency along the x axis.
6. Accuracy: (TP + TN) / (TP + TN + FP + FN).

Data Visualisation and Cleaning

After cleaning the data, we then proceeded to see how the target value was distributed. To further understand this we visualised the distribution using a Pie Chart. We found that normal was at 53.5% whereas attack was at 46.5%. This told us that the data was almost evenly distributed and that it wouldn't lead to any kind of bias in the future.

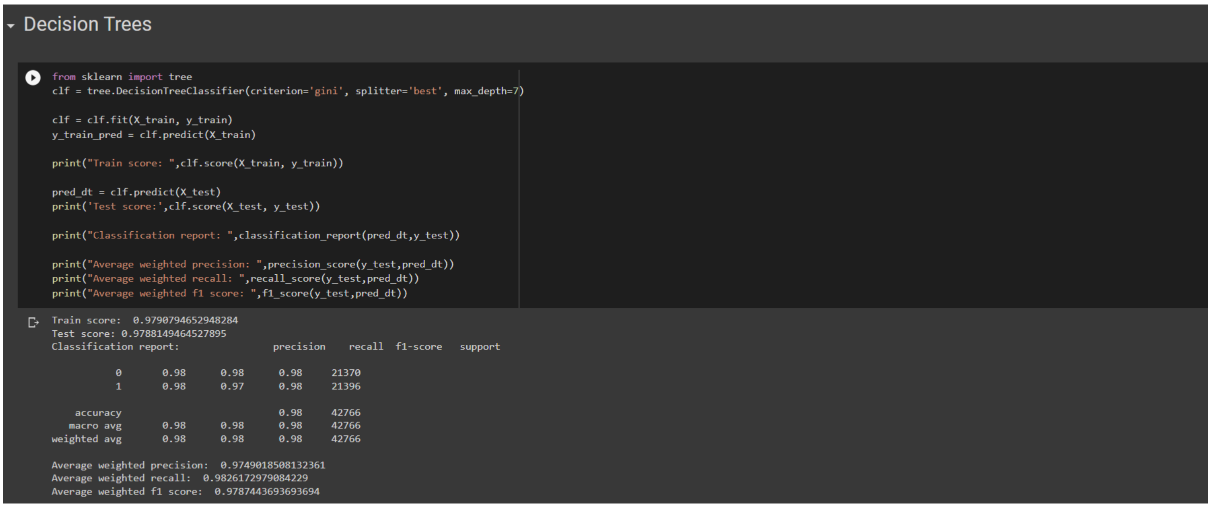
We then proceeded to see the percentage of the various attacks in the given dataset. DOS was found to be the maximum at 79.6% after which was the Probe attack at 20.2%. R2L was slightly more than U2R but both were around the 1% range and so their contribution was found not to be of much significance

Once we had a basic analysis of our data and had cleaned it removing all redundancies, we then proceeded to train our models on it.

Models:

Decision trees:

A decision tree starts with a root node and that node is compared with other attributes/features in the dataset for a perfect split. A perfect split means the number of outputs of one class comes on one side of the tree and another class on the other side of the tree. For example if a node splits the data in such a way that all the outputs which are “No” come on one side(left/right) and outputs which are “Yes” come on one side(left/right). In this way every node gets split until it arrives at a perfect split , the outcome of a perfect split becomes the leaf node of a tree. The main challenge in constructing a decision tree is selection of attributes, that is which attribute we use as root node or an internal node since we have a lot of attributes, it is kind of difficult to select . For that we have two techniques: entropy, information gain and gini index.

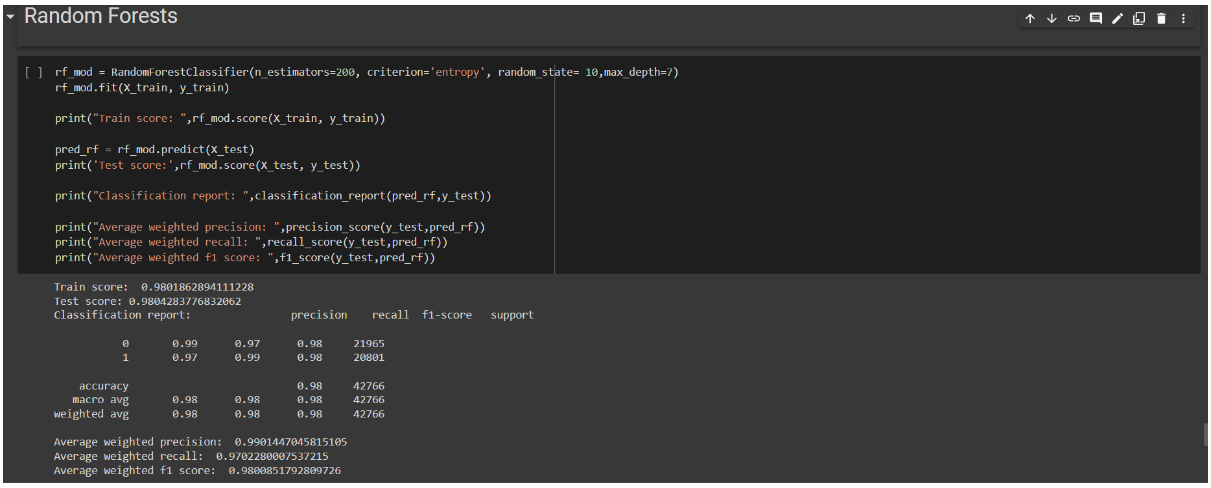


```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7)

clf = clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)

print("Train score: ",clf.score(X_train, y_train))

pred_dt = clf.predict(X_test)
print('Test score:',clf.score(X_test, y_test))

print("Classification report: ",classification_report(pred_dt,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_dt))
print("Average weighted recall: ",recall_score(y_test,pred_dt))
print("Average weighted f1 score: ",f1_score(y_test,pred_dt))
```

```
Train score:  0.9790794652948284
Test score: 0.9788149464527895
Classification report:               precision    recall  f1-score   support

           0       0.98      0.98      0.98     21370
           1       0.98      0.97      0.98     21396

    accuracy                           0.98     42766
   macro avg       0.98      0.98      0.98     42766
weighted avg       0.98      0.98      0.98     42766

Average weighted precision:  0.9749018508132361
Average weighted recall:  0.9826172979084229
Average weighted f1 score:  0.9787443693693694
```

Random Forests:

Like the name random forest suggests, the forest consists of a lot of trees. To make these models, first a bootstrapped dataset should be made and in each step we consider a subset of variables as a candidate for the root node, these steps are called bagging or bootstrap aggregating. These steps keep repeating till there is a good amount of trees. When there's a new input we take the prediction from all of these trees and the prediction that is most prominent among them will be our prediction.



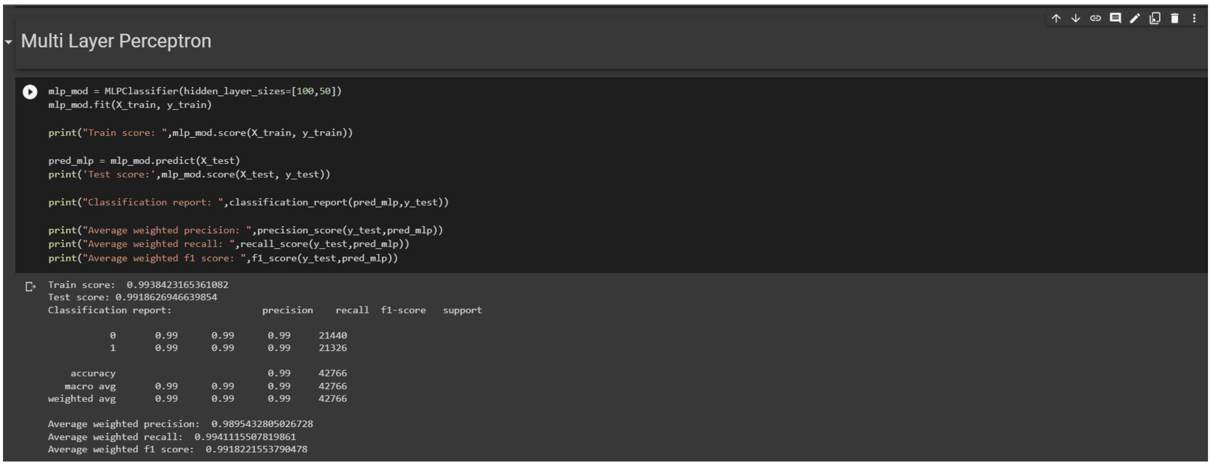
Random Forests

```
rf_mod = RandomForestClassifier(n_estimators=200, criterion='entropy', random_state= 10,max_depth=7)
rf_mod.fit(X_train, y_train)

print("Train score: ",rf_mod.score(X_train, y_train))

pred_rf = rf_mod.predict(X_test)
print('Test score:',rf_mod.score(X_test, y_test))

print("Classification report: ",classification_report(pred_rf,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_rf))
print("Average weighted recall: ",recall_score(y_test,pred_rf))
print("Average weighted f1 score: ",f1_score(y_test,pred_rf))
```

```
Train score:  0.9801862894111228
Test score: 0.9804283776832062
Classification report:                precision    recall  f1-score   support

           0       0.99      0.97      0.98     21965
           1       0.97      0.99      0.98     20801

    accuracy                           0.98     42766
   macro avg       0.98      0.98      0.98     42766
weighted avg       0.98      0.98      0.98     42766

Average weighted precision:  0.9901447045815105
Average weighted recall:  0.9702280007537215
Average weighted f1 score:  0.9800851792809726
```

Multi-Layer Perceptron:

Multi layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable.



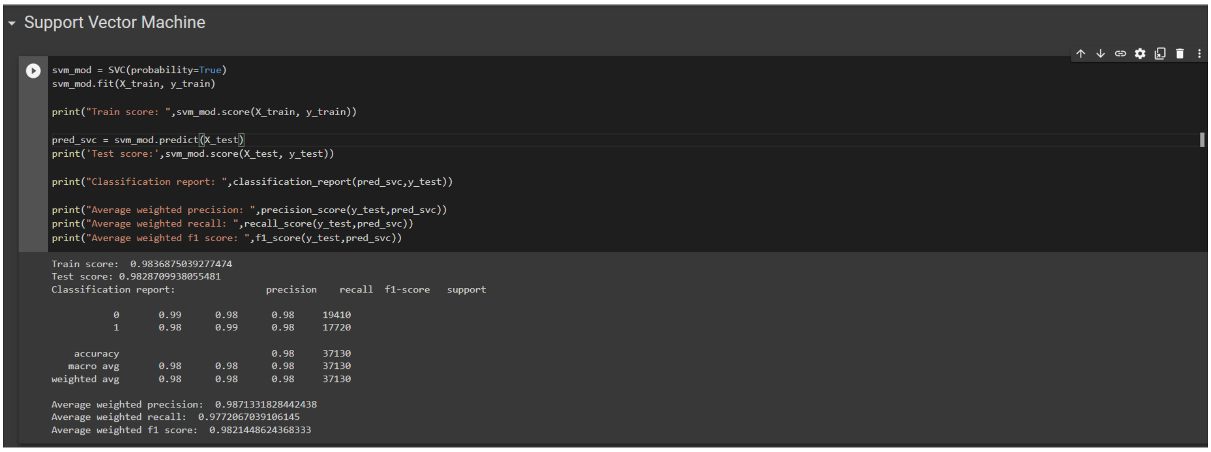
Multi Layer Perceptron

```
mlp_mod = MLPClassifier(hidden_layer_sizes=[100,50])
mlp_mod.fit(X_train, y_train)

print("Train score: ",mlp_mod.score(X_train, y_train))

pred_mlp = mlp_mod.predict(X_test)
print('Test score:',mlp_mod.score(X_test, y_test))

print("Classification report: ",classification_report(pred_mlp,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_mlp))
print("Average weighted recall: ",recall_score(y_test,pred_mlp))
print("Average weighted f1 score: ",f1_score(y_test,pred_mlp))
```

```
Train score:  0.9938423165361082
Test score: 0.9918626946639854
Classification report:                precision    recall  f1-score   support

           0       0.99      0.99      0.99     21440
           1       0.99      0.99      0.99     21326

    accuracy                           0.99     42766
   macro avg       0.99      0.99      0.99     42766
weighted avg       0.99      0.99      0.99     42766

Average weighted precision:  0.9895432805026728
Average weighted recall:  0.9941115507819861
Average weighted f1 score:  0.9918221553790478
```

Support Vector Machines:

The primary objective of SVM is to find a plane that separates an n-dimensional space into classes. It selects two data points which are nearest to each other from both classes as support vectors, the distance between the vectors and the plane which separates the classes is called margin. The main goal here is to find a plane which has maximum margin . So the plane which separates with a maximum margin is an ideal hyperplane. When we are given a new data point and asked to classify it we just see the separating plane and depending on the side it lies on we classify the point as that. We use kernel functions to make the support vector machine classify when there is no plane that divides the points properly, these kernel functions transform the points to another form which could be divisible by the plane.

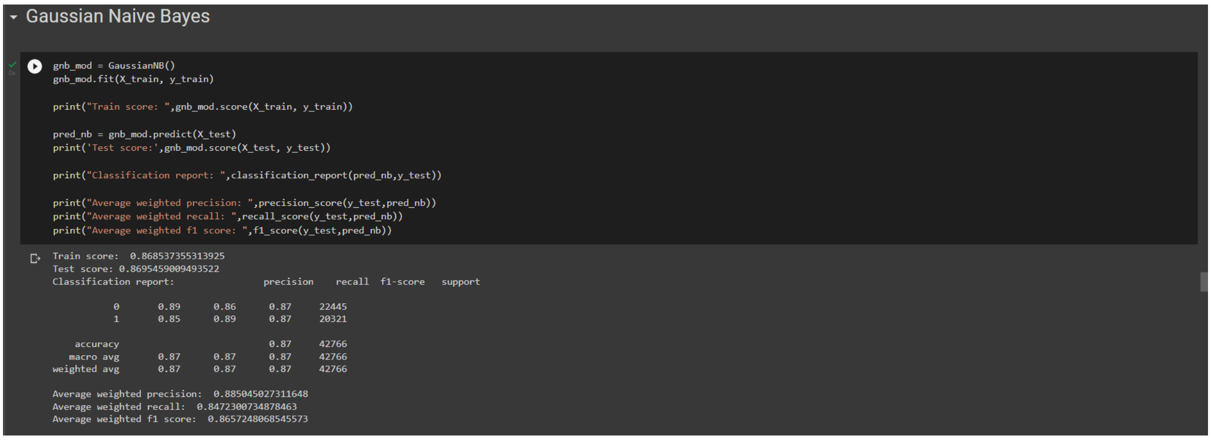


```
svm_mod = SVC(probability=True)
svm_mod.fit(X_train, y_train)

print("Train score: ",svm_mod.score(X_train, y_train))

pred_svc = svm_mod.predict(X_test)
print('Test score:',svm_mod.score(X_test, y_test))

print("Classification report: ",classification_report(pred_svc,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_svc))
print("Average weighted recall: ",recall_score(y_test,pred_svc))
print("Average weighted f1 score: ",f1_score(y_test,pred_svc))
```

```
Train score:  0.9836875039277474
Test score: 0.9828709938055481
Classification report:                precision    recall  f1-score   support

           0       0.99      0.98      0.98     19410
           1       0.98      0.99      0.98     17720

    accuracy                           0.98     37130
   macro avg       0.98      0.98      0.98     37130
weighted avg       0.98      0.98      0.98     37130

Average weighted precision:  0.9871331828442438
Average weighted recall:  0.9772067039106145
Average weighted f1 score:  0.9821448624368333
```

Naïve bayes:

Naive Bayes is the supervised learning algorithm, which is based on bayes theorem and is used for solving classification problems.It is a probabilistic classifier, which means it predicts based on the probabilities of an object



Gaussian Naive Bayes

```
gnb_mod = GaussianNB()
gnb_mod.fit(X_train, y_train)

print("Train score: ",gnb_mod.score(X_train, y_train))

pred_nb = gnb_mod.predict(X_test)
print('Test score:',gnb_mod.score(X_test, y_test))

print("Classification report: ",classification_report(pred_nb,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_nb))
print("Average weighted recall: ",recall_score(y_test,pred_nb))
print("Average weighted f1 score: ",f1_score(y_test,pred_nb))
```

```
Train score:  0.868537355313925
Test score: 0.8695459009493522
Classification report:               precision    recall  f1-score   support

           0       0.89      0.86      0.87     22445
           1       0.85      0.89      0.87     20321

    accuracy                           0.87     42766
   macro avg       0.87      0.87      0.87     42766
weighted avg       0.87      0.87      0.87     42766

Average weighted precision:  0.885045027311648
Average weighted recall:  0.8472300734878463
Average weighted f1 score:  0.8657248068545573
```

Logistic regression:

Regression is a form of statistical modeling that attempts to evaluate the relationship between one variable (termed the dependent variable) and one or more other variables (termed the independent variables). It is a form of global analysis as it only produces a single equation for the relationship. In simple words it is a model for predicting one variable from one or multiple variables. In linear regression we try to fit a linear model to data where the dependent variable is continuous:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \square \ + \beta_n X_n + \varepsilon$$

In simple words we try to find a line that best fits the points on the n-dimensional plane.

For logistic regression we try to fit a curve to data in which the dependent variable is binary, or dichotomous. In logistic regression, we seek a model like:

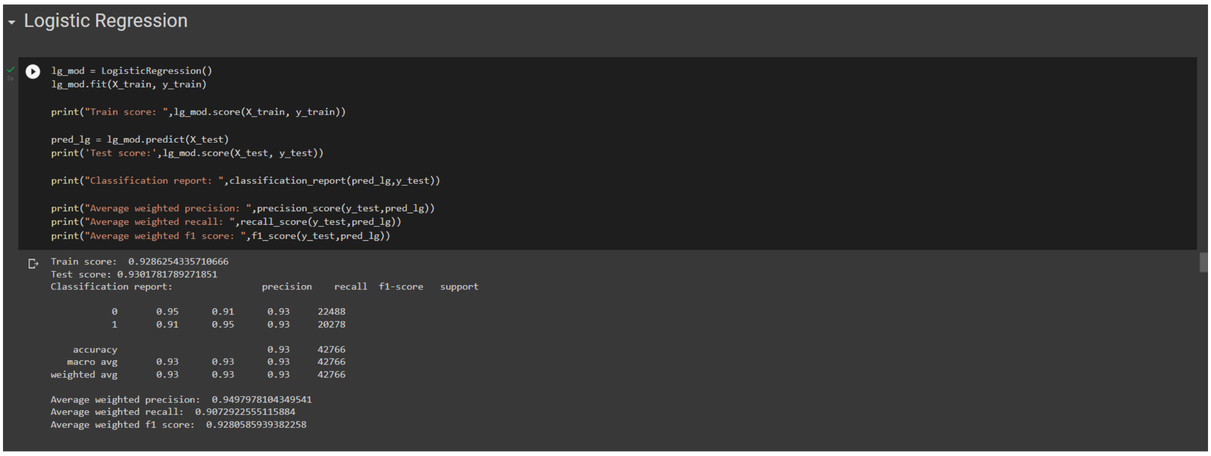
$$\operatorname{logit}(p) = \beta_0 + \beta_1 X$$

That is, the log of odds (logit) is assumed to be linearly related to the independent variable X and then the equation is solved in the same way as we solve linear regression. The probability is obtained in the following manner:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

$$\Leftrightarrow \frac{p}{1-p} = e^{\beta_0 + \beta_1 X}$$

$$\Leftrightarrow p = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$



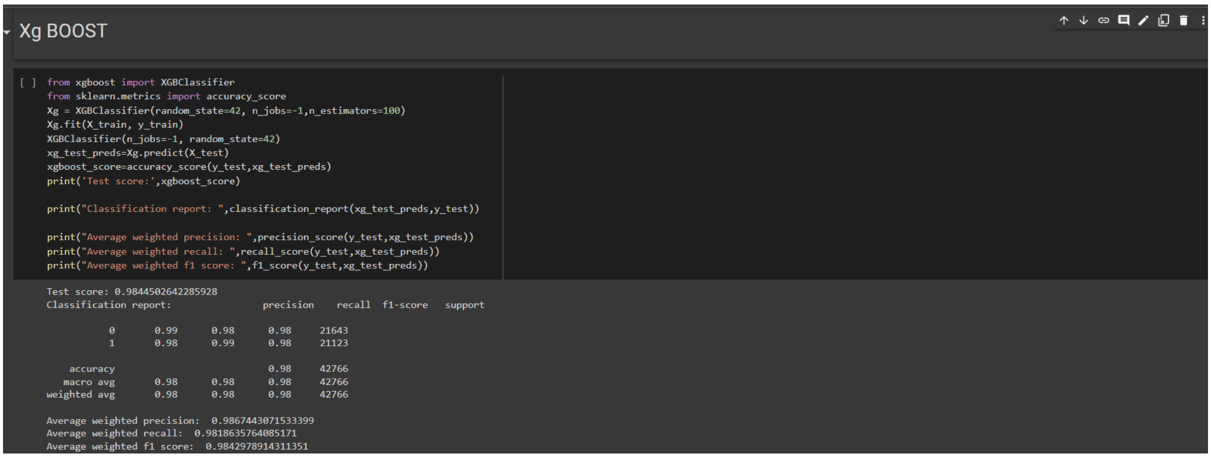
Logistic Regression

```
lg_mod = LogisticRegression()
lg_mod.fit(X_train, y_train)

print("Train score: ",lg_mod.score(X_train, y_train))

pred_lg = lg_mod.predict(X_test)
print('Test score:',lg_mod.score(X_test, y_test))

print("Classification report: ",classification_report(pred_lg,y_test))

print("Average weighted precision: ",precision_score(y_test,pred_lg))
print("Average weighted recall: ",recall_score(y_test,pred_lg))
print("Average weighted f1 score: ",f1_score(y_test,pred_lg))
```

```
Train score:  0.9286254335710666
Test score: 0.9301781789271851
Classification report:                precision    recall  f1-score   support

           0       0.95      0.91      0.93     22488
           1       0.91      0.95      0.93     20278

    accuracy                           0.93     42766
   macro avg       0.93      0.93      0.93     42766
weighted avg       0.93      0.93      0.93     42766

Average weighted precision:  0.9497978104349541
Average weighted recall:  0.9072922555115884
Average weighted f1 score:  0.9280585939382258
```

Xg-Boost:

Extreme Gradient Boosting (XGBoost) is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning framework. It is the top machine learning package for regression, classification, and ranking tasks, and it supports parallel tree boosting.



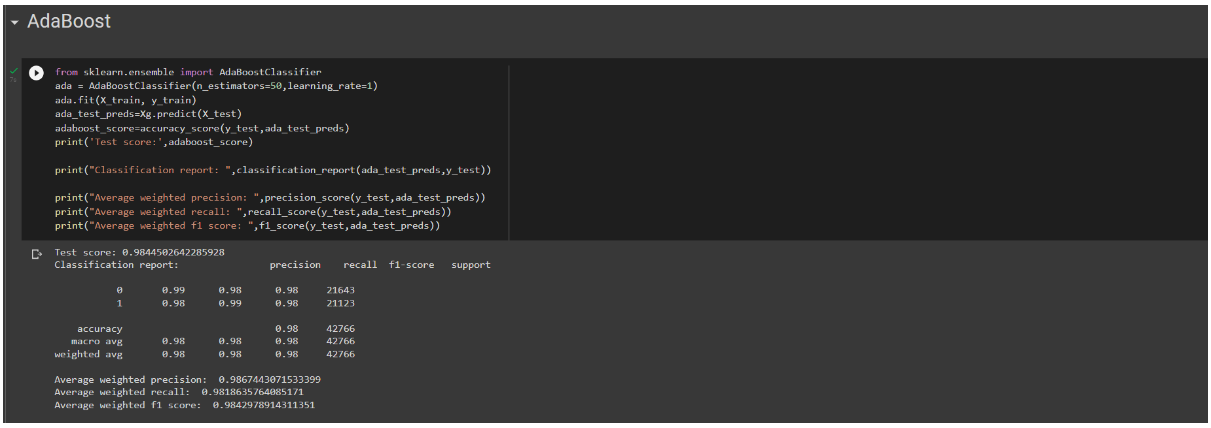
Xg BOOST

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
Xg = XGBClassifier(random_state=42, n_jobs=-1,n_estimators=100)
Xg.fit(X_train, y_train)
XGBClassifier(n_jobs=-1, random_state=42)
xg_test_preds=Xg.predict(X_test)
xgboost_score=accuracy_score(y_test,xg_test_preds)
print('Test score:',xgboost_score)

print("Classification report: ",classification_report(xg_test_preds,y_test))

print("Average weighted precision: ",precision_score(y_test,xg_test_preds))
print("Average weighted recall: ",recall_score(y_test,xg_test_preds))
print("Average weighted f1 score: ",f1_score(y_test,xg_test_preds))
```

```
Test score: 0.9844502642285928
Classification report:                precision    recall  f1-score   support

           0       0.99      0.98      0.98     21643
           1       0.98      0.99      0.98     21123

    accuracy                           0.98     42766
   macro avg       0.98      0.98      0.98     42766
weighted avg       0.98      0.98      0.98     42766

Average weighted precision:  0.9867443071533399
Average weighted recall:  0.9818635764085171
Average weighted f1 score:  0.9842978914311351
```

AdaBoost:

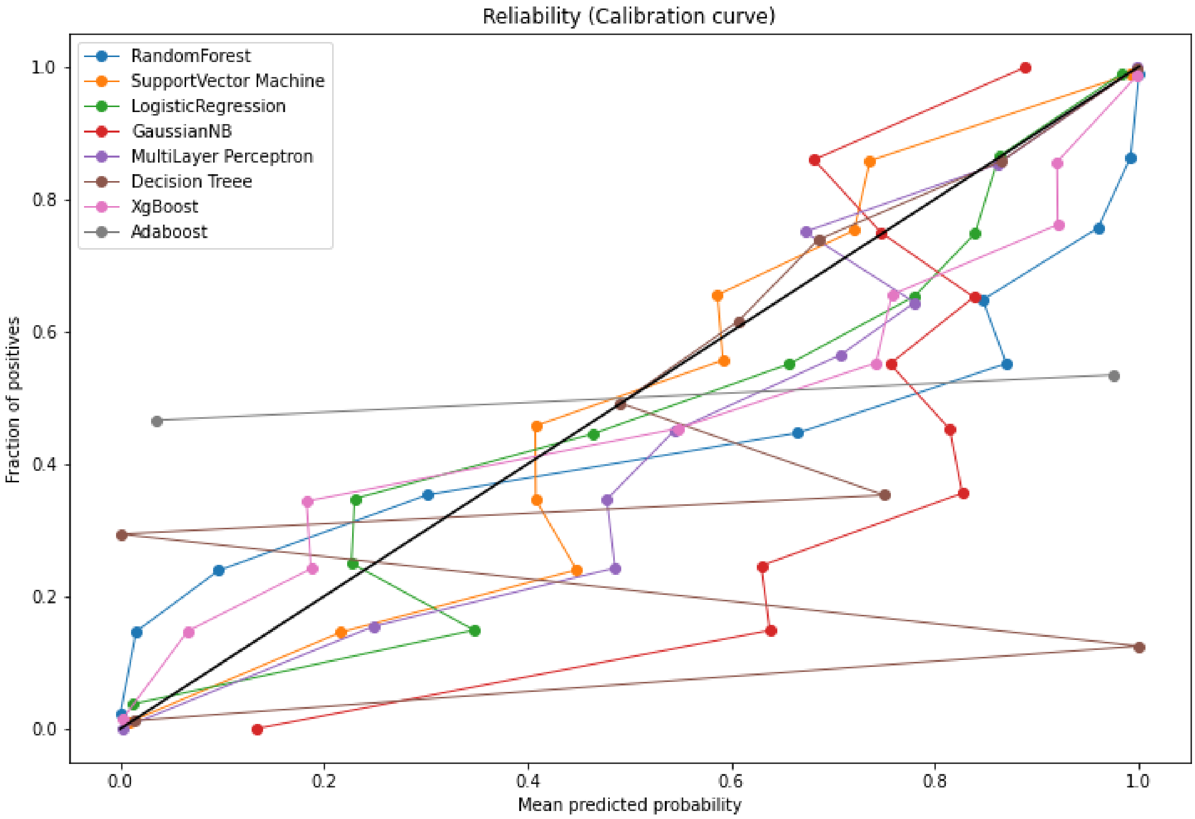
AdaBoost is a particular kind of algorithm that weighs numerous inputs using an ensemble learning strategy. Adaptive Boosting is a Machine Learning approach used as an Ensemble Method.



```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=50,learning_rate=1)
ada.fit(X_train, y_train)
ada_test_preds=Xg.predict(X_test)
adaboost_score=accuracy_score(y_test,ada_test_preds)
print('Test score:',adaboost_score)

print("Classification report: ",classification_report(ada_test_preds,y_test))

print("Average weighted precision: ",precision_score(y_test,ada_test_preds))
print("Average weighted recall: ",recall_score(y_test,ada_test_preds))
print("Average weighted f1 score: ",f1_score(y_test,ada_test_preds))
```

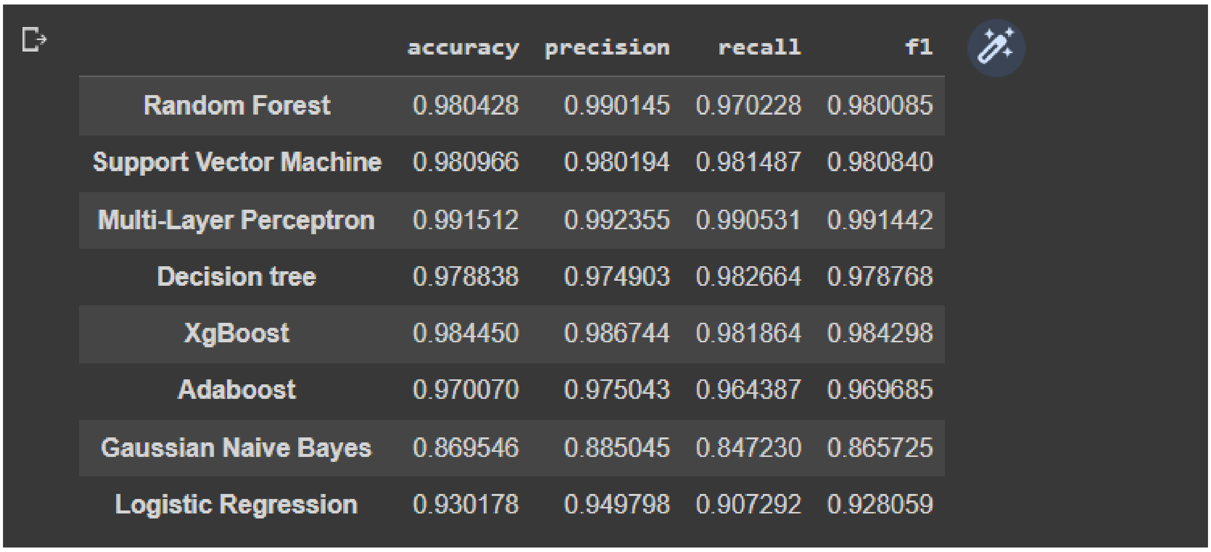
```
Test score: 0.9844502642285928
Classification report:                precision    recall  f1-score   support

           0       0.99      0.98      0.98     21643
           1       0.98      0.99      0.98     21123

    accuracy                           0.98     42766
   macro avg       0.98      0.98      0.98     42766
weighted avg       0.98      0.98      0.98     42766

Average weighted precision:  0.9867443071533399
Average weighted recall:  0.9818635764085171
Average weighted f1 score:  0.9842978914311351
```

Results:



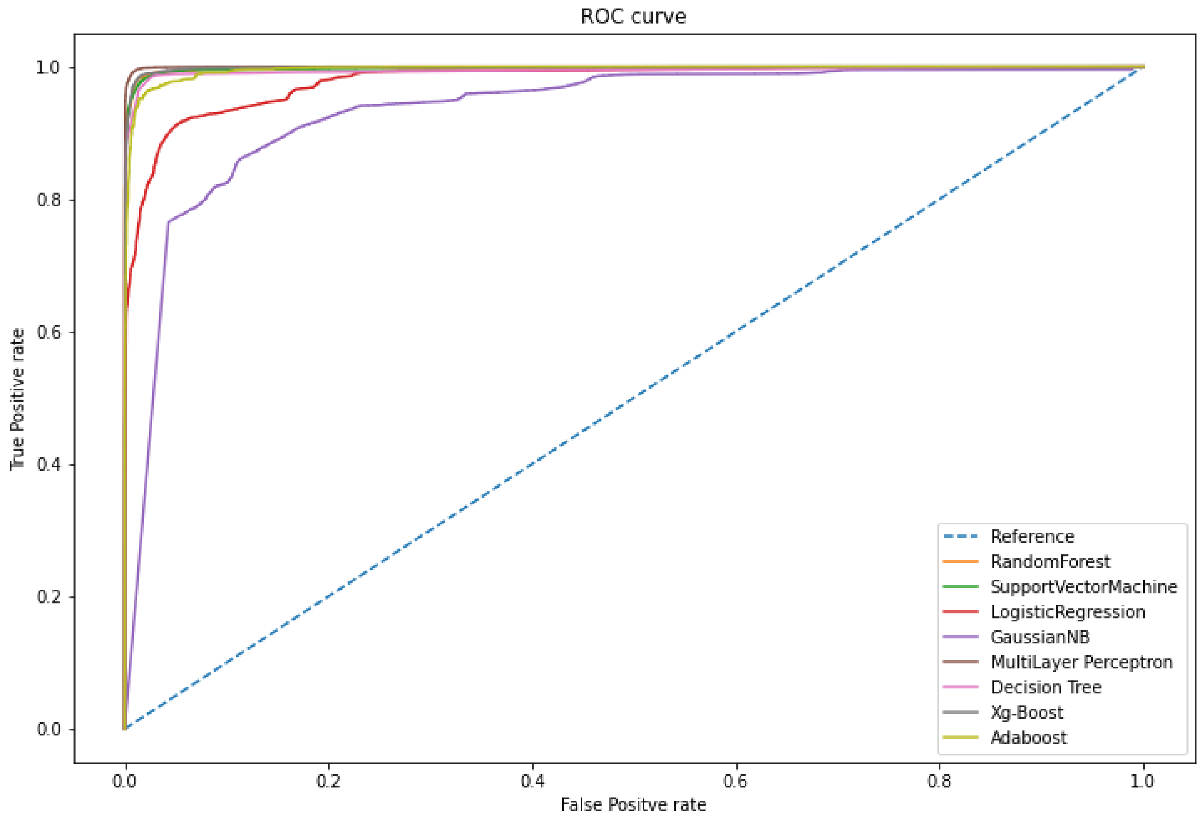
On the x-axis we have the mean predicted probability and on the y-axis we have the fraction of positives. The ideal line is the diagonal line in black.

Performance Criteria:



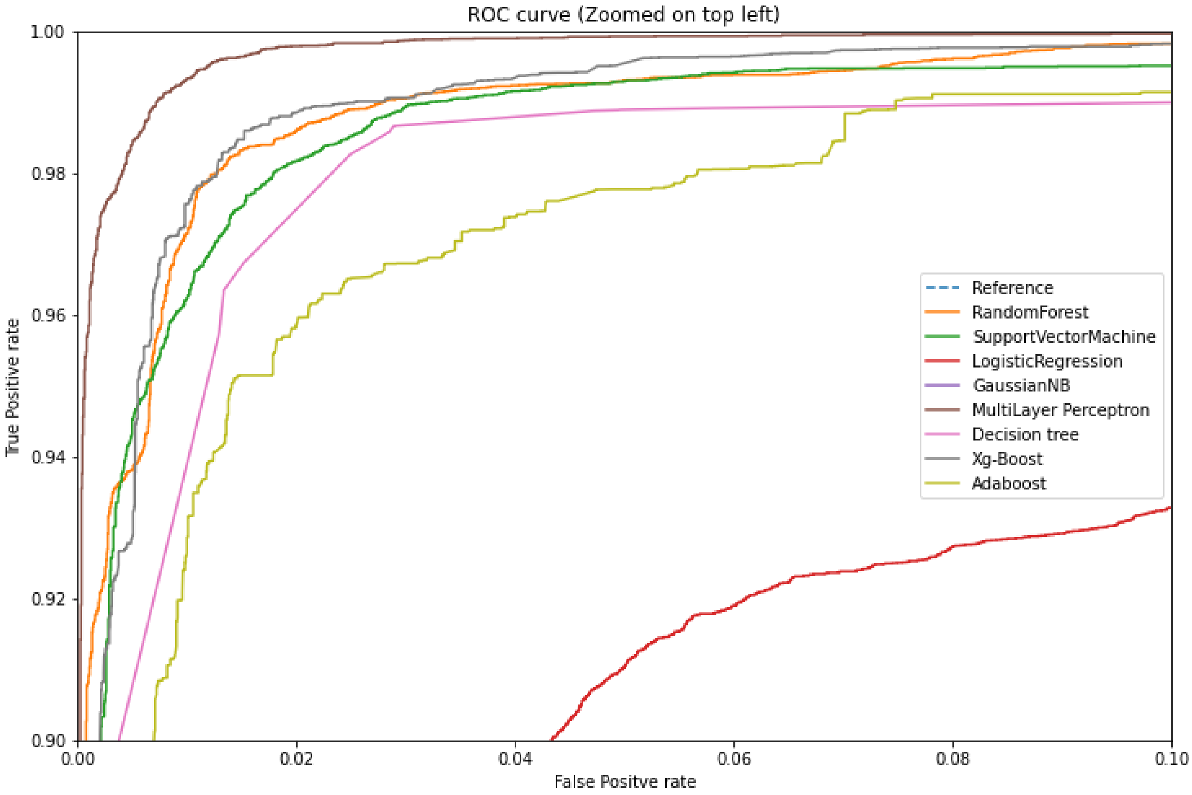
	accuracy	precision	recall	f1
Random Forest	0.980428	0.990145	0.970228	0.980085
Support Vector Machine	0.980966	0.980194	0.981487	0.980840
Multi-Layer Perceptron	0.991512	0.992355	0.990531	0.991442
Decision tree	0.978838	0.974903	0.982664	0.978768
XgBoost	0.984450	0.986744	0.981864	0.984298
Adaboost	0.970070	0.975043	0.964387	0.969685
Gaussian Naive Bayes	0.869546	0.885045	0.847230	0.865725
Logistic Regression	0.930178	0.949798	0.907292	0.928059

ROC Curve:



X-axis has the false positive rate and the y-axis has the true positive rate. The area under the curve should be as high as possible for the model to be a good model. Which implies that the model is good at distinguishing positive and negative cases.

ROC Curve Zoomed:



Conclusion

After training all the six models, listed below:-

1) Random Forest

2) Decision Trees

3) Support Vector Machine

4) Logistic Regression

5) Gaussian Naive Bayes

6) Multi-Layer Perceptron

7) Adaboost

8)Xgboost

Random Forest and Multi-Layer Perceptron have roughly 0.99 area under the curve on the ROC curve. The Area under the curve for Random Forest is the highest, while Decision Tree is the lowest. The Random Forest classifier most closely resembles the diagonal line in the

Reliability diagram (Calibration curve), resulting in dependable predictions. The Multi-Perceptron classifier has been producing acceptable results until now, but the calibration curve reveals that it is significantly off from the other two, indicating that the model's predictions are unreliable.The Random Forest ,Adaboost, Multi-Layer Perceptron classifiers have 99 percent accuracy, precision, recall, and f1 scores, according to the results table, based on these trained models. With a score of 98 percent, the Support Vector machine isn't far behind.So, it can be concluded that when it comes to identifying network intrusion, the Random Forest classifier outperforms all other models.

Challenges in Implementing

a) The dataset was not available in csv format. It was in txt format and had to be converted to csv format to process it.
b) The column names were also not available in the dataset. They were inferred from previous works.
c) There were 2 files included within the dataset. KDDTest+ i.e. the testing dataset and KDDTrain+ i.e. the training dataset. The test set contains a few types of attacks which are not in the train set. Every attack has a different kind of signature. The model can learn the signature of a type of an attack and can thus detect it in future. But for those attacks whose signatures are not known might not always be detected by the model. Hence the two sets were combined and then randomly split into train and test dataset again.

Future Scope

The future scope of this project lies in the idea that this kind of comparison can be extended to accommodate more machine learning models which are available. Also, to further strengthen the purpose of concluding which model fares the best, more metrics can be computed which can give more accurate results. Nevertheless, this project has already carried out comparison between 6 of the most widely used machine learning models and hence is reliable when choosing the right model for better results**.**

Contribution To The Society

As discussed earlier, Network Intrusion Detection Systems are present in our computer terminals to protect us from malware transmitted through the network and help combat intrusion threats. So by studying the effectiveness of various machine learning models, this project has actually contributed in choosing the best model to withstand intrusions and contribute in the field of Information Security

References:

1. Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017, September). Evaluation of machine learning algorithms for intrusion detection system. In *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)* (pp. 000277-000282). IEEE.
2. M. Shafiq, Z. Tian, A. K. Bashir, X. Du and M. Guizani, "CorrAUC: A Malicious Bot-IoT Traffic Detection Method in IoT Network Using Machine-Learning Techniques," in IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3242-3254, 1 March1, 2021, doi: 10.1109/JIOT.2020.3002255.
3. Mishra, P., Varadharajan, V., Tupakula, U., & Pilli, E. S. (2018). A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE communications surveys & tutorials*, *21*(1), 686-728.
4. Di Mauro, M., Galatro, G., Fortino, G., & Liotta, A. (2021). Supervised feature selection techniques in network intrusion detection: A critical review. *Engineering Applications of Artificial Intelligence*, *101*, 104216.
5. Paulauskas, N., & Auskalnis, J. (2017). Analysis of data preprocessing influence on intrusion detection using NSL-KDD dataset. Electrical, Electronic and Information Sciences.
6. Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, *5*, 21954-21961.
7. Biswas, S. K. (2018). Intrusion detection using machine learning: A comparison study. *International Journal of pure and applied mathematics*, *118*(19), 101-114.
8. Halimaa, A., & Sundarakantham, K. (2019, April). Machine learning based intrusion detection system. In *2019 3rd International conference on trends in electronics and informatics (ICOEI)* (pp. 916-920). IEEE.

9. Sarker, I. H., Abushark, Y. B., Alsolami, F., & Khan, A. I. (2020). Intrudtree: a machine learning based cyber security intrusion detection model. *Symmetry*, *12*(5), 754.
10. A. Kumar and T. J. Lim, "EDEMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques," 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), 2019, pp. 289-294, doi: 10.1109/WF-IoT.2019.8767194.

Video Link :
https://drive.google.com/file/d/1QGtx_YCPxfc9y7wqBRjGvwPYQypEXDLf/view?usp=sharing