**CYBER SECURITY**

**Final Report**

Harshil Gupta -17BCE1112

Shekhar Gaur – 17BCE1183

Shikhar Chaudhary – 17BCE1238

Mohika Thampi -17BCE1079

Pranjal Sachan -17BCE1027

**Classification Algorithm Used:**

* Decision Tree
* Gaussian Process
* Gradient Boosting

**DECISION TREE:-**

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation.

A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute and each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Dataset is divided into two sets of training and test dataset and each of the leaf node represent two classes to which test data belong to.

**GAUSSIAN PROCESS:-**

This classifier classifies new data into classes which has been made by training and test data initially. for a given new data(x), we want to estimate p ( y = 1|x) and p ( y = 2|x). X is assigned to any class which has the highest probability. There are two variants of Gaussian classifier, depending on whether covariance matrices of classes are assumed to be equal or not. Covariance matrix assumption has an impact on the class boundary. Shared covariance matrix leads to the linear boundary while separate covariance matrices lead to the quadratic boundary.

Our Dataset of different attacks has been classified in two various classes , where we predict for each attribute of data to belong in class 1 or class 2.

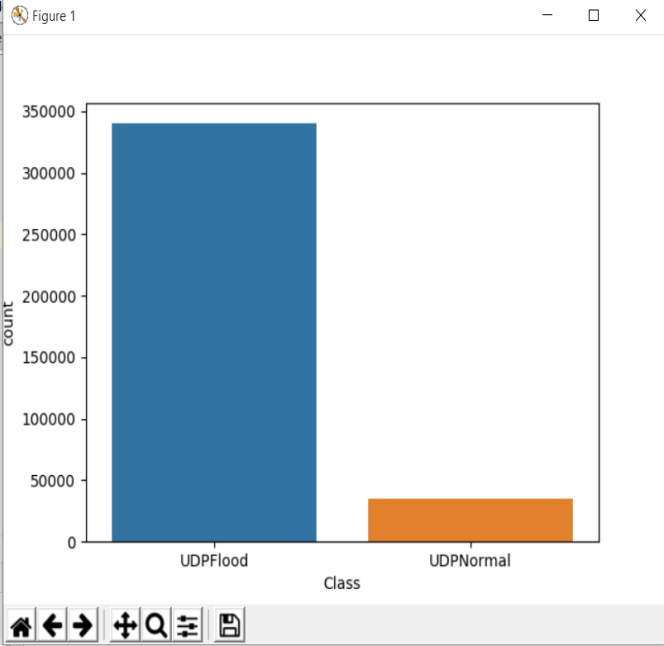
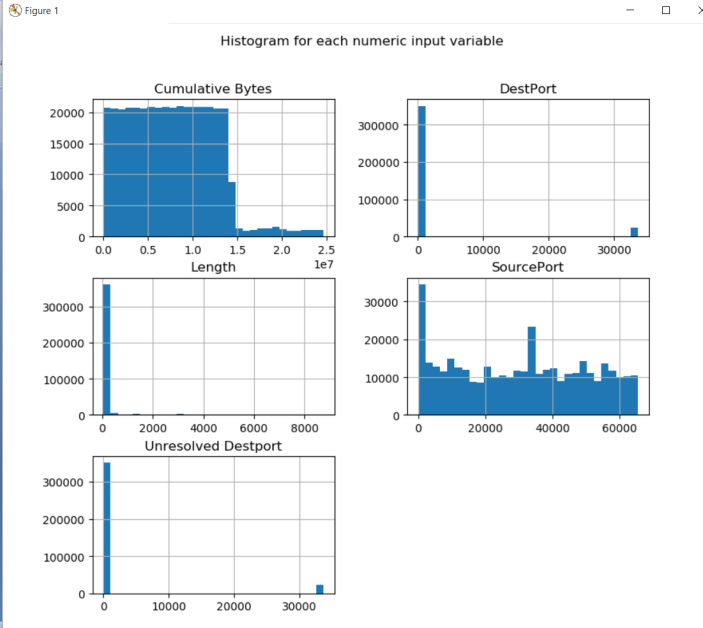
**GRADIENT BOOSTING:-**

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

VISUALIZATION OF DATASET

1. UDP

Figure 1- Count vs Class for UDP Figure 2 – Attributes vs Count for UDP

 ****

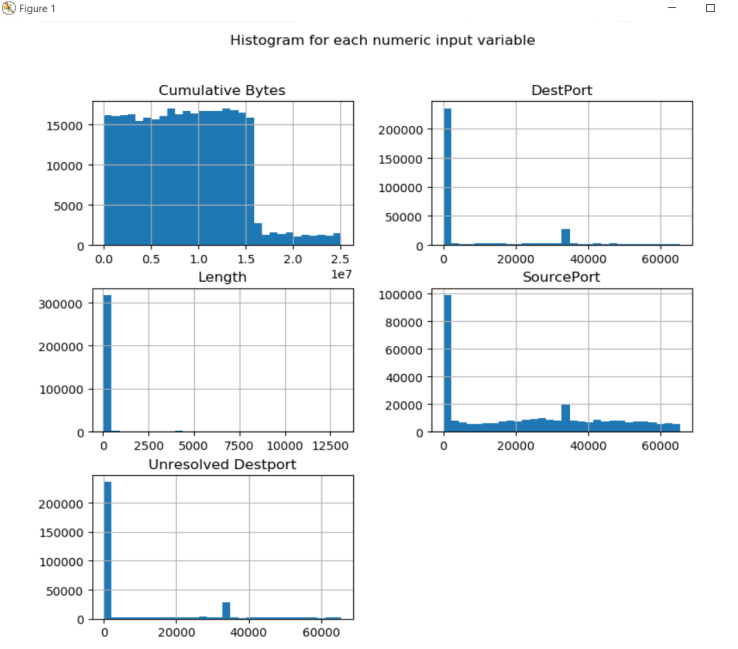
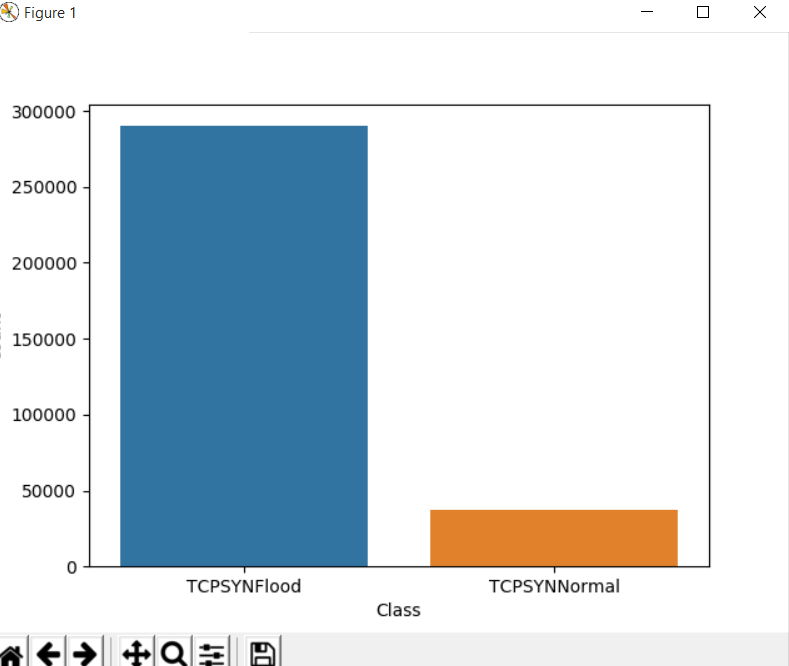
**Figure 1** depicts classification of whole dataset into classes.Here blue region depicts entries in UDPFlood class where as Orange depicts entries in UDPNormal Class.

**Figure 2** depicts classification of different attributes into two classes based on count.Y axis depicts count and x axis depicts values for the attributes.

\*\* Variation in histogram is because of particular range of values in attack file. Here length and DestPort have almost similar range values whereas other have varying values

1. TCPSYN

Figure 3- Count vs Class for TCPSYN Figure 4 – Attributes vs Count for TCPSYN



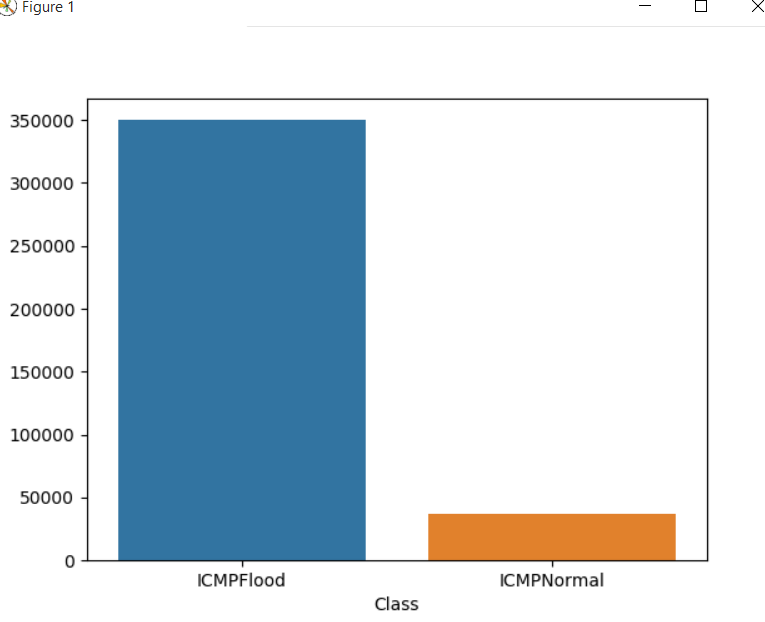
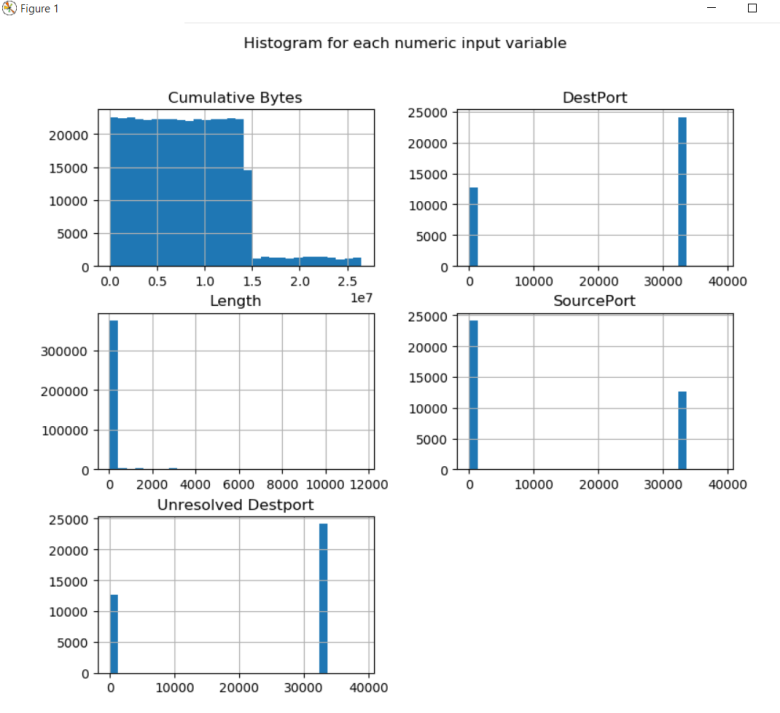
**Figure 3** depicts classification of whole dataset into classes.Here blue region depicts entries in TCPSYNFlood class where as Orange depicts entries in TCPSYNNormal Class.

**Figure 4** depicts classification of different attributes into two classes based on count.Y axis depicts count and x axis depicts values for the attributes.

\*\* Variation in histogram is because of particular range of values in attack file. Here length and DestPort,Unresoved DestPort have almost similar range values whereas other have varying values

1. ICMP

Figure 5- Count vs Class for ICMP Figure 6 – Attributes vs Count for ICMP

** **

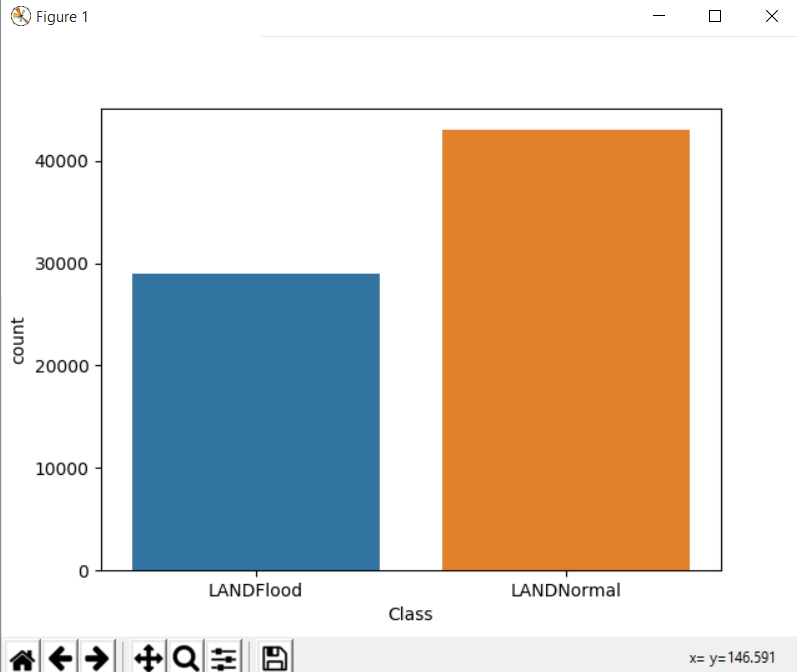
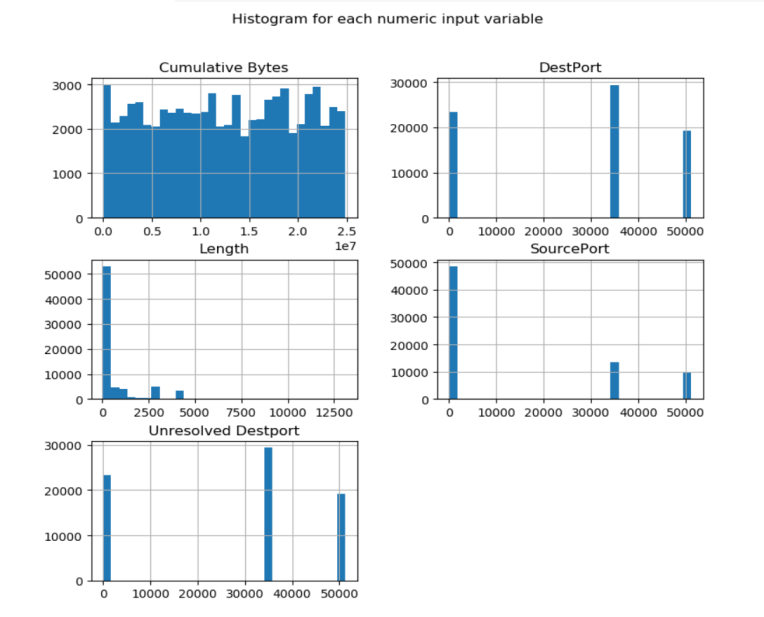
**Figure 5** depicts classification of whole dataset into classes.Here blue region depicts entries in ICMPFlood class where as Orange depicts entries in ICMPNormal Class.

**Figure 6** depicts classification of different attributes into two classes based on count.Y axis depicts count and x axis depicts values for the attributes.

**\*\*For Figure-1,3,5- High count value of UDPFlood infers more prone of data to DOS attack**

1. LAND

Figure 7- Count vs Class for LAND Figure 8 – Attributes vs Count for LAND

** **

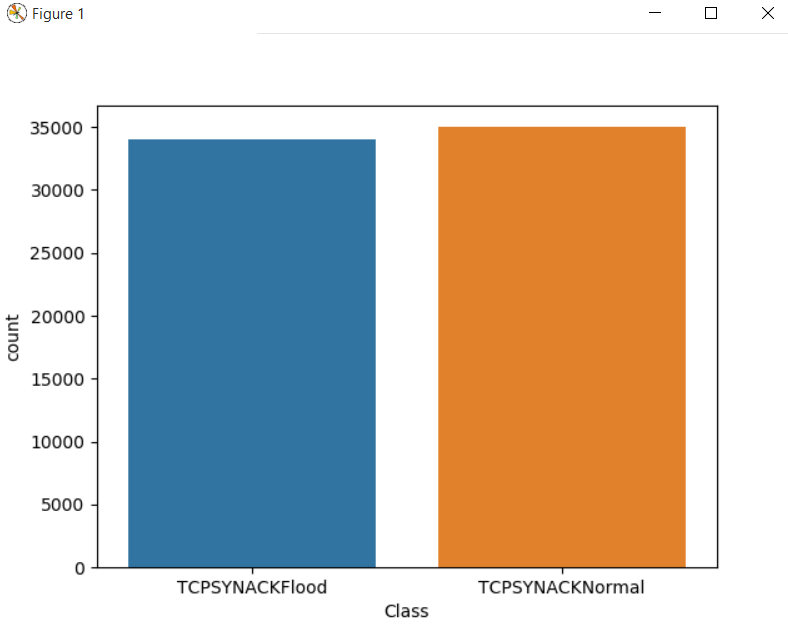
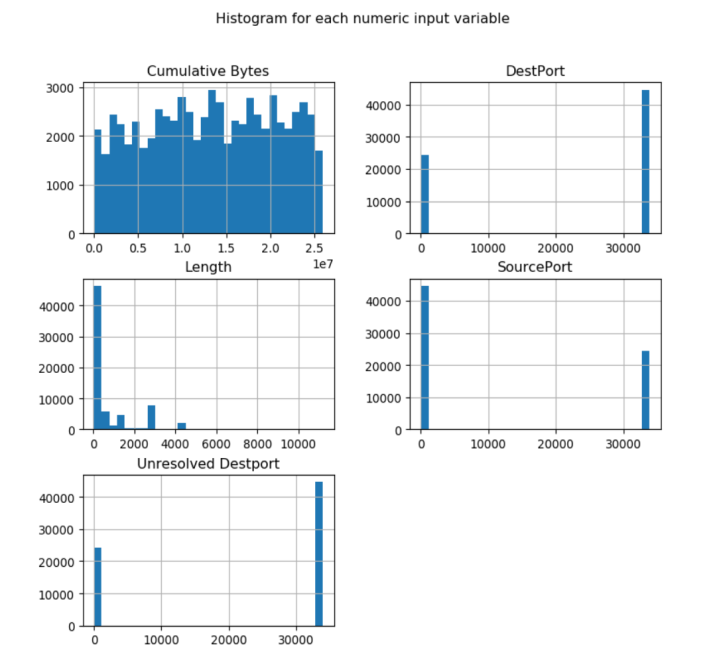
**Figure7** depicts classification of whole dataset into classes.Here blue region depicts entries in LANDFlood class where as Orange depicts entries in LANDNormal Class.

**\*\*For Figure-7 Less count value of UDPFlood infers less prone of data to DOS attack**

**Figure 8** depicts classification of different attributes into two classes based on count.Y axis depicts count and x axis depicts values for the attributes.

1. TCPSYNACK

Figure 9- Count vs Class for TCPSYNACK Figure 10 – Attributes vs Count for TCPSYNACK

** **

**Figure 9** depicts classification of whole dataset into classes.Here blue region depicts entries in TCPSYNACKFlood class where as Orange depicts entries in TCPSYNACKNormal Class.

**\*\*For Figure-9 Same values for Flood and Normal data means data entries equally prone to attacked.**

**Figure 10** depicts classification of different attributes into two classes based on count.Y axis depicts count and x axis depicts values for the attributes.

**By Varying Different Parameters for Classifiers**

1. **Decision Tree**

Table 1 – Decision Tree accuracy for different attacks.

|  |  |
| --- | --- |
| **Attack** | **Accuracy** |
| UDP | 0.9952 |
| TCPSYN | 0.999311 |
| ICMP | 0.999418 |
| LAND | 0.834027 |
| TCPSYNACK | 0.772717 |

\*\*Less accuracy for TCPSYNACK because of almost equal distribution of data into two classes.

1. **Gradient Boosting**

Varying Parameter: Learning Rate

Table 2 – Gradient Boosting accuracy for UDP

|  |  |  |
| --- | --- | --- |
| **Attack** | **Learning Rate** | **Accuracy** |
| **UDP** | 0.2 | 0.9997266666666667 |
|  | 0.5 | 0.9997266666666667 |
|  | 1.0 | 0.9997266666666667 |
|  | 1.5 | 0.97086 |
|  | 2.0 | 0.9707666666666667 |

Table 2 – Gradient Boosting accuracy for TCPSYN

|  |  |  |
| --- | --- | --- |
| **Attack** | **Learning Rate** | **Accuracy** |
| **TCPSYN** | 0.2 | 0.9996253822629969 |
|  | 0.5 | 0.9996253822629969 |
|  | 1.0 | 0.9996253822629969 |
|  | 1.5 | 0.9820336391437309 |
|  | 2.0 | 0.8861238532110092 |
| **Attack** | **Learning Rate** | **Accuracy** |
| **ICMP** | 0.2 | 0.9996640826873385 |
|  | 0.5 | 0.999657622739018 |
|  | 1.0 | 0.9996770025839793 |
|  | 1.5 | 0.9995025839793281 |
|  | 2.0 | 0.9993023255813953 |

Table 3– Gaussian process accuracy for ICMP

Table 4 – Gradient Boosting accuracy for LAND

|  |  |  |
| --- | --- | --- |
| **Attack** | **Learning Rate** | **Accuracy** |
| **LAND** | 0.2 | 0.6538541666666666 |
|  | 0.5 | 0.6625694444444444 |
|  | 1.0 | 0.6758333333333333 |
|  | 1.5 | 0.6765277777777777 |
|  | 2.0 | 0.5698958333333334 |

Table 5 – Gradient Boosting accuracy for TCPSYNACK

|  |  |  |
| --- | --- | --- |
| **Attack** | **Learning Rate** | **Accuracy** |
| **TCPSYNACK** | 0.2 | 0.5839855072463768 |
|  | 0.5 | 0.5942391304347826 |
|  | 1.0 | 0.6068115942028985 |
|  | 1.5 | 0.608804347826087 |
|  | 2.0 | 0.5204710144927536 |

\*\*Table 4,5 – Less accuracy for classification at higher learning rate because boosting shrinks the contribution of each new base model.

1. **Gaussian Process**

Varying Parameter: Kernel

Table 6 – Gaussian process accuracy for UDP

|  |  |  |
| --- | --- | --- |
| **Attack** | **Kernel** | **Accuracy** |
| **UDP** | 0.5 | 0.9158466666666667 |
|  | 1.0 | 0.9158466666666667 |
|  | 1.5 | 0.9158466666666667 |
|  | 2.0 | 0.9158466666666667 |
|  |  |  |

Table 7 – Gaussian process accuracy for TCPSYN

|  |  |  |
| --- | --- | --- |
| **Attack** | **Kernel** | **Accuracy** |
| **TCPSYN** | 0.5 | 0.970565749235474 |
|  | 1.0 | 0.9501376146788991 |
|  | 1.5 | 0.9501376146788991 |
|  | 2.0 | 0.9501376146788991 |
|  |  |  |

Table 8 – Gaussian process accuracy for ICMP

|  |  |  |
| --- | --- | --- |
| **Attack** | **Kernel** | **Accuracy** |
| **ICMP** | 0.5 | 0.9246511627906977 |
|  | 1.0 | 0.9436692506459948 |
|  | 1.5 | 0.9516020671834625 |
|  | 2.0 | 0.9516020671834625 |
|  |  |  |

Table 9 – Gaussian process accuracy for LAND

|  |  |  |
| --- | --- | --- |
| **Attack** | **Kernel** | **Accuracy** |
| **LAND** | 0.5 | 0.553125 |
|  | 1.0 | 0.553125 |
|  | 1.5 | 0.553125 |
|  | 2.0 | 0.553125 |
|  |  |  |

Table 10 – Gaussian process accuracy for TCPSYNACK

|  |  |  |
| --- | --- | --- |
| **Attack** | **Kernel** | **Accuracy** |
| **TCPSYNACK** | 0.5 | 0.5244565217391305 |
|  | 1.0 | 0.5115217391304347 |
|  | 1.5 | 0.4914855072463768 |
|  | 2.0 | 0.4914855072463768 |
|  |  |  |

\*\* Table 6,7,8,9,10 – Almost same Accuracy for different kernel because of same covariance of data.

\*\*Table 9,10 – Less accuracy rate due to more iterations in the Gaussian function.

**VALIDATION METRICES-:**

1. **UDP**

Table 11– Performance measures for Decision Tree

|  |  |
| --- | --- |
| **TP** | 149928 |
| **FN** | 72 |
| **FP** | 37 |
| **TN** | 135779 |
| **Recall** | 0.99952 |
| **Precision** | 0.999753275764345 |
| **F-measure** | 0.9996366242728318 |

Table 12– Performance measures for Gradient Boosting

|  |  |
| --- | --- |
| **TP** | 295543 |
| **FN** | 4457 |
| **FP** | 39 |
| **TN** | 271593 |
| **Recall** | 0.9851433333333334 |
| **Precision** | 0.9998680569182156 |
| **F-measure** | 0.9924510814631738 |

Table 13– Performance measures for Gaussian Classifier

|  |  |
| --- | --- |
| **TP** | 432920 |
| **FN** | 17080 |
| **FP** | 39 |
| **TN** | 407409 |
| **Recall** | 0.962044444444445 |
| **Precision** | 0.9999099221866273 |
| **F-measure** | 0.9806117837861101 |

1. **TCPSYN**

Table 14– Performance measures for Decision Tree

|  |  |
| --- | --- |
| **TP** | 130710 |
| **FN** | 90 |
| **FP** | 50 |
| **TN** | 115829 |
| **Recall** | 0.9993119266055046 |
| **Precision** | 0.9996176200672988 |
| **F-measure** | 0.9994647499617678 |

Table 15– Performance measures for Gradient Boosting

|  |  |
| --- | --- |
| **TP** | 2246615 |
| **FN** | 14985 |
| **FP** | 50 |
| **TN** | 231708 |
| **Recall** | 0.9451911314984709 |
| **Precision** | 0.9998625129399586 |
| **F-measure** | 0.9704189743539294 |

Table 16– Performance measures for Gaussian Classifier

|  |  |
| --- | --- |
| **TP** | 370893 |
| **FN** | 21507 |
| **FP** | 51 |
| **TN** | 347586 |
| **Recall** | 0.9451911314948709 |
| **Precision** | 0.9998625129399586 |
| **F-measure** | 0.97175847324402421 |

1. **ICMP**

Table 17– Performance measures for Decision Tree

|  |  |
| --- | --- |
| **TP** | 154710 |
| **FN** | 90 |
| **FP** | 43 |
| **TN** | 139898 |
| **Recall** | 0.9994186046511628 |
| **Precision** | 0.9997221378583937 |
| **F-measure** | 0.9995703482117763 |

Table 18– Performance measures for Gradient Boosting

|  |  |
| --- | --- |
| **TP** | 309402 |
| **FN** | 198 |
| **FP** | 104 |
| **TN** | 279778 |
| **Recall** | 0.9993604651162791 |
| **Precision** | 0.9996639806659645 |
| **F-measure** | 0.9995121998494604 |

Table 19– Performance measures for Gaussian

|  |  |
| --- | --- |
| **TP** | 456710 |
| **FN** | 7690 |
| **FP** | 104 |
| **TN** | 419719 |
| **Recall** | 0.9834409991386736 |
| **Precision** | 0.9997723362243714 |
| **F-measure** | 0.9915394251498567 |

1. **LAND**

Table 20– Performance measures for Decision Tree

|  |  |
| --- | --- |
| **TP** | 24015 |
| **FN** | 4785 |
| **FP** | 2244 |
| **TN** | 9256 |
| **Recall** | 0.8338541666666667 |
| **Precision** | 0.9145435850565521 |
| **F-measure** | 0.8723369476379884 |

Table 21– Performance measures for Gradient Boosting

|  |  |
| --- | --- |
| **TP** | 40428 |
| **FN** | 17172 |
| **FP** | 8992 |
| **TN** | 14008 |
| **Recall** | 0.701875 |
| **Precision** | 0.8180493727235937 |
| **F-measure** | 0.7555223322743412 |

Table 22– Performance measures for Gaussian

|  |  |
| --- | --- |
| **TP** | 56358 |
| **FN** | 30042 |
| **FP** | 16416 |
| **TN** | 18084 |
| **Recall** | 0.6522916666666667 |
| **Precision** | 0.7744249319812021 |
| **F-measure** | 0.7081307248671266 |

1. **TCPSYNACK**

Table 23– Performance measures for Decision Tree

|  |  |
| --- | --- |
| **TP** | 21310 |
| **FN** | 6290 |
| **FP** | 2978 |
| **TN** | 10587 |
| **Recall** | 0.7721014492753623 |
| **Precision** | 0.8773880105401844 |
| **F-measure** | 0.8213845205057045 |

Table 24– Performance measures for Gradient Boosting

|  |  |
| --- | --- |
| **TP** | 35675 |
| **FN** | 19525 |
| **FP** | 4839 |
| **TN** | 22291 |
| **Recall** | 0.646286231884058 |
| **Precision** | 0.8805598064866466 |
| **F-measure** | 0.7454499864178699 |

Table 25– Performance measures for Gaussian

|  |  |
| --- | --- |
| **TP** | 49240 |
| **FN** | 33560 |
| **FP** | 4839 |
| **TN** | 35856 |
| **Recall** | 0.5946859903381643 |
| **Precision** | 0.9105197951145546 |
| **F-measure** | 0.7194675589389168 |

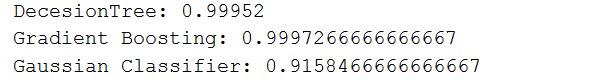
**\*\* Table 11-25 – High precision for each classification means that each measurement is close to each other**

**ATTACKS :**

1. **UDP**

Table 26– Accuracy Table for UDP

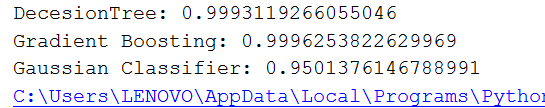
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Decision Tree** | 0.9952 |
| **2.** | **Gaussian Process** | 0.915846 |
| **3.** | **Gradient Boosting** | 0.999726 |

****

1. **TCPSYN**

Table 27– Accuracy Table for TCPSYN

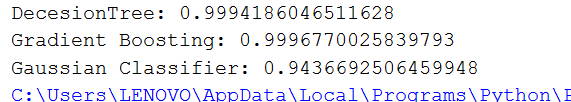
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Decision Tree** | 0.999311 |
| **2.** | **Gaussian Process** | 0.950137 |
| **3.** | **Gradient Boosting** | 0.999625 |

****

1. **ICMP**

Table 28– Accuracy Table for ICMP

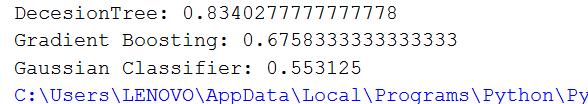
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Decision Tree** | 0.999418 |
| **2.** | **Gaussian Process** | 0.9436692 |
| **3.** | **Gradient Boosting** | 0.999677 |

****

1. **LAND**

Table 29– Accuracy Table for LAND

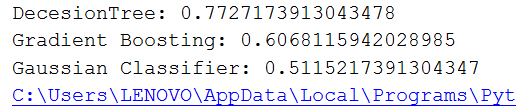
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Decision Tree** | 0.834027777 |
| **2.** | **Gaussian Process** | 0.553125 |
| **3.** | **Gradient Boosting** | 0.6758333 |

****

1. **TCPSYNACK**

Table 30– Accuracy Table for TCPSYNACK

|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Decision Tree** | 0.772717 |
| **2.** | **Gaussian Process** | 0.511521 |
| **3.** | **Gradient Boosting** | 0.606811 |

****

\*\*Table 29,30- Reason behind less accuracy of Gradient Boosting in different attack files is because number of iterations increases in this process which takes more time as compared to Decision Tree .

**ALGORITHM**

* Install and import scikit,numpy,sklearn libraries in python
* Clean the dataset by removing null or NaN values and convert other values into readable format.
* Split the data into training and test data for classification
* Import functions of different classifiers with suitable parameters for classification process.
* Get accuracy of each classifier and calculate required TP,FP,TN,FN scores.
* Plot the graph of Dataset using matplotlib with number of values on y-axis and classes on x-axis.

**Ensemble Algorithm Used:**

* Stacking Ensemble
* Major Voting Ensemble
* Weighted Ensemble Classifier

**STACKING ENSEMBLE:-**

Stacking is a ensemble model, where a new model is trained from the combined predictions of two (or more) previous model. The predictions from the models are used as inputs for each sequential layer, and combined to form a new set of predictions. These can be used on additional layers, or the process can stop here with a final result

Ensemble stacking can be referred to as blending, because all the numbers are blended to produce a prediction or classification.

Here we have used Decision tree ,Gradient Boosting and random forest as basic classifiers.

**MAJOR VOTING ENSEMBLE:-**

This model makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes. If none of the predictions get more than half of the votes, we may say that the ensemble method could not make a stable prediction for this instance.

**WEIGHTED ENSEMBLER CLASSIFIER:-**

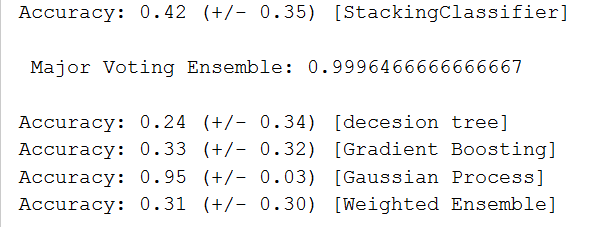
Unlike majority voting, where each model has the same rights, we can increase the importance of one or more models. In weighted voting you count the prediction of the better models multiple times. Finding a reasonable set of weights is up to you.

**ATTACKS :**

1. **UDP**

Table 31– Ensemble Accuracy Table for UDP

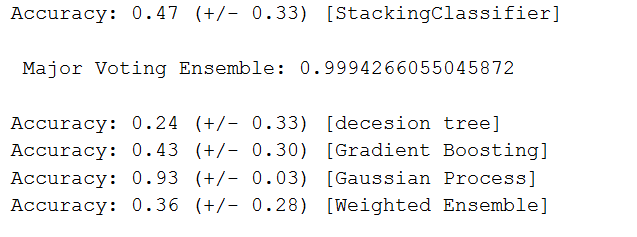
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Stacking Ensemble** | 0.42 (+/- 0.35) |
| **2.** | **Major Voting Ensemble** | 0.9996466666666667 |
| **3.** | **Weighted Ensemble Classifier** | 0.31 (+/- 0.30) |

****

1. **TCPSYN**

Table 32– Ensemble Accuracy Table for TCPSYN

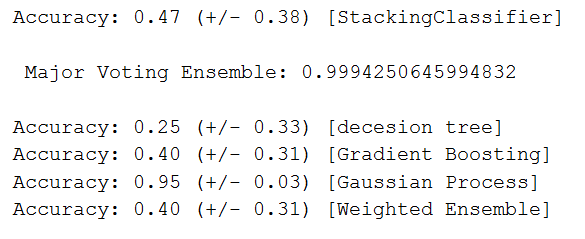
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Stacking Ensemble** | 0.47 (+/- 0.33) |
| **2.** | **Major Voting Ensemble** | 0.9994266055045872 |
| **3.** | **Weighted Ensemble Classifier** | 0.36 (+/- 0.28) |

****

1. **ICMP**

Table 33– Ensemble Accuracy Table for ICMP

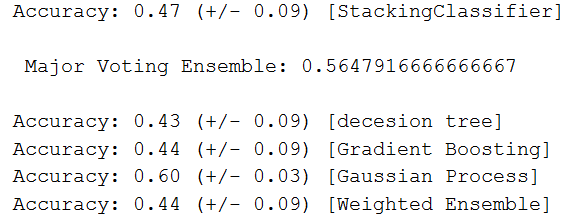
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Stacking Ensemble** | 0.47 (+/- 0.38) |
| **2.** | **Major Voting Ensemble** | 0.9994250645994832 |
| **3.** | **Weighted Ensemble Classifier** | 0.40 (+/- 0.31) |

****

1. **LAND**

Table 34– Ensemble Accuracy Table for LAND

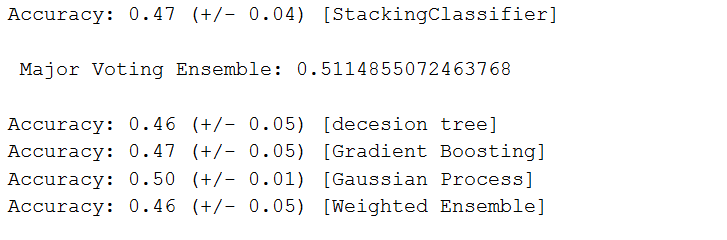
|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Stacking Ensemble** | 0.47 (+/- 0.09) |
| **2.** | **Major Voting Ensemble** | 0.5639930555555556 |
| **3.** | **Weighted Ensemble Classifier** | 0.44 (+/- 0.09) |

****

1. **TCPSYNACK**

Table 35– Ensemble Accuracy Table for TCPSYNACK

|  |  |  |
| --- | --- | --- |
| **S.No** | **Model** | **Accuracy** |
| **1.** | **Stacking Ensemble** | 0.47 (+/- 0.04) |
| **2.** | **Major Voting Ensemble** | 0.5086594202898551 |
| **3.** | **Weighted Ensemble Classifier** | 0.46 (+/- 0.05) |

****

\*\*Table 31-35-Ensemble algorithm used Decision tree, Gaussian and Gradient Boosting as classifiers.

Majority Voting Ensemble shows higher accuracy result because it makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes.

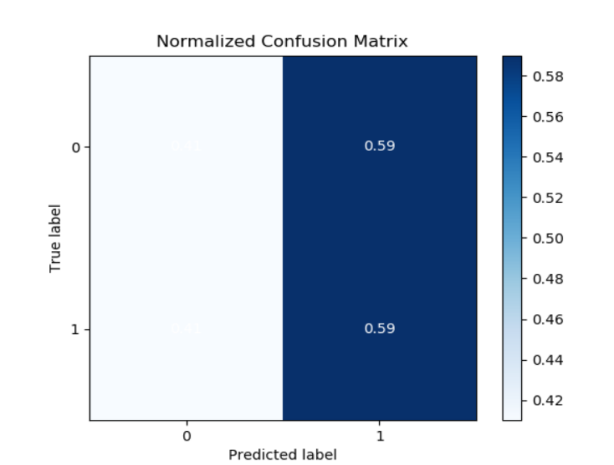
Whereas in stacking ensemble result is based on the training of previous models of classification.

**VISUALIZATION OF DATASET (Using Normalized Confusion Matrix)**

1. UDP

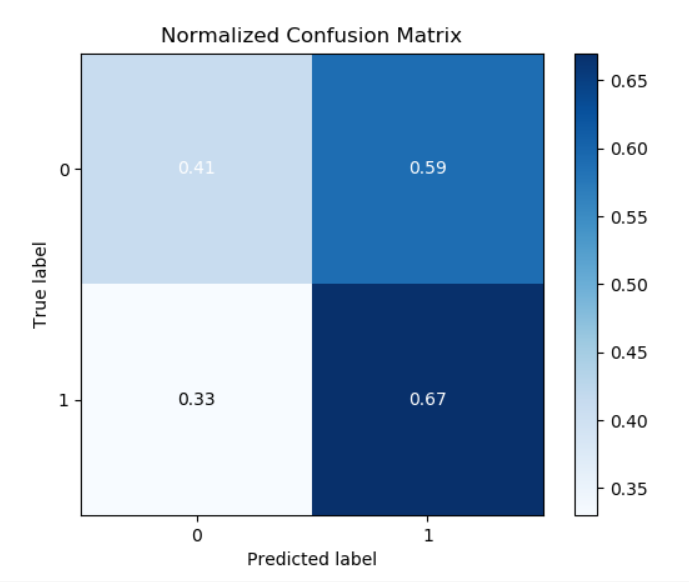
**Stacking Ensemble: TP: 0.41, FN: 0.59,FP :0.41,TN: 0.59**

Figure 11 – Confusion Matrix for Stacking Ensemble

****

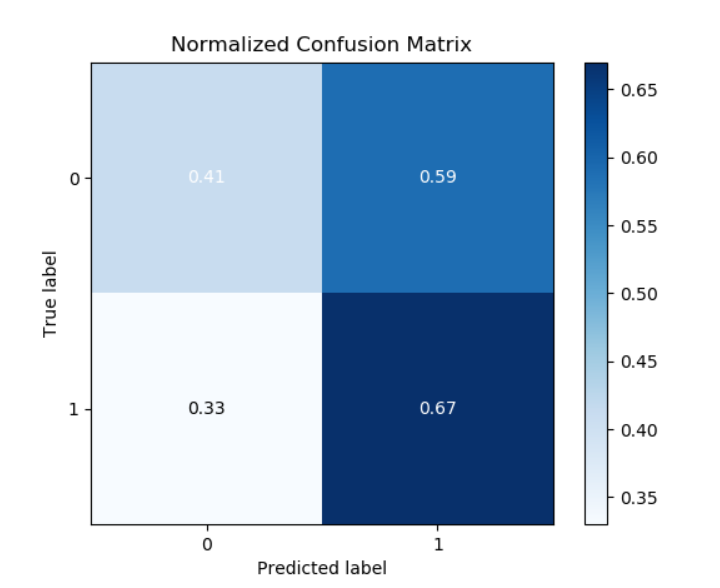
**Major Voting Ensemble TP: 0.41, FN: 0.59,FP :0.33,TN: 0.67**

Figure 12 – Confusion Matrix for Major Voting Ensemble

****

**Weighted Ensembler Classifier TP: 0.41, FN: 0.59,FP :0.33,TN: 0.67**

Figure 13 – Confusion Matrix for Weighted Ensemble

****

**Figure 11** depicts the normalized confusion matrix for stacking ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 12** depicts the normalized confusion matrix for Major Voting classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

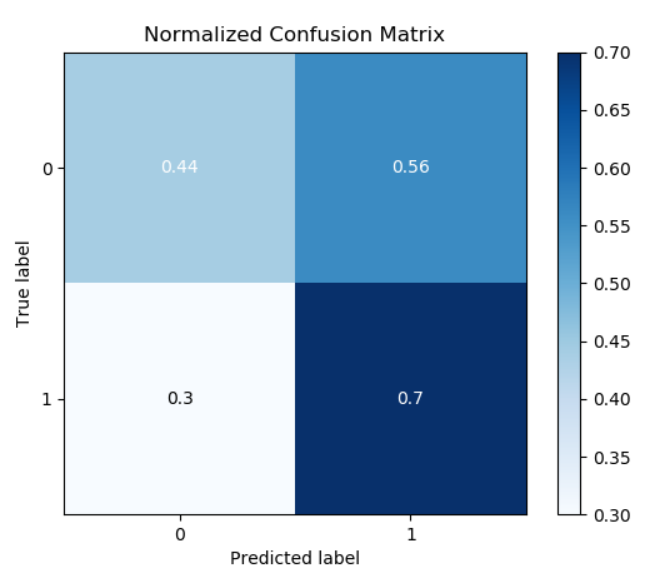
**Figure 13** depicts the normalized confusion matrix for Weighted ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**\*\* Figure 11-13 High TN values means more non predicted precised data by ensembles.**

1. TCPSYN

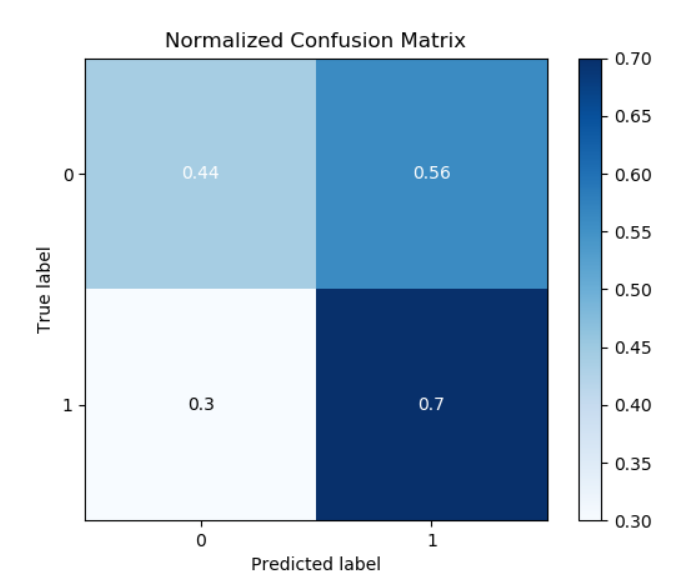
**Stacking Ensemble TP: 0.44, FN: 0.56,FP :0.3,TN: 0.7**

Figure 14 – Confusion Matrix for Stacking Ensemble



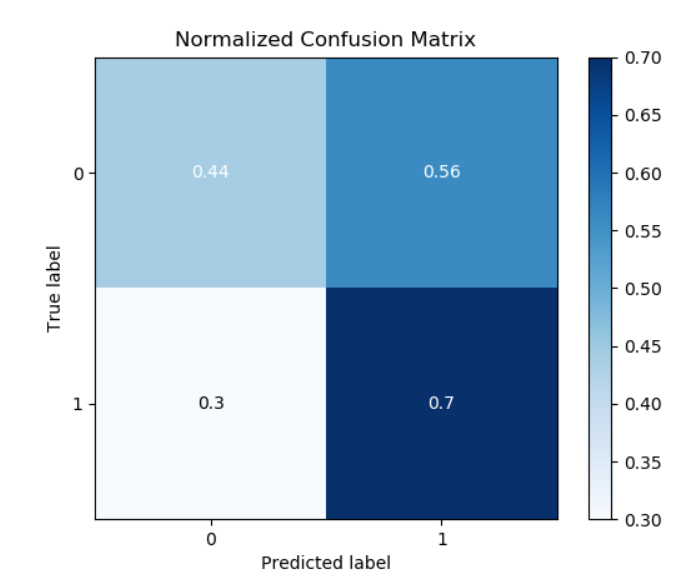
**Major Voting Ensemble TP: 0.44, FN: 0.56,FP :0.3,TN: 0.7**

Figure 15 – Confusion Matrix for Major Voting



**Weighted Ensemble Classifier TP: 0.44, FN: 0.56,FP :0.3,TN: 0.7**

Figure 16 – Confusion Matrix for Weighted Ensemble

****

**Figure 14** depicts the normalized confusion matrix for stacking ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 15** depicts the normalized confusion matrix for Major Voting classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

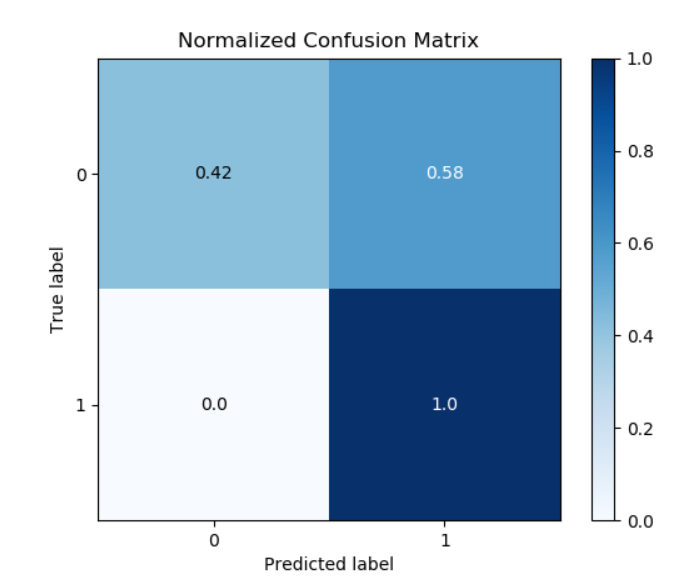
**Figure 16** depicts the normalized confusion matrix for Weighted ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**\*\* Figure 12-16 High TN values means more non predicted precised data by ensembles.**

1. ICMP

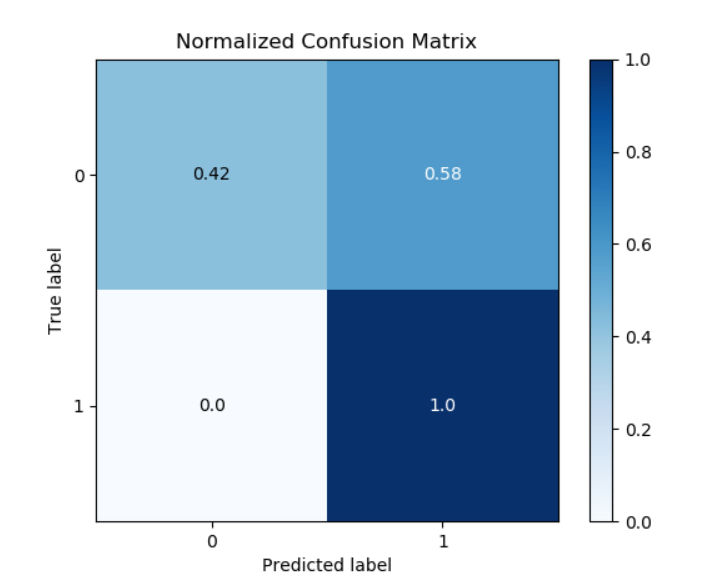
**Stacking Ensemble TP: 0.42, FN: 0.58,FP :0.0,TN: 1.0**

Figure 17 – Confusion Matrix for Stacking Ensemble



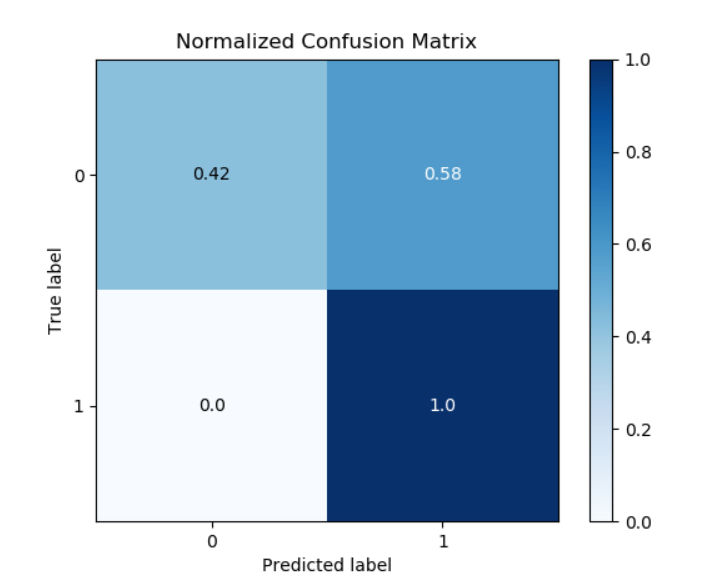
**Major Voting Ensemble TP: 0.42, FN: 0.58,FP :0.0,TN: 1.0**

Figure 18 – Confusion Matrix for Major Voting

****

**Weighted Ensemble Classifier TP: 0.42, FN: 0.58,FP :0.0,TN: 1.0**

Figure 19 – Confusion Matrix for Weighted Ensemble

****

**Figure 17** depicts the normalized confusion matrix for stacking ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 18** depicts the normalized confusion matrix for Major Voting classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

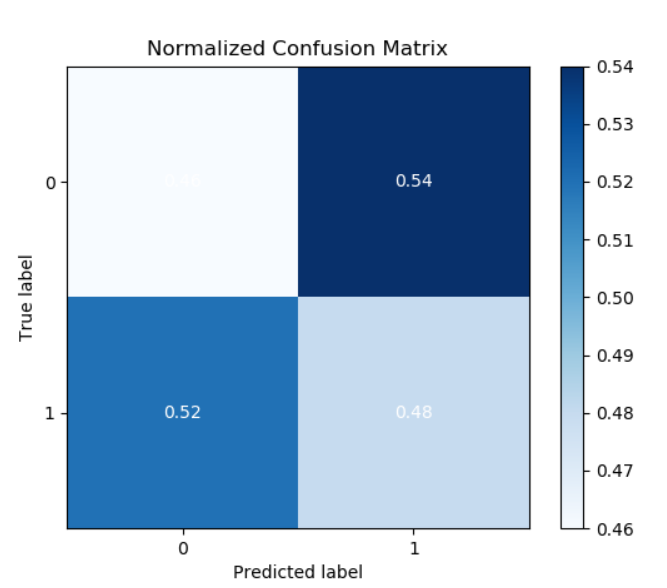
**Figure 19** depicts the normalized confusion matrix for Weighted ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**\*\* Figure 17-1 9 TN value=1 means that all the non- predicted documents by classifiers are precised**

1. LAND

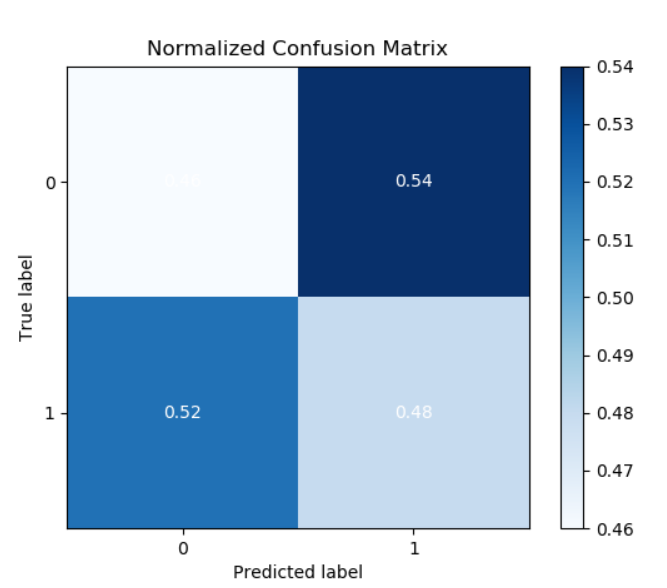
**Stacking Ensemble TP: 0.46, FN: 0.54,FP :0.52,TN: 0.48**

Figure 20 – Confusion Matrix for Stacking Ensemble



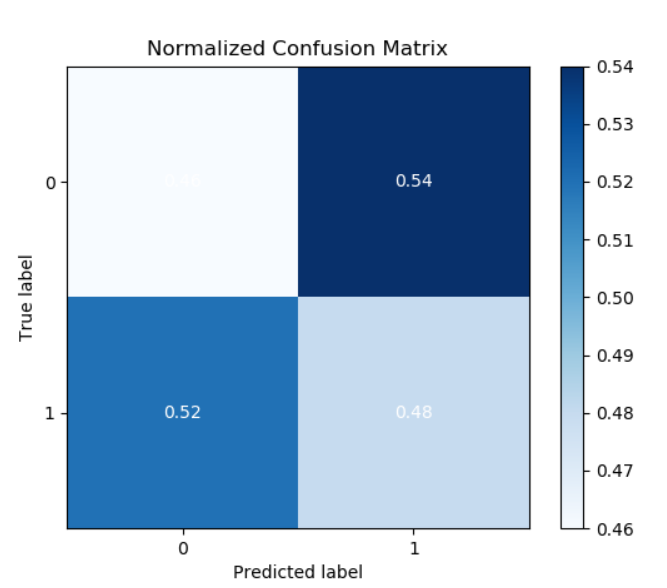
**Major Voting Ensemble TP: 0.46, FN: 0.54,FP :0.52,TN: 0.48**

Figure 21 – Confusion Matrix for Major Voting



**Weighted Ensemble Classifier TP: 0.46, FN: 0.54,FP :0.52,TN: 0.48**

Figure 22 – Confusion Matrix for Weighted Ensemble

****

**Figure 20** depicts the normalized confusion matrix for stacking ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

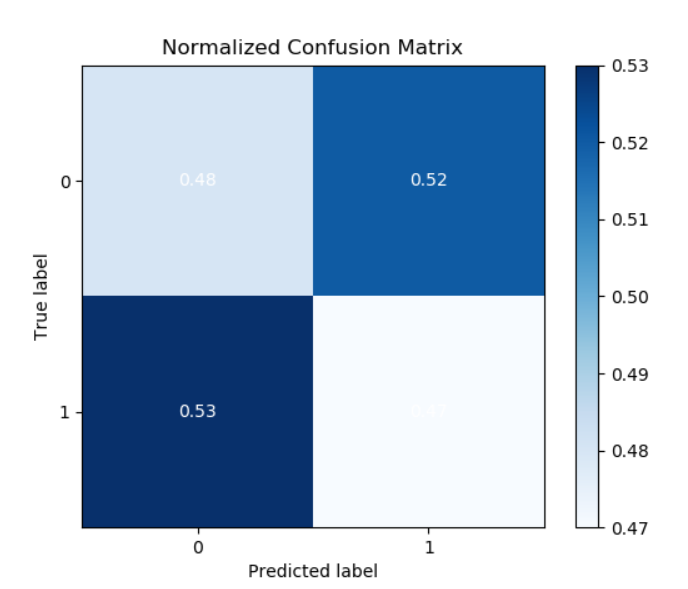
**Figure 21** depicts the normalized confusion matrix for Major Voting classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 22** depicts the normalized confusion matrix for Weighted ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

1. TCPSYNACK

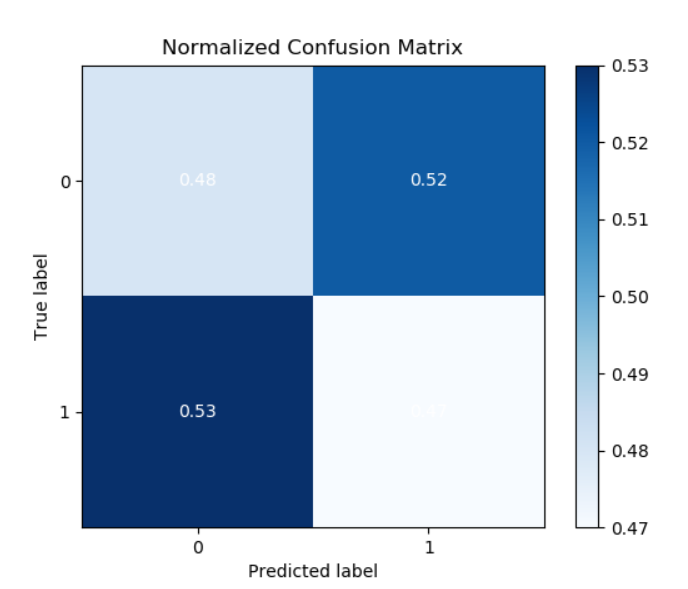
**Stacking Ensemble TP: 0.48, FN: 0.52,FP :0.53,TN: 0.47**

Figure 23 – Confusion Matrix for Stacking Ensemble



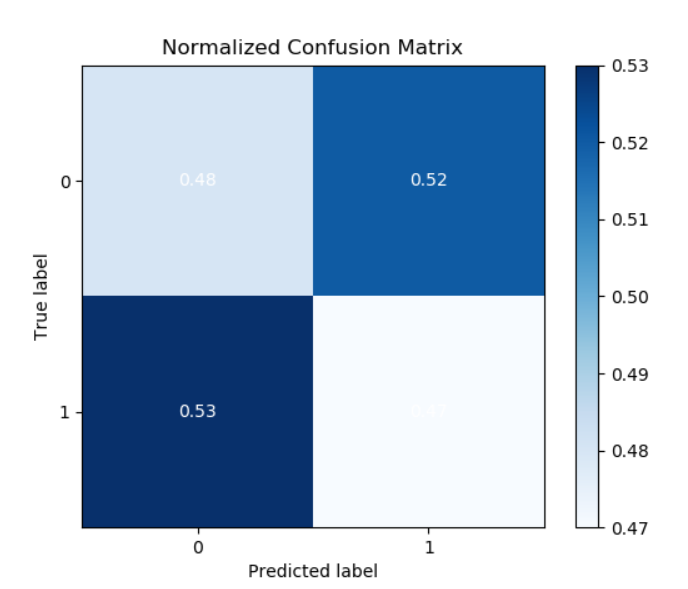
**Major Voting Ensemble TP: 0.48, FN: 0.52,FP :0.53,TN: 0.47**

Figure 24 – Confusion Matrix for Major Voting



**Weighted Ensemble Classifier TP: 0.48, FN: 0.52,FP :0.53,TN: 0.47**

Figure 25 – Confusion Matrix for Weighted Ensemble

****

**Figure 23** depicts the normalized confusion matrix for stacking ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 24** depicts the normalized confusion matrix for Major Voting classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**Figure 25** depicts the normalized confusion matrix for Weighted ensemble classification algorithm which infers the predicted labels from the dataset and hence gives values of TP,FP,TN,FN.

**ALGORITHM**

* Import necessary libraries like sklearn,scikit,numpy.
* Clean dataset and divide into train and test data.
* Use sklearn inbuilt classification algorithms.
* Zip all these algorithms as parameters for different ensemblers.
* Get accuracy of each ensemble.

**Classification Algorithm Used:**

* Gaussian
* Decision Tree
* Gradient Boosting

**Note:**

Algorithm 1,Algorithm 2,Algorithm 3= weights[clf1 , clf2, clf3]

Clf1 - Decision Tree

Clf2 - Gaussian

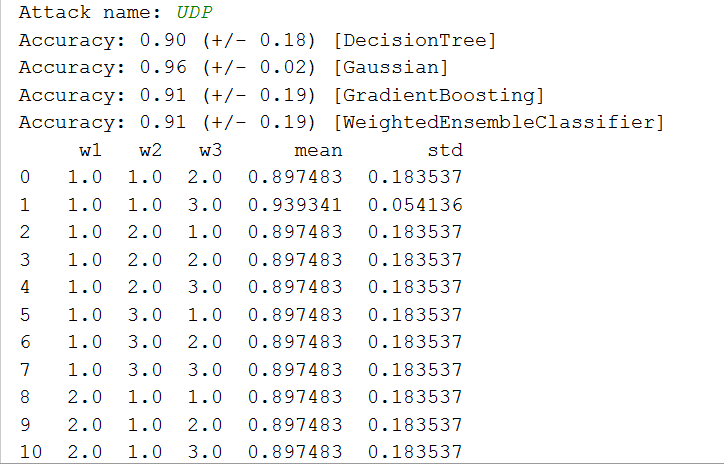
Clf3 – Gradient Boosting

**ATTACKS :**

1. **UDP**

Table 36– Weighted Ensemble with different weights ,Mean ,std

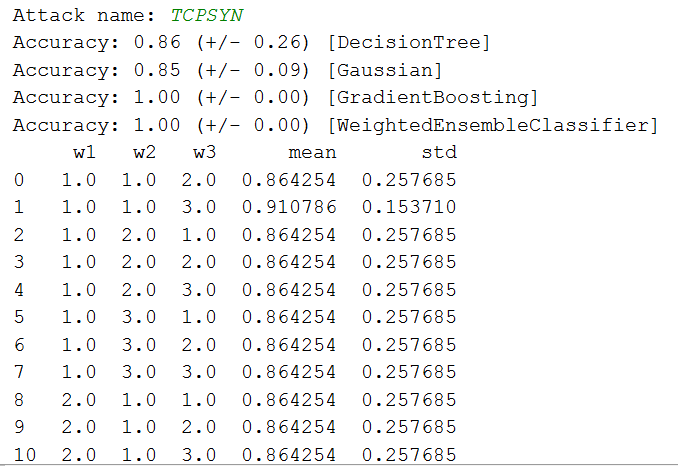
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Algorithm 1** | **Algorithm 2** | **Algorithm 3** | **Mean** | **Std** |
| **1** | 1.0 | 1.0 | 2.0 | 0.897483 | 0.183537 |
| **2** | 1.0 | 1.0 | 3.0 | 0.939341 | 0.054136 |
| **3** | 1.0 | 2.0 | 1.0 | 0.897483 | 0.183537 |

****

1. **TCPSYN**

Table 37– Weighted Ensemble with different weights ,Mean ,std

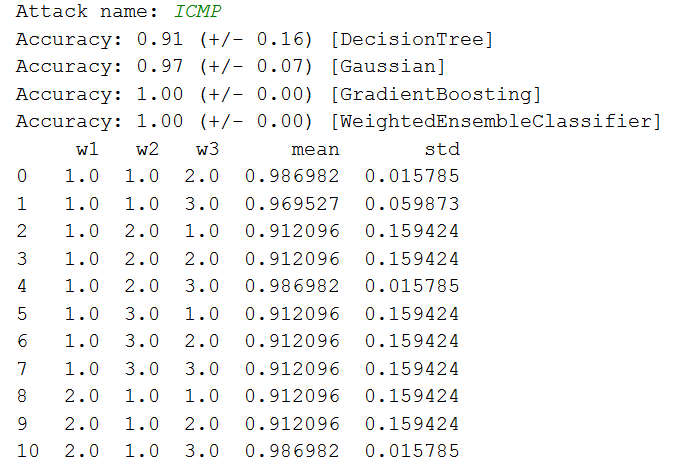
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Algorithm 1** | **Algorithm 2** | **Algorithm 3** | **Mean** | **Std** |
| **1** | 1.0 | 1.0 | 2.0 | 0.864254 | 0.257685 |
| **2** | 1.0 | 1.0 | 3.0 | 0.910786 | 0.153710 |
| **3** | 1.0 | 2.0 | 1.0 | 0.864254 | 0.257685 |

****

1. **ICMP**

Table 38– Weighted Ensemble with different weights ,Mean ,std

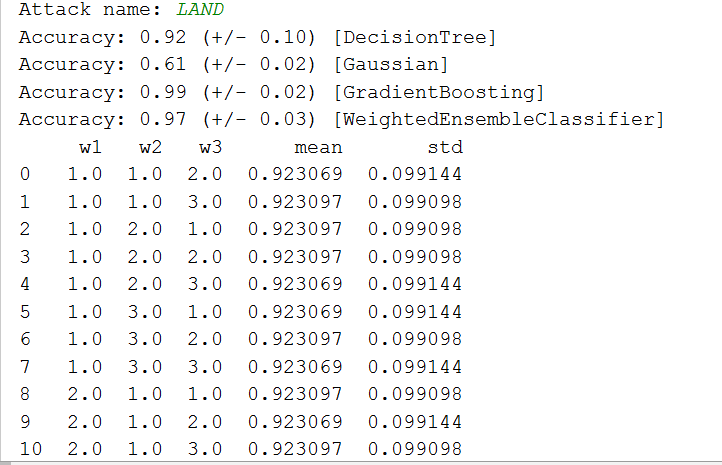
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Algorithm 1** | **Algorithm 2** | **Algorithm 3** | **Mean** | **Std** |
| **1** | 1.0 | 1.0 | 2.0 | 0.986982 | 0.015785 |
| **2** | 1.0 | 1.0 | 3.0 | 0.969527 | 0.059873 |
| **3** | 1.0 | 2.0 | 1.0 | 0.912096 | 0.159424 |

****

1. **LAND**

Table 39– Weighted Ensemble with different weights ,Mean ,std

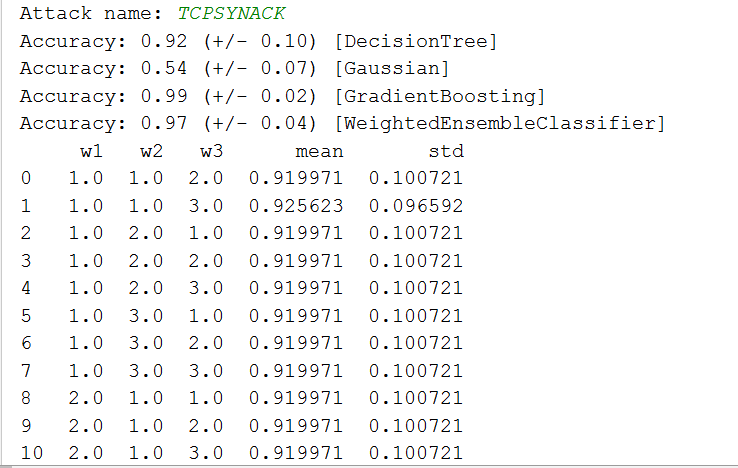
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Algorithm 1** | **Algorithm 2** | **Algorithm 3** | **Mean** | **Std** |
| **1** | 1.0 | 1.0 | 2.0 | 0.923097 | 0.099098 |
| **2** | 1.0 | 1.0 | 3.0 | 0.923069 | 0.099144 |
| **3** | 1.0 | 2.0 | 1.0 | 0.923097 | 0.099098 |

****

1. **TCPSYNACK**

Table 40– Weighted Ensemble with different weights ,Mean ,std

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Algorithm 1** | **Algorithm 2** | **Algorithm 3** | **Mean** | **Std** |
| **1** | 1.0 | 1.0 | 2.0 | 0.919971 | 0.100721 |
| **2** | 1.0 | 1.0 | 3.0 | 0.925623 | 0.096592 |
| **3** | 1.0 | 2.0 | 1.0 | 0.919971 | 0.100721 |

****

**\*\* Table 36-40 –Different weights in weighted ensemble means proportion of data gone into particular classifier.**

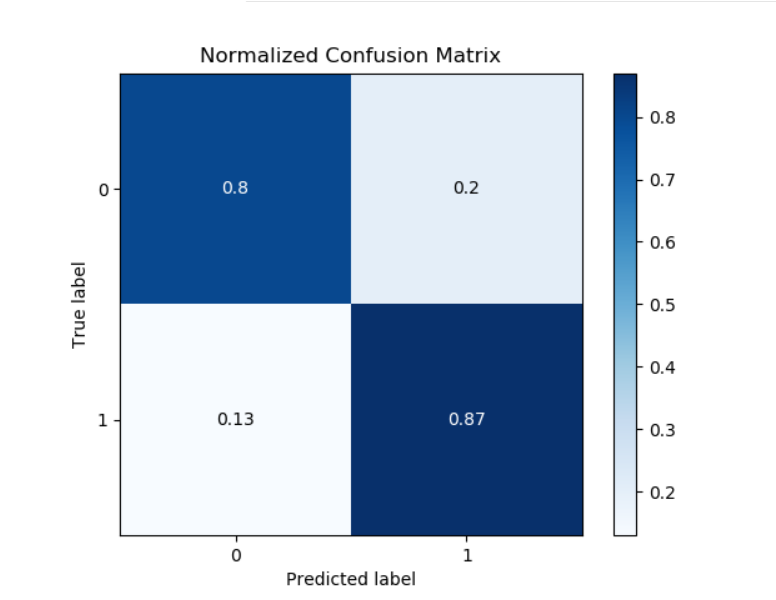
**High accuracy in weighted ensemble is because of same proportion of data in each classifier.**

**VISUALIZATION OF DATASET (Using Normalized Confusion Matrix)**

1. **UDP**

**Predicted Matrix from Training Data**

Figure 26 – Confusion Matrix for Weighted Ensemble for training data

****

**Weights=[clf1,clf2,clf3]=[Adaboost,DecisionTree,LogReg]**

**[1,1,1]**

Figure 27– Confusion Matrix for Weighted Ensemble for weights[1,1,1]

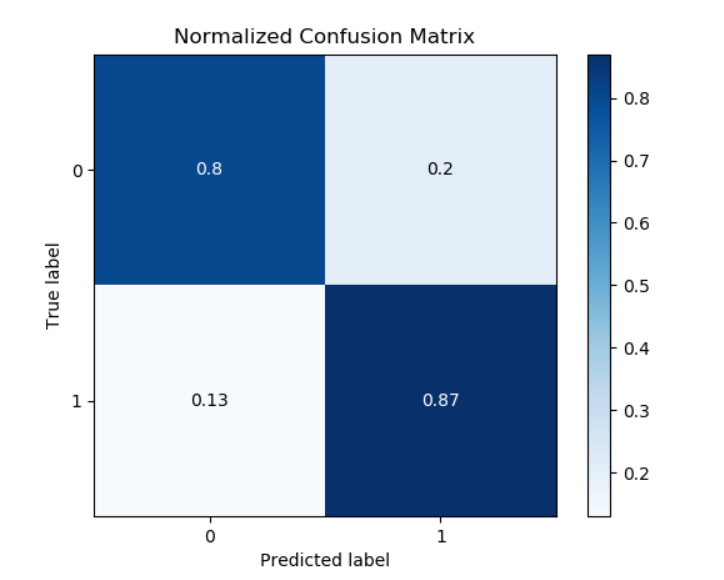
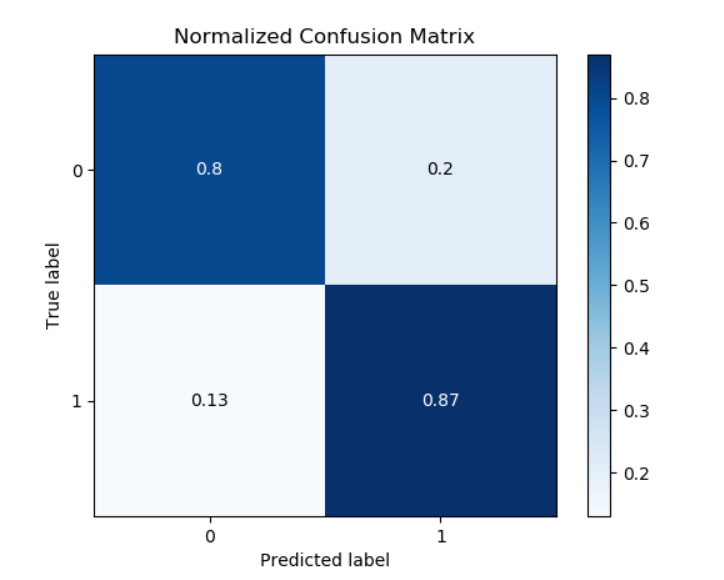
****

Figure 28 – Confusion Matrix for Weighted Ensemble for weights[1,2,,3]

****

Normalized confusion matrix helps to visually interpret how the labels are being predicted.

**Figure 26** plot shows the prediction of a ensemble classifier.

**Figure 27** plot shows prediction of ensemble classifier when weights are 1,1,1.

**TP-0.8**

**FP-0.2**

**TN-0.13**

**FN-0.87**

**Figure 28** plot shows prediction of ensemble classifier when weights are 1,2,3.

**TP-0.8**

**FP-0.2**

**TN-0.13**

**FN-0.87**

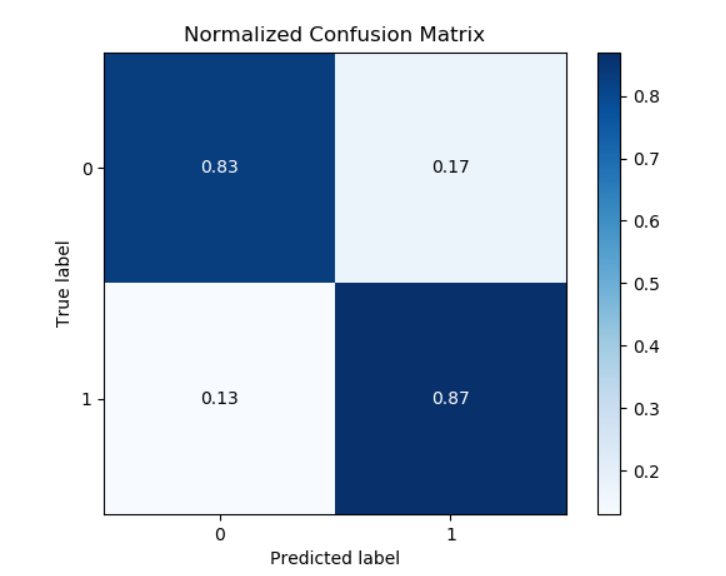
**\*\*No major difference on predicted labels can be seen by varying weights for UDP attack files.**

**\*\*Figure27-28 High TP values means high amount of predicted precised values.**

1. **TCPSYN**

**Predicted Matrix from Training Data**

Figure 29 – Confusion Matrix for Weighted Ensemble for training data

****

**Weights=[clf1,clf2,clf3]=[Adaboost,DecisionTree,LogReg]**

**[1,1,1]**

Figure 30 – Confusion Matrix for Weighted Ensemble for weights[1,1,1]

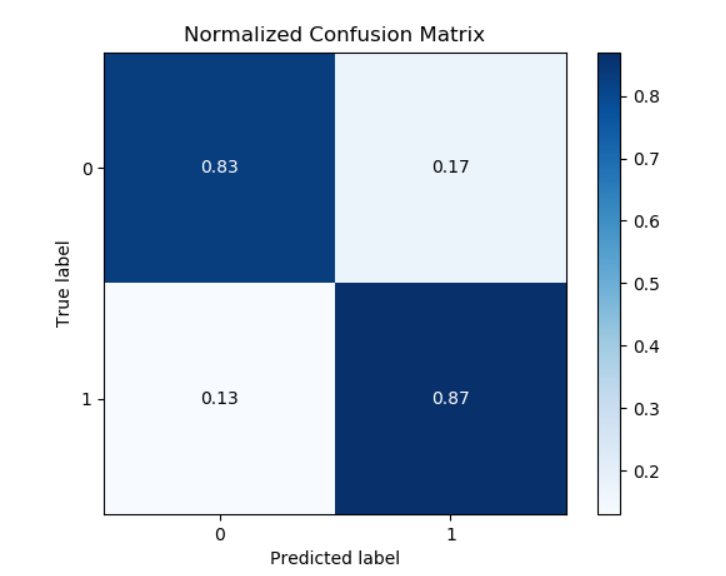
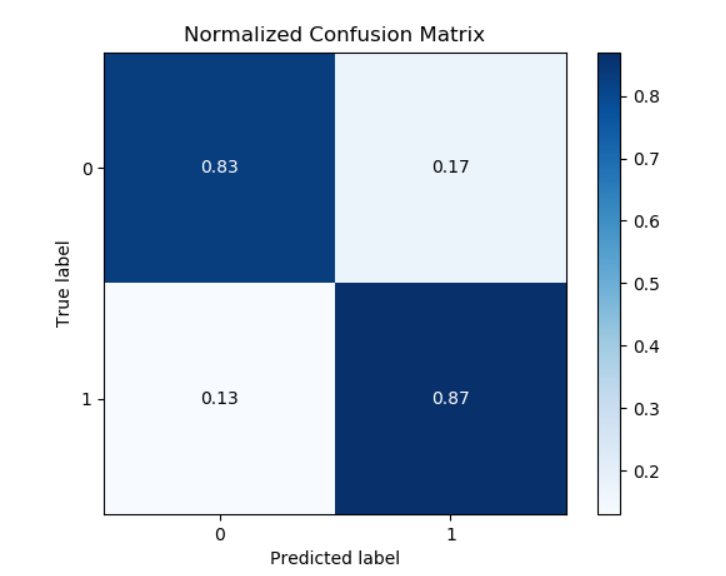
****

Figure 31 – Confusion Matrix for Weighted Ensemble for weights[1,2,,3]

****

**Figure 29** plot shows the prediction of a ensemble classifier.

**Figure 30** plot shows prediction of ensemble classifier when weights are 1,1,1.

**TP-0.83**

**FP-0.17**

**TN-0.13**

**FN-0.87**

**Figure 31** plot shows prediction of ensemble classifier when weights are 1,2,3.

**TP-0.83**

**FP-0.17**

**TN-0.13**

**FN-0.87**

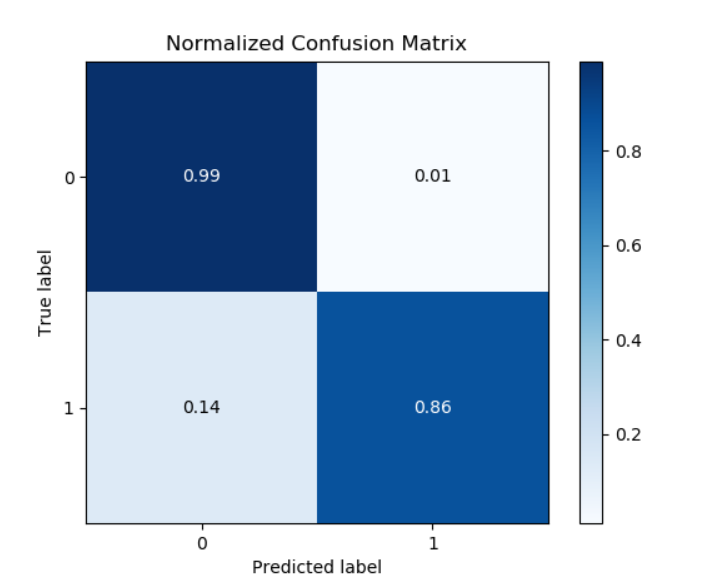
**\*\*No major difference on predicted labels can be seen by varying weights for UDP attack files.**

**\*\*Figure30-31 High FN values means high amount of non predicted not precised values..**

1. **ICMP**

**Predicted Matrix from Training Data**

Figure 32 – Confusion Matrix for Weighted Ensemble for training data

****

**Weights=[clf1,clf2,clf3]=[Adaboost,DecisionTree,LogReg]**

**[1,1,1]**

Figure 33 – Confusion Matrix for Weighted Ensemble for weights[1,1,1]

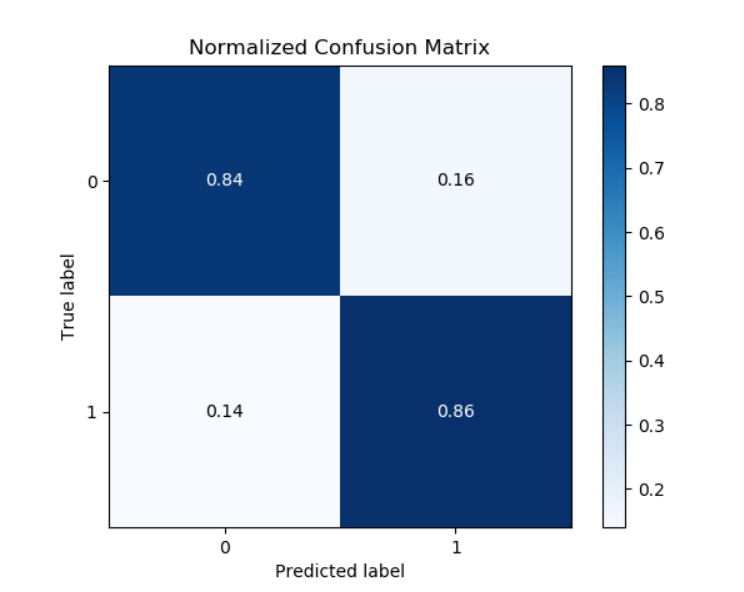
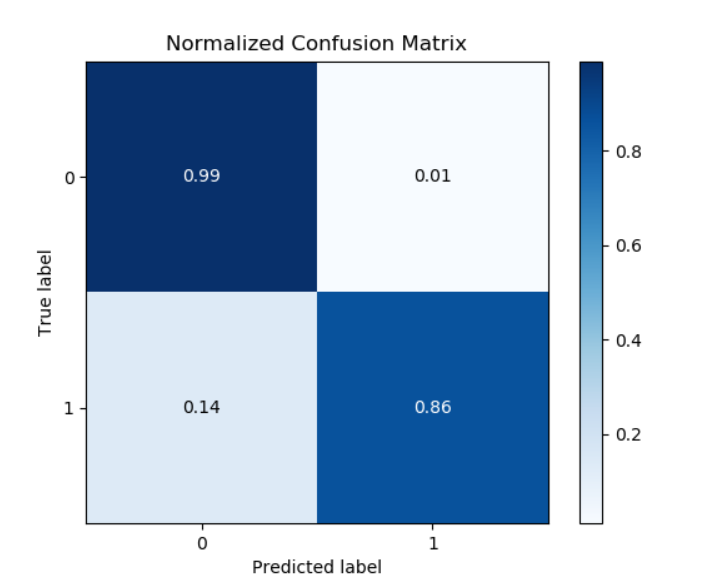
****

Figure 34 – Confusion Matrix for Weighted Ensemble for weights[1,2,,3]

****

**Figure 32** plot shows the prediction of a ensemble classifier.

**Figure 33** plot shows prediction of ensemble classifier when weights are 1,1,1.

**TP-0.84**

**FP-0.16**

**TN-0.14**

**FN-0.86**

**Figure 34** plot shows prediction of ensemble classifier when weights are 1,2,3.

**TP-0.99**

**FP-0.04**

**TN-0.14**

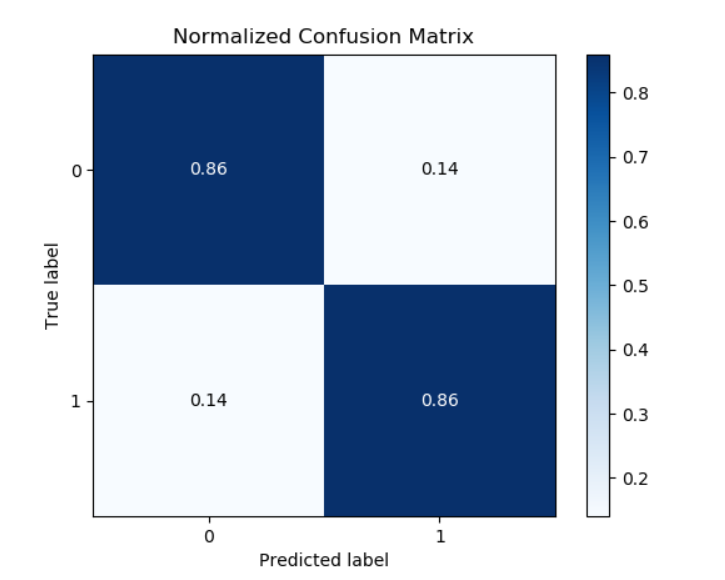
**FN-0.86**

**\*\*Figure33-34 High FN values means high amount of non predicted not precised values..**

1. **LAND**

**Predicted Matrix from Training Data**

Figure 35 – Confusion Matrix for Weighted Ensemble for training data.

****

**Weights=[clf1,clf2,clf3]=[Adaboost,DecisionTree,LogReg]**

**[1,1,1]**

Figure 36 – Confusion Matrix for Weighted Ensemble for weights[1,1,1]

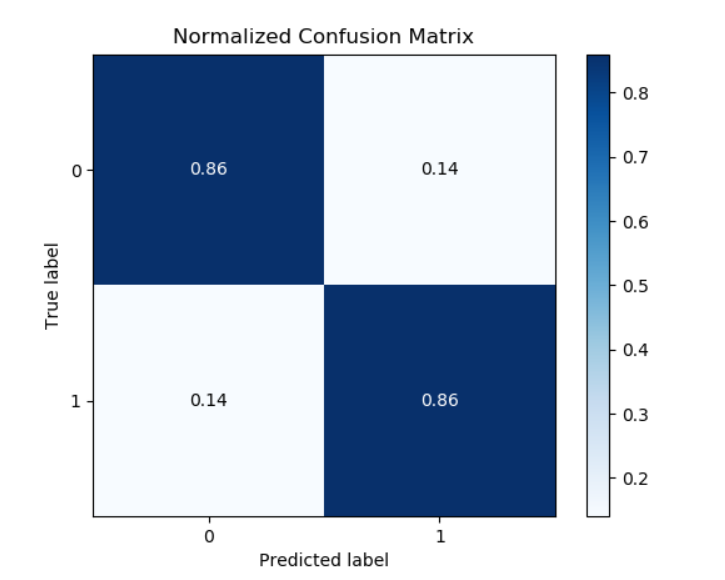
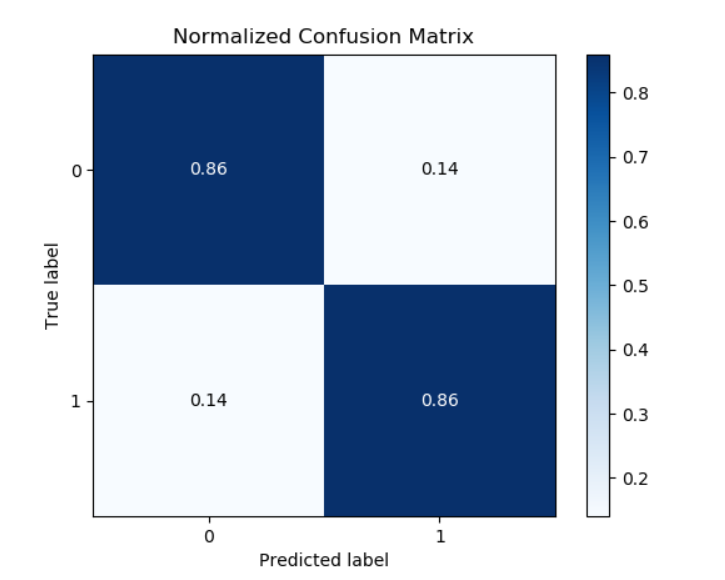
****

Figure 37 – Confusion Matrix for Weighted Ensemble for weights[1,2,,3]

****

**Figure 35** plot shows the prediction of a ensemble classifier.

**Figure 36** plot shows prediction of ensemble classifier when weights are 1,1,1.

**TP-0.86**

**FP-0.14**

**TN-0.14**

**FN-0.86**

**Figure 37** plot shows prediction of ensemble classifier when weights are 1,2,3.

**TP-0.86**

**FP-0.14**

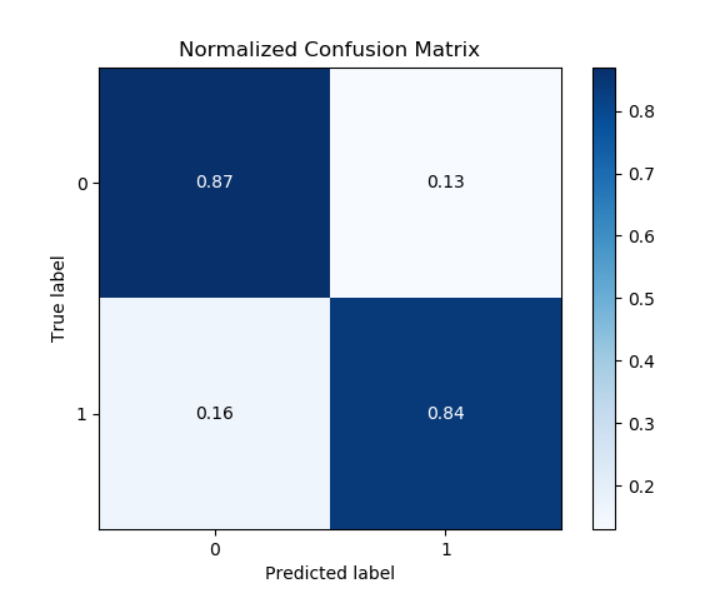
**TN-0.14**

**FN-0.86**

1. **TCPSYNACK**

**Predicted Matrix from Training Data**

Figure 38 – Confusion Matrix for Weighted Ensemble for training data

****

**Weights=[clf1,clf2,clf3]=[Adaboost,DecisionTree,LogReg]**

**[1,1,1]**

Figure 39– Confusion Matrix for Weighted Ensemble for weights[1,1,1]

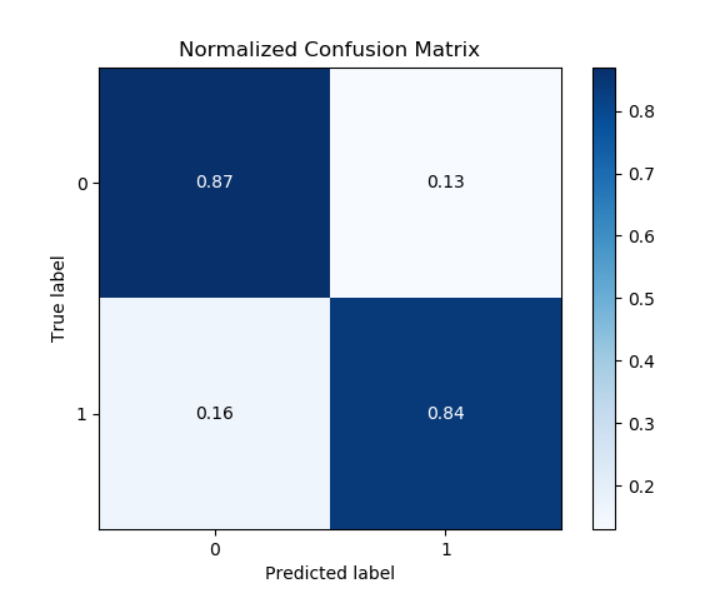
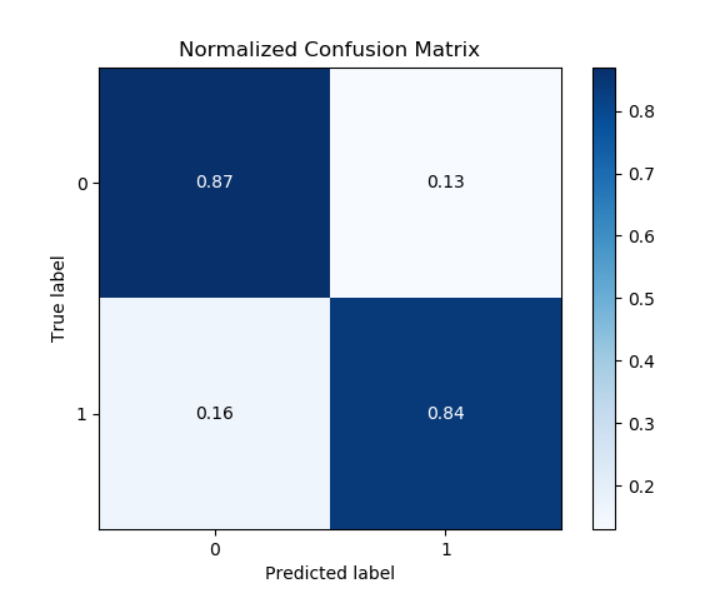
****

Figure 40 – Confusion Matrix for Weighted Ensemble for weights[1,2,,3]

****

**Figure 38** plot shows the prediction of a ensemble classifier.

**Figure 39** plot shows prediction of ensemble classifier when weights are 1,1,1.

**TP-0.87**

**FP-0.13**

**TN-0.16**

**FN-0.84**

**Figure 40** plot shows prediction of ensemble classifier when weights are 1,2,3.

**TP-0.87**

**FP-0.13**

**TN-0.16**

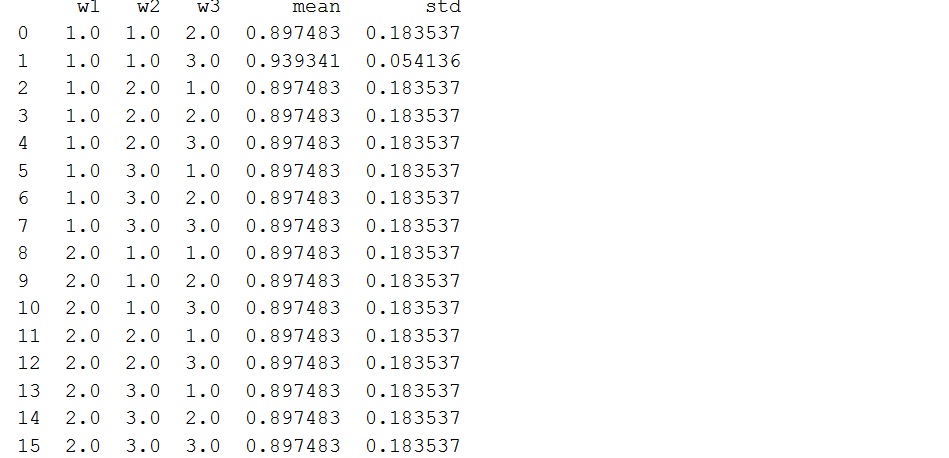
**FN-0.84**

**CLASSIFICATION BY VARYING PARAMETERS**

1. **UDP**

* **Varying Weights**

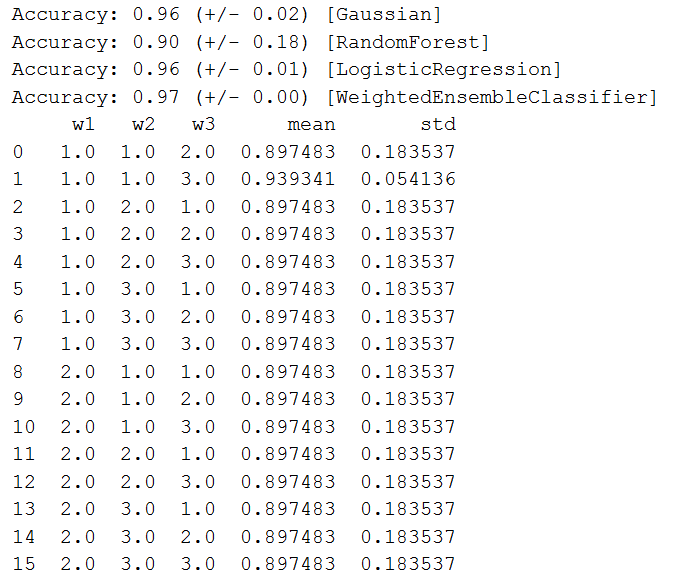
Figure 41

****

* **Varying Classifiers**

**Used – Gaussian ,RandomForest,Logistic Regression**

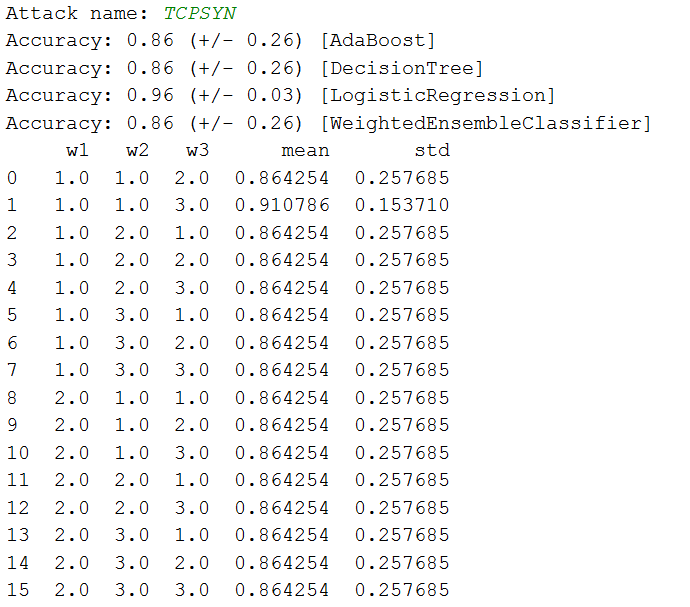
Figure 42

****

1. **TCPSYN**

* **Varying Weights**

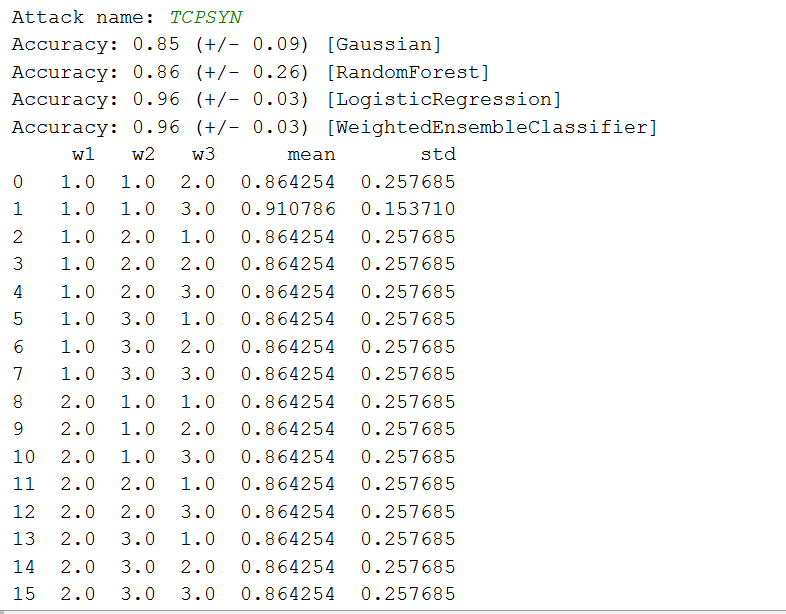
Figure 43

****

* **Varying Classifiers**

**Used – Gaussian ,RandomForest,Logistic Regression**

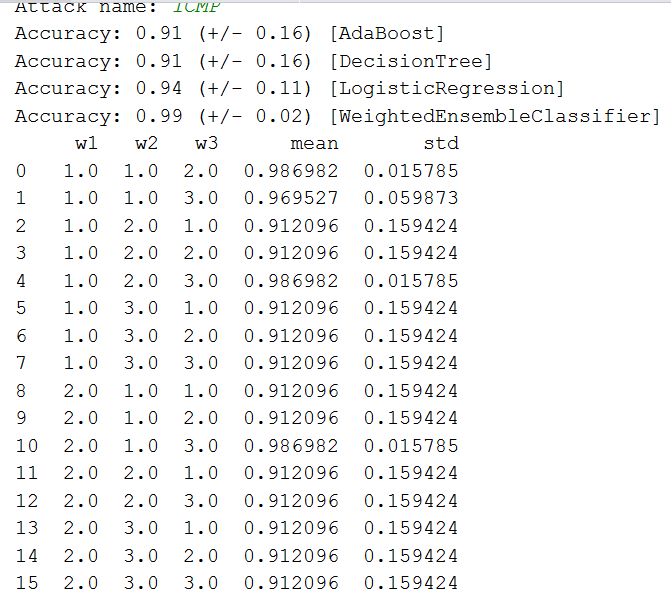
Figure 44

****

1. **ICMP**

* **Varying Weights**

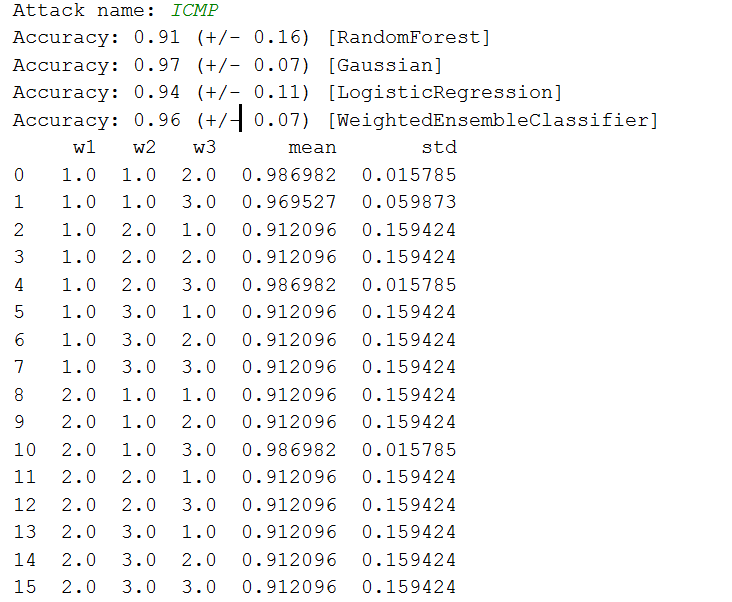
Figure 45

****

* **Varying Classifiers**

**Used – Gaussian ,RandomForest,Logistic Regression**

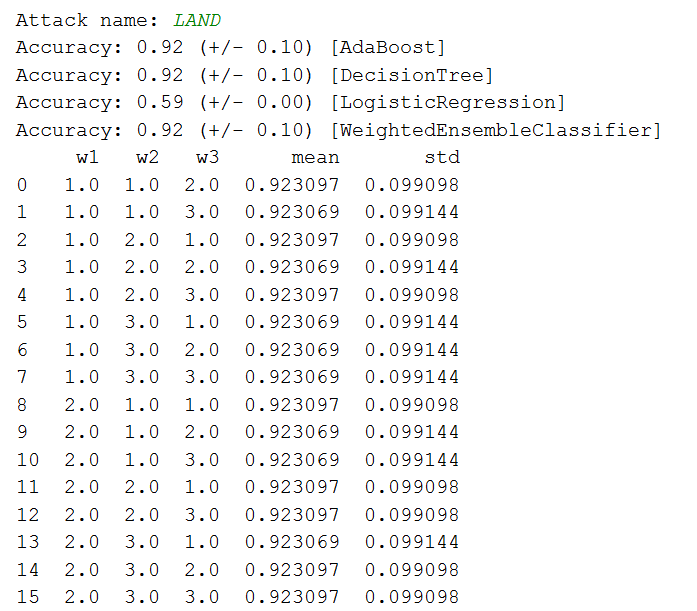
Figure 46

****

1. **LAND**

* **Varying Weights**

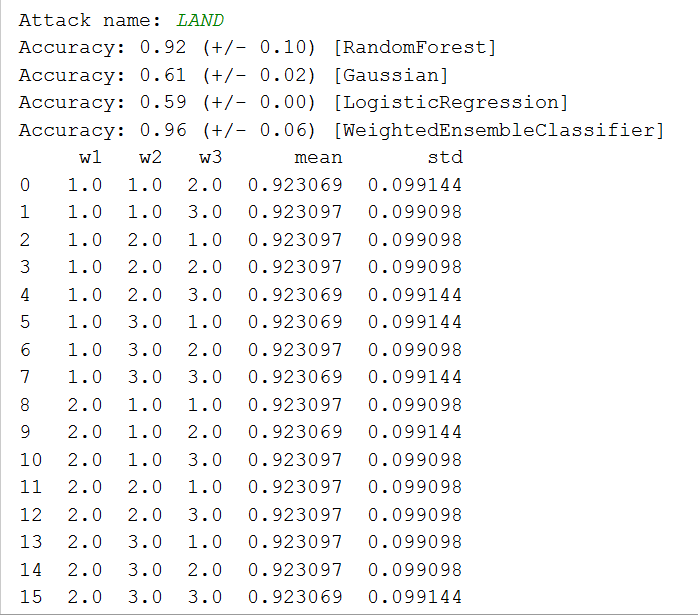
Figure 47

****

* **Varying Classifiers**

**Used – Gaussian ,RandomForest,Logistic Regression**

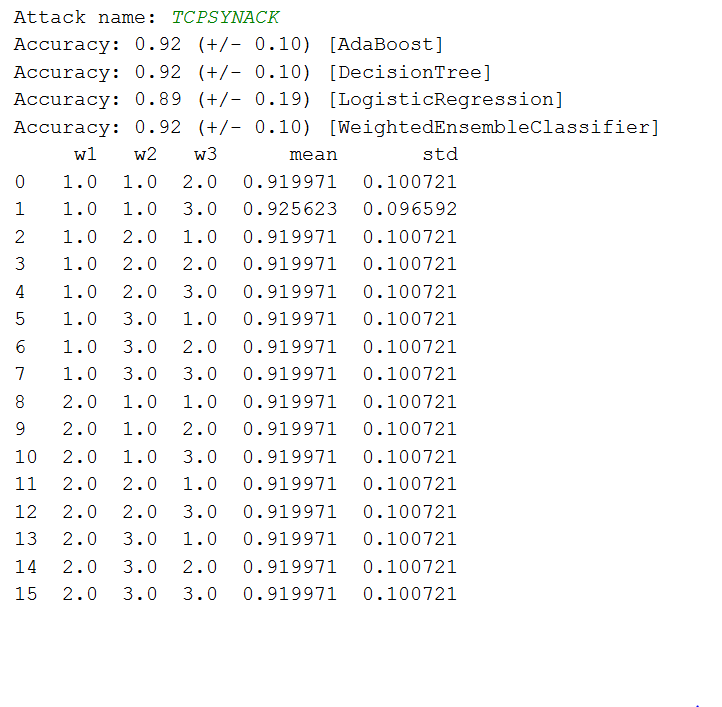
Figure 48

****

1. **TCPSYNACK**

* **Varying Weights**

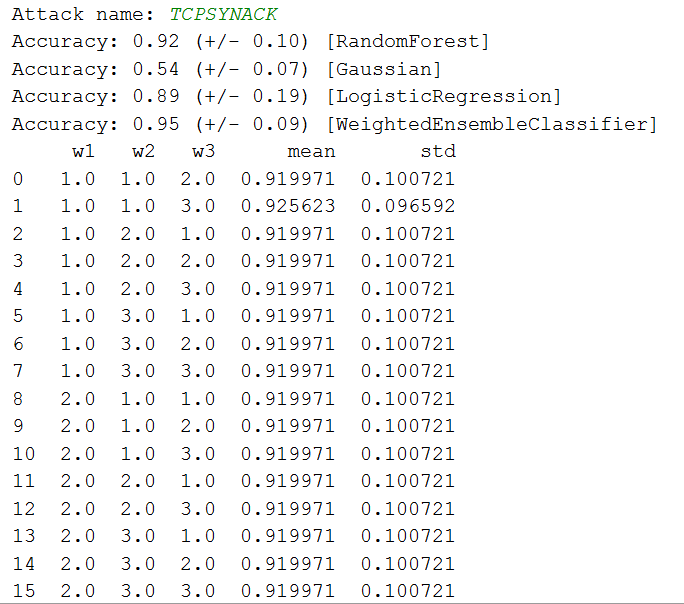
Figure 49

****

* **Varying Classifiers**

**Used – Gaussian ,RandomForest,Logistic Regression**

Figure 50

****

**CONCLUSION**

By using different classification algorithms and different ensemble methods we predicted for all attack files the accuracy and also data which belong to Flood class or Normal Class.

Data in Flood class is attacked data .It can be a DOS attack or other communication attack. Dataset contains different attributes like Cumulative bytes, Port address, Source address, destination address etc on the basis of which flood and normal class has been identified.

Our job was to predict the accuracy of classification into two different classes.

**Learning-**

We got to learn about different classifiers and ensemblers , nature of dataset and how different values of attributes gives different results of classification.

How varying parameters can vary accuracies of classifiers and how for different classifiers in ensemble can also vary result.

**Inferences:-**

**UDP-**

As individual **gradient boosting** shows the maximum accuracy whereas while ensembling **major voting** shows best result.

In UDP attack file individual classifier works better than ensemble.

**TCPSYN-**

As individual **gradient boosting** shows the maximum accuracy whereas while ensembling **major voting** shows best result.

In TCPSYN attack file individual classifier works better than ensemble.

**ICMP-**

As individual **gradient boosting** shows the maximum accuracy whereas while ensembling **major voting** shows best result.

In ICMP attack file individual classifier works better than ensemble.

**LAND-**

As individual **decision tree** shows the maximum accuracy whereas while ensembling **major voting** shows best result.

In LAND attack file individual classifier works better than ensemble.

**TCPSYNACK-**

As individual **decision tree** shows the maximum accuracy whereas while ensembling **major voting** shows best result.

In TCPSYNACK attack file individual classifier works better than ensemble.

\*\*Individual classifier work better than ensemble because ensemble works on the prediction model of individual classifier whereas individual works on the train and test data carried out separately by classifier function.