Report on

End-to-end project delivery on cyber-security data analytics

SIT 719

Prateek Singh

221218743

Table of Contents

1 – Introduction

Artificial Intelligence is a growing field which finds its application into a plethora of domains. One of such domains is cyber security, various applications like intrusion-detection, spam classification, malware detection etc. are now being driven by Artificial Intelligence algorithms. The major reason for this is the easy access to increased computational capabilities and large amount of organically generated data. Algorithms developed using the insights from this large amount of data are generally more robust and generic than hand crafted rules. Artificial Intelligence algorithms can be subdivided into many sub-classes, one such division is machine learning and deep learning algorithms. While there are many differences between the two, the major difference is the feature extraction or selection step. Deep learning algorithms generally operate on raw data while extracting features and learning from them, whereas machine learning algorithms work on features fed into them by a preceding feature extraction step.

The purpose of this experiment is to evaluate and establish the efficacy of machine learning algorithms for developing end to end cyber-attack classification systems e.g., intrusion detection systems. This experiment also touches upon a basic Deep Learning Model (Multi-Layer Perceptron) but does not delve deep into the domain of exploring architectures of deep learning approaches. All the algorithms used in this experiment are from scikit-learn library and Python has been used as the language of implementation. This report is divided into three sections:

1. **Introduction**: This section defines the problem statement and the purpose of the experiment.
2. **Overview**: This section explores the algorithms used and dives into the underlying principle and parameters used for every algorithm.
3. **Results**: Here, results obtained on the dataset(s) used are compared. This section also explores the advantages and disadvantages of every algorithm and their effectiveness for the problem at hand. This section also contains closing remarks based on the results obtained and explains the understanding of the used algorithms.

2 – Overview

Two datasets have been used for benchmarking the performance of machine learning algorithms, KDD dataset used in [1] and TON IoT Telemetry Dataset used in [2]. Both the datasets have labels denoting potential attack scenario and normal scenario. KDD dataset is a multiclass (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R) and TON IoT Telemetry is a binary class dataset. As both the datasets are labelled thus making this a classification scenario, we benchmark them against a set of supervised learning algorithms [K nearest neighbours, Logistic Regression, SVM, Decision Tree, Random Forest, AdaBoost and Multi-Layer Perceptron] from scikit-learn library. All these algorithms are evaluated on same set of metrics:

1. Accuracy: (TP + TN) / All Data
2. Precision: TP / (TP + FP)
3. Recall: TP / (TP + FN)
4. F1-Score: (2 * Recall * Precision) / (Recall + Precision)
5. FPR: FP / (FP + TN)
6. Train Time (Model training)
7. and Test Time (Prediction) for both the datasets

TP: Model predicts attack when there is actual attack. In multiclass scenario, the definition is extended to correctly predicting the type of attack.

FP: Model predicts attack when there is no attack. In multiclass scenario, the definition changes to include inter class misclassification.

TN: Model predicts normal or benign scenario when there is no attack.

FN: Model predicts normal or benign scenario when there is actual attack.

Experiments:

K Nearest Neighbours [4]: K Nearest Neighbours is a supervised classification algorithm which finds its way into several classification scenarios. The algorithm uses the principle that representation of any data must be closer in distance to a similar datapoint for a given feature space. Using this idea, a new data sample is assigned the class label it is closest to, based on the number of voters which is regulated by the parameter K. The popular implementations of KNN algorithms use Euclidean Distance as the distance metric to calculate similarity between two feature vectors.

This also means that there is no model or weights which are trained, and the algorithms just stores the instances during learning and during prediction distance based on majority votes is used to assign class labels. This is also known as instance based learning or lazy learning.

Algo:

1. The parameter K is selected denoting the number of neighbours or voters to consider.
2. Choose the top K neighbours from the input datapoint based on Euclidean distance.
3. The class which has maximum number of samples present in top K neighbours becomes the class label for the input data.
4. Key Param: K – Denotes the number of neighbours to be selected.

Following tables show the evaluation metrics values on both the datasets for KNN.

Attack Class	Precision	Recall	F1-Score	Support
Benign	66%	97%	79%	9711
DoS	96%	78%	86%	7636
Probe	83%	67%	74%	2423
R2L	95%	7%	12%	2574
U2R	90%	4%	9%	200

Table 1.1: *Evaluation metrics for each class obtained from KNN on KDD dataset.*

Attack Class	Precision	Recall	F1-Score	Support
Normal	82%	93%	87%	73495
Attack	86%	68%	76%	46841

Table 1.2: *Evaluation metrics for TON IoT Telemetry Dataset obtained from KNN.*

Following figures show the confusion matrices on both the datasets for KNN.

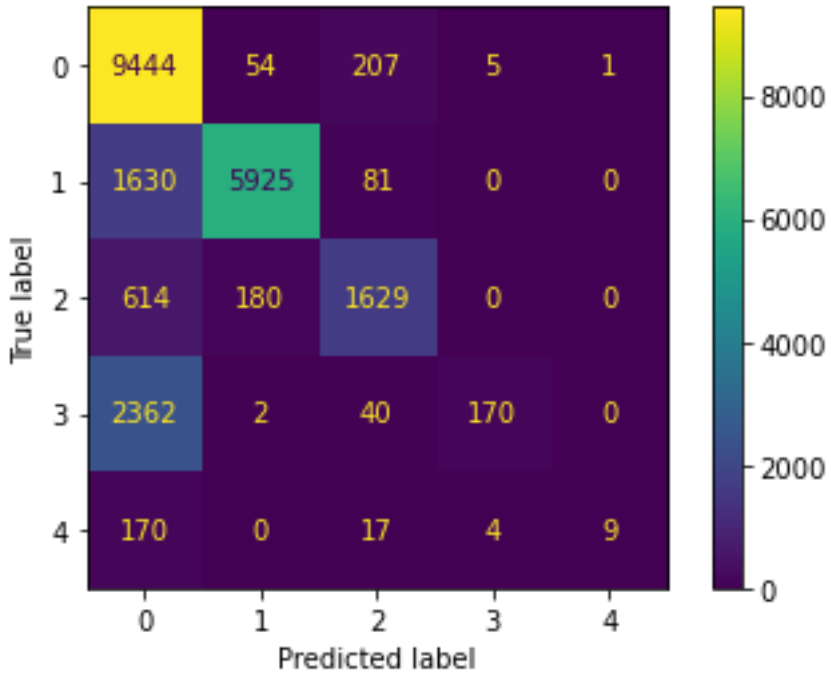

Fig 1.1: Confusion Matrix for KNN on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)

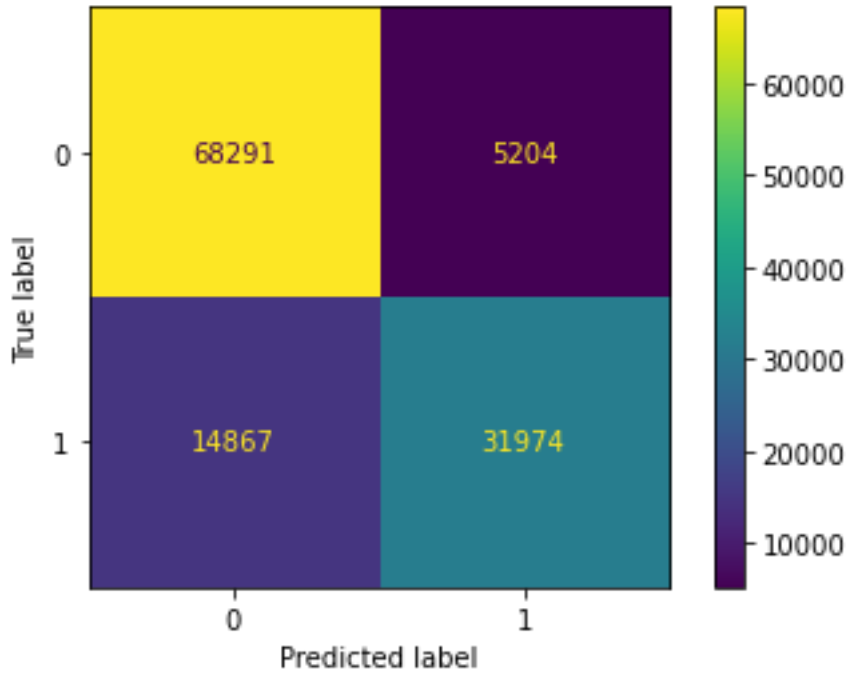

Fig 1.2: Confusion Matrix for KNN on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

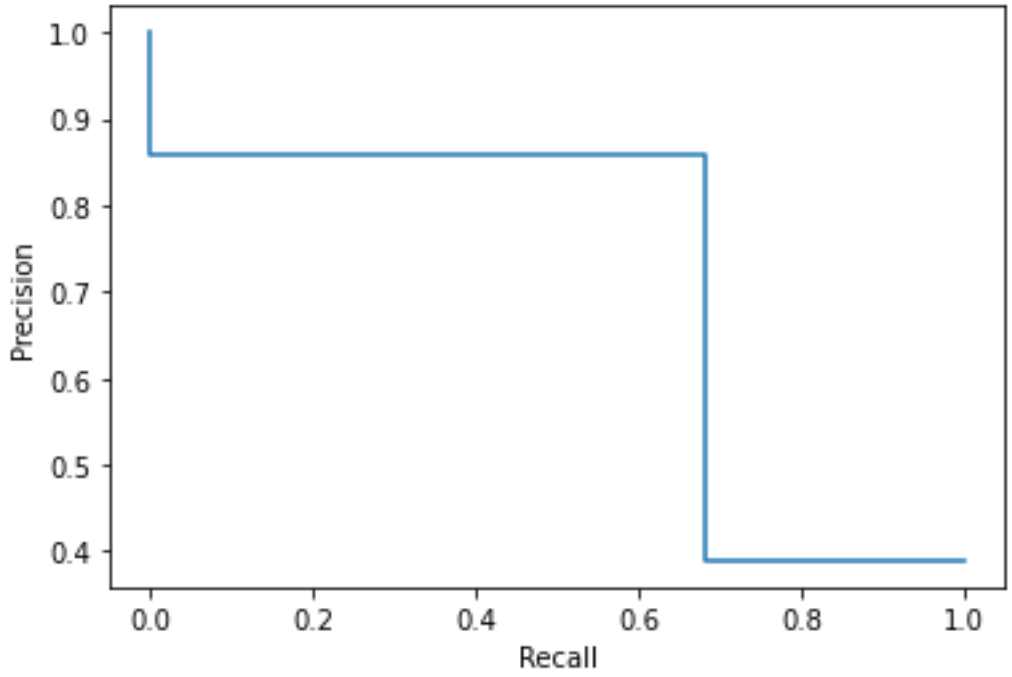

Fig 1.3: Precision-Recall Curve for KNN *on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)*

Logistic Regression [6]: Logistic Regression is a supervised classification algorithm which seems to have a misleading name but has a well explained reason behind its nomenclature. In Logistic Regression, we perform classification by having a threshold on probability, this allows us to have a class label based on the probability value obtained.

The underlying process for both the linear regression and logistic regression is similar, the difference arises during the training process where in linear regression the objective is to fit the regression line whereas in logistic regression, we fit a logistic (sigmoid) function which yields a value ranging between 0 and 1. This value is responsible for the class label assigned to the input data point.

Algo:

1. Training involves fitting logistic (sigmoid) function on the training set.
2. Use of gradient descent or similar algorithm to minimize the cost function.
3. Probability is calculated for the input data point (test data).
4. A threshold is set on probability to predict the assigned class label for input data point.
5. Key Params:
    a. criterion – denotes the splitting criteria to be used for attribute selection.
    b. max_depth – determines the complexity of the tree by limiting the depth of the tree.

Following tables show the evaluation metrics values on both the datasets for Logistic Regression.

Attack Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign	66%	93%	77%	9711
DoS	97%	79%	87%	7636
Probe	74%	75%	75%	2423
R2L	91%	4%	7%	2574
U2R	80%	4%	8%	200

Table 1.3: *Evaluation metrics for each class obtained from* Logistic Regression *on KDD dataset.*

Attack Class	**Precision**	**Recall**	**F1-Score**	**Support**
Normal	67%	98%	79%	73495
Attack	86%	24%	37%	46841

Table 1.4: *Evaluation metrics for TON IoT Telemetry Dataset obtained from* Logistic Regression.

Following figures show the confusion matrices on both the datasets for Logistic Regression.

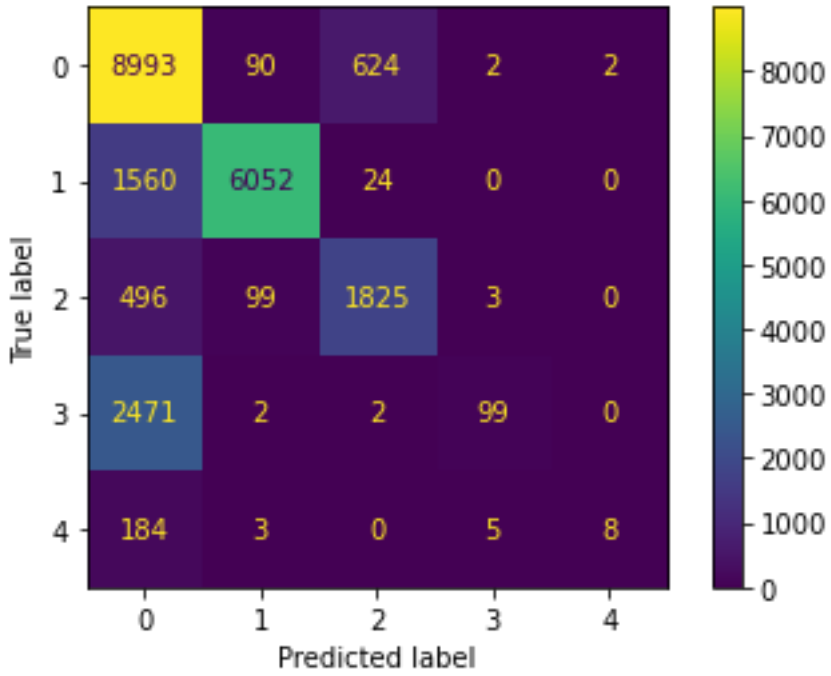

Fig 1.4: Confusion Matrix for Logistic Regression *on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)*

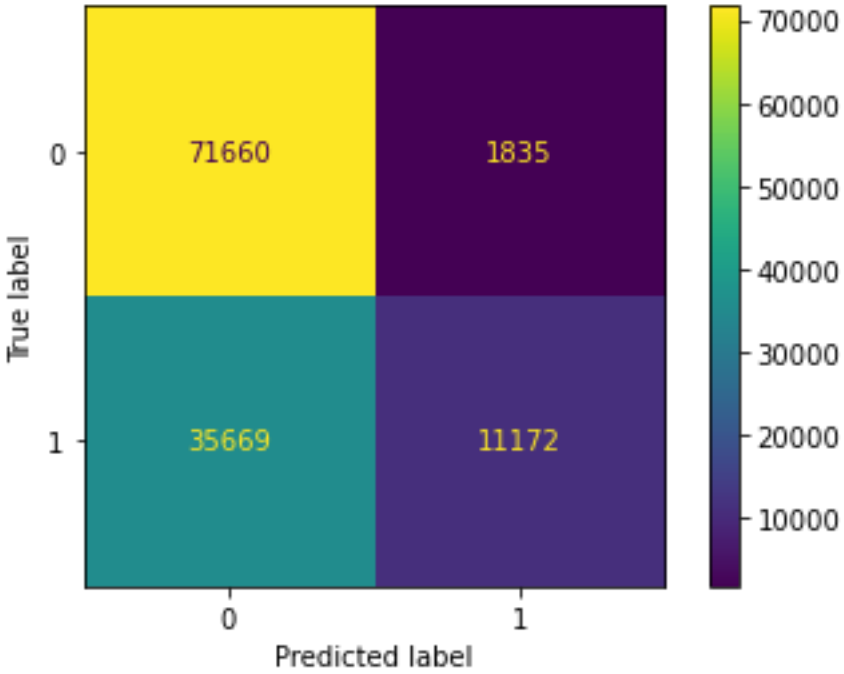

Fig 1.5: Confusion Matrix for Logistic Regression *on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)*

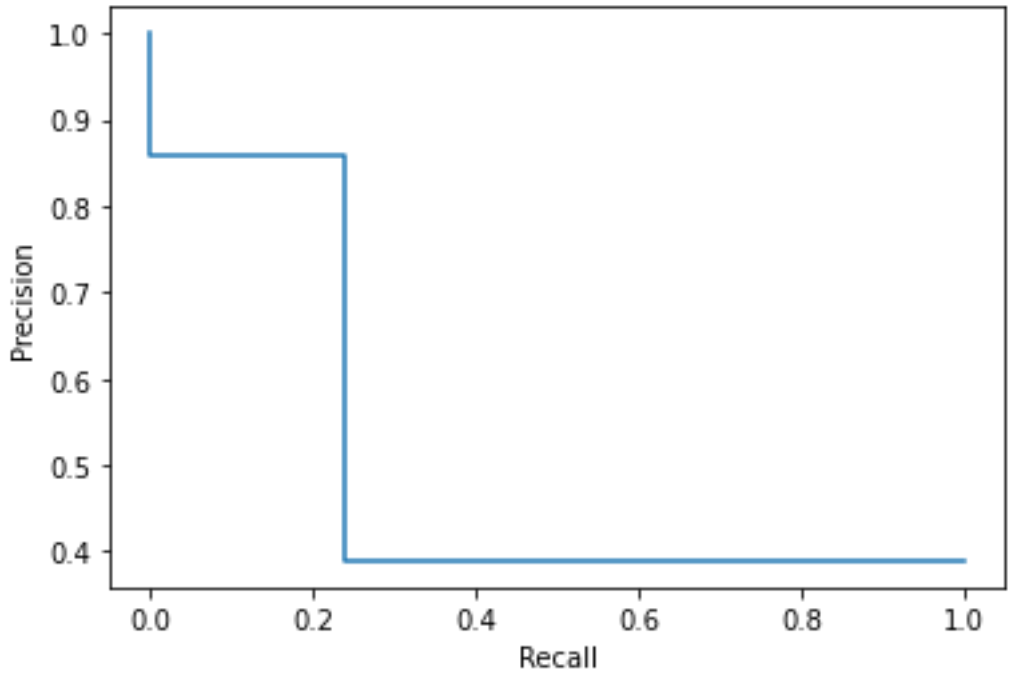

Fig 1.6: Precision-Recall Curve for Logistic Regression *on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)*

SVM [3]: SVM is a supervised learning algorithm which stands for Support Vector Machine, SVM has two variants, regressor and classifier used for both regression and classification respectively. The underlying principle of SVM is to calculate a hyperplane that can approximate a boundary function to effectively separate data points. The data points that lie on the edge of their respective classes are known as the support vectors. The driving idea behind SVM is to maximize the margin between the support vectors, this allows the best separation between the classes.

Another key highlight of SVM is the popular kernel trick, this allows us to choose a nonlinear kernel which transforms the data into higher dimension which can help in finding better approximations for boundary functions in case of data that is not linearly separable. This makes SVMs a powerful tool even in scenarios where other linear classifiers or regressors struggle.

Algo:

1. Find the boundaries between the support vectors.
2. Choose the boundary that has maximum margin between the support vectors.
3. Key Params:
    a. C – Penalty term for regularization
        i. SVMs are prone to overfitting, this makes choosing best C an essential step.
    b. Kernel – Allows the choice between a linear or a nonlinear (rbf, sigmoid etc.) kernel, by default rbf is used in sklearn SVM.

Following tables show the evaluation metrics values on KDD dataset for SVM.

Attack Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign	64%	97%	78%	9711

DoS	96%	75%	84%	7636
Probe	84%	58%	69%	2423
R2L	99%	10%	18%	2574
U2R	100%	3%	5%	200

Table 1.5: *Evaluation metrics for each class obtained from* SVM *on KDD dataset.* ***Possible overfitting****.*

Following figures show the confusion matrices on KDD dataset for SVM.

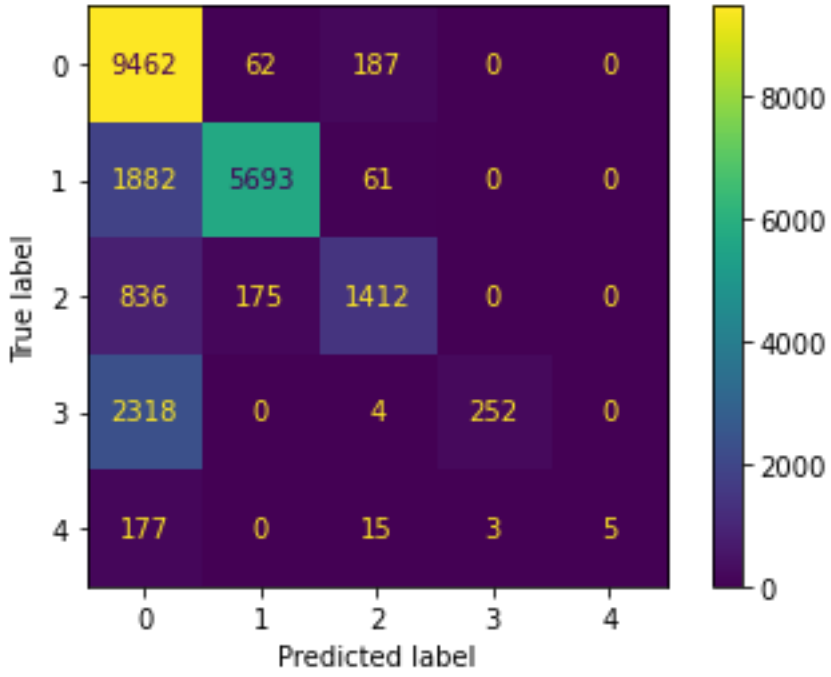

Fig 1.7: Confusion Matrix for SVM *on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)*

**SVM not implemented for TON IoT Telemetry dataset due to computational constraints.*

Decision Tree [3]: Decision Tree is a supervised learning algorithm which can be used for regression as well as classification problems. In classification problems, each leaf node of a fully formed Decision Tree represents a class. This is a rule-based classification technique in which the branches denote the subset which differs from remaining data by certain splitting criteria.

The driving principle of this algorithms is the attribute selection step, generally Information Gain or Gini Index is used to select the best attribute for splitting. Information Gain is nothing but the change in entropy when the tree is split into subsequent sub trees. The other popular method for attribute selection is Gini Index (default value in sklearn), this is the measure of probability of any specific feature when chosen randomly will end up being classified incorrectly. The attribute with the least value of Gini Index is chosen for splitting.

Algo:

1. The decision tree classification or regression process starts with the whole training data as a root node.
2. Next step is to calculate or select a node which can be used to split the data based on a given rule.
3. This step is repeated until we can not split the dataset further.

4. After the process is completed, we have leaf nodes as the class labels and all the other nodes act as decision nodes.

Following tables show the evaluation metrics values on both the datasets for Decision Tree.

Attack Class	Precision	Recall	F1-Score	Support
Benign	67%	96%	79%	9711
DoS	96%	79%	86%	7636
Probe	79%	63%	70%	2423
R2L	98%	11%	19%	2574
U2R	79%	7%	14%	200

Table 1.6: *Evaluation metrics for each class obtained from Decision Tree on KDD dataset.*

Attack Class	Precision	Recall	F1-Score	Support
Normal	85%	94%	90%	73495
Attack	89%	74%	8%	46841

Table 1.7: *Evaluation metrics for TON IoT Telemetry Dataset obtained from Decision Tree.*

Following figures show the confusion matrices on both the datasets for Decision Tree.

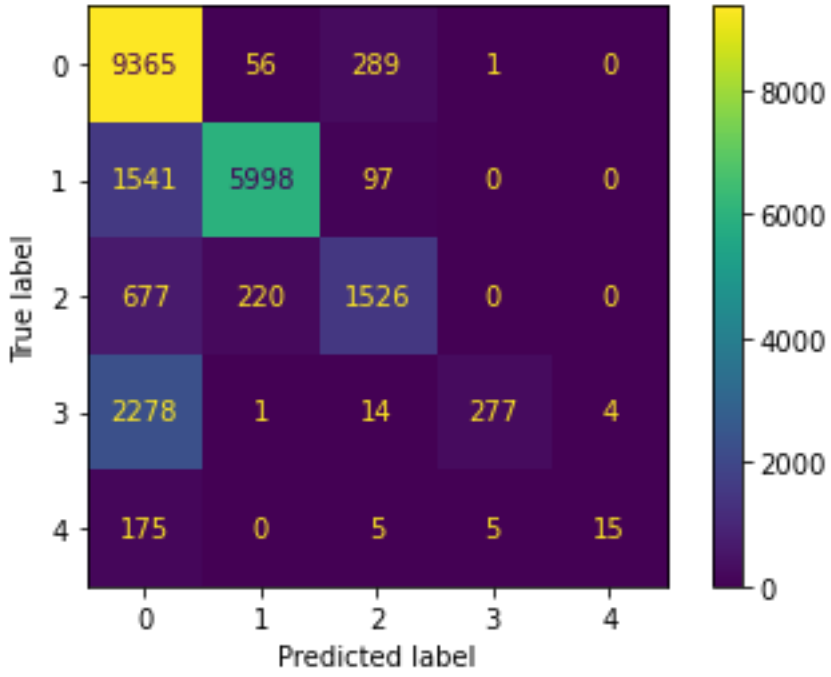

Fig 1.8: Confusion Matrix for Decision Tree on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)

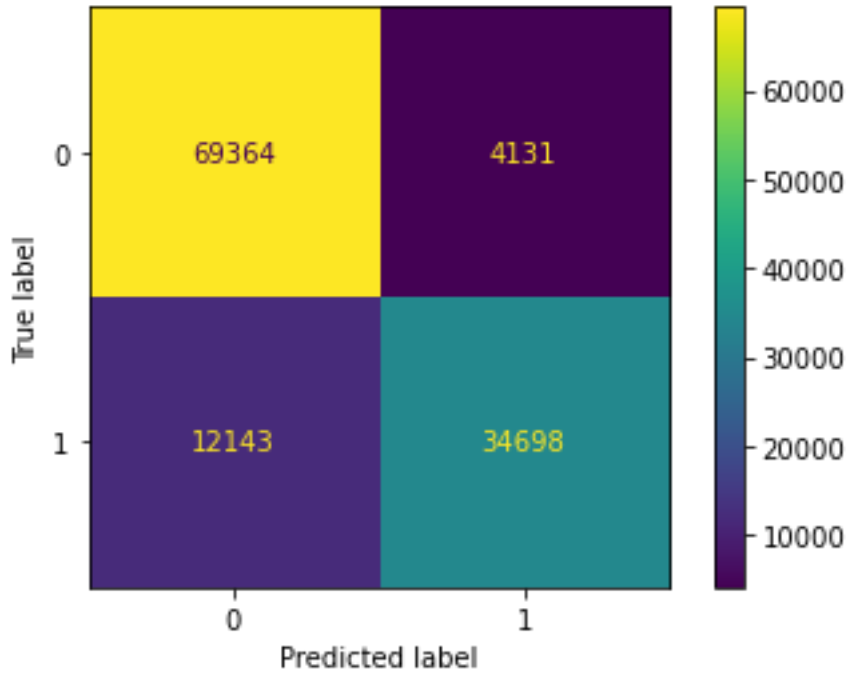

Fig 1.9: Confusion Matrix for Decision Tree on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

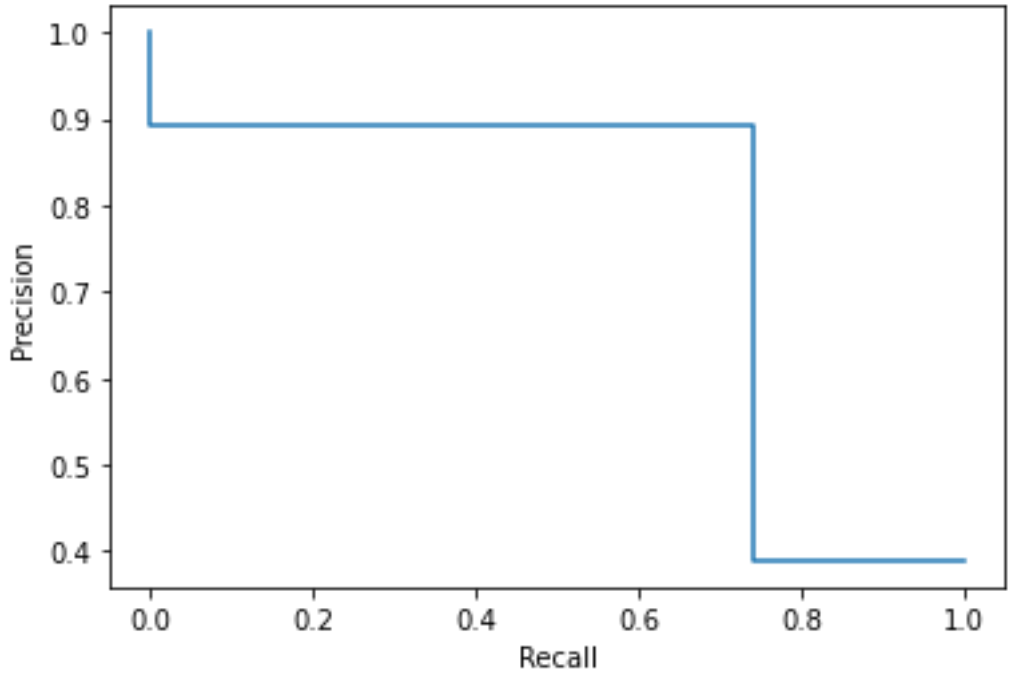

Fig 1.10: Precision-Recall Curve for Decision Tree on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

Random Forest [3]: Random Forest is an ensemble technique that can be used for both classification and regression problems. This approach works on the principle that the combined strength of the group overpowers the strength of individual. Using this idea, Random Forest uses a collection of decision trees to create a model that is superior to the constituent decision trees. Generally, the rule of thumb is to have weak decision trees as a constituent so that the ensemble is a generalised model.

To ensure that this happens Random Forest approach uses decision trees that are trained only on a subset of the entire dataset and the collection equals the whole dataset.

Algo:

1. Divide training set into subsets based on the number of trees.
2. When incoming data point is encountered then the majority vote is considered, the output label which gets the maximum votes from the constituent trees is chosen as the final class label.
3. **Key Params:**
    a. **n_estimators:** The number of trees in the Forest.
    b. **Criterion:** Like Decision Trees, attribute selection param.

Decision Trees are prone to overfitting as they are a greedy algorithm, Random Forest try to solve that problem by training weak Decision Trees on only a subset of data. This makes Random Forest powerful as they become generalised and yield high accuracy on unseen data.

Following tables show the evaluation metrics values on both the datasets for Random Forest.

Attack Class	Precision	Recall	F1-Score	Support
Benign	64%	97%	77%	9711
DoS	96%	74%	84%	7636
Probe	86%	63%	73%	2423
R2L	98%	5%	10%	2574
U2R	60%	1%	3%	200

Table 1.8: *Evaluation metrics for each class obtained from Random Forest on KDD dataset.*

Attack Class	Precision	Recall	F1-Score	Support
Normal	85%	95%	90%	73495
Attack	91%	74%	82%	46841

Table 1.9: *Evaluation metrics for TON IoT Telemetry Dataset obtained from Random Forest.*

Following figures show the confusion matrices on both the datasets for *Random Forest*.

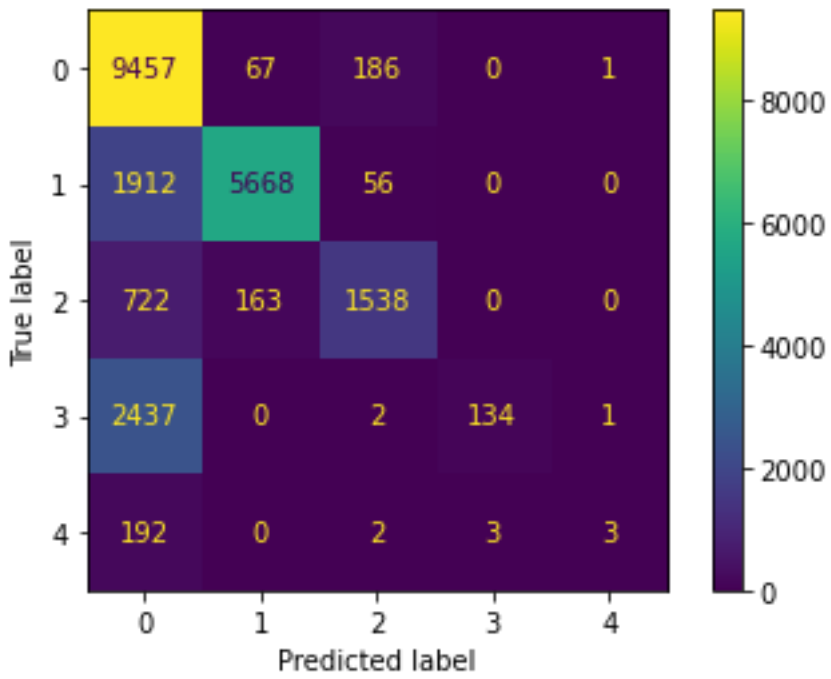

Fig 1.11: Confusion Matrix for Random Forest on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)

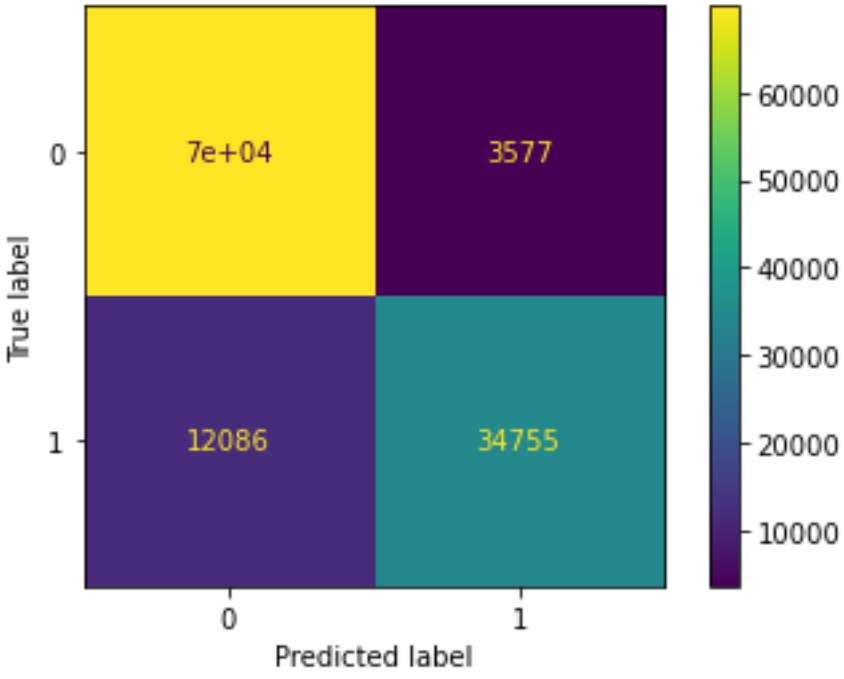

Fig 1.12: Confusion Matrix for Random Forest on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

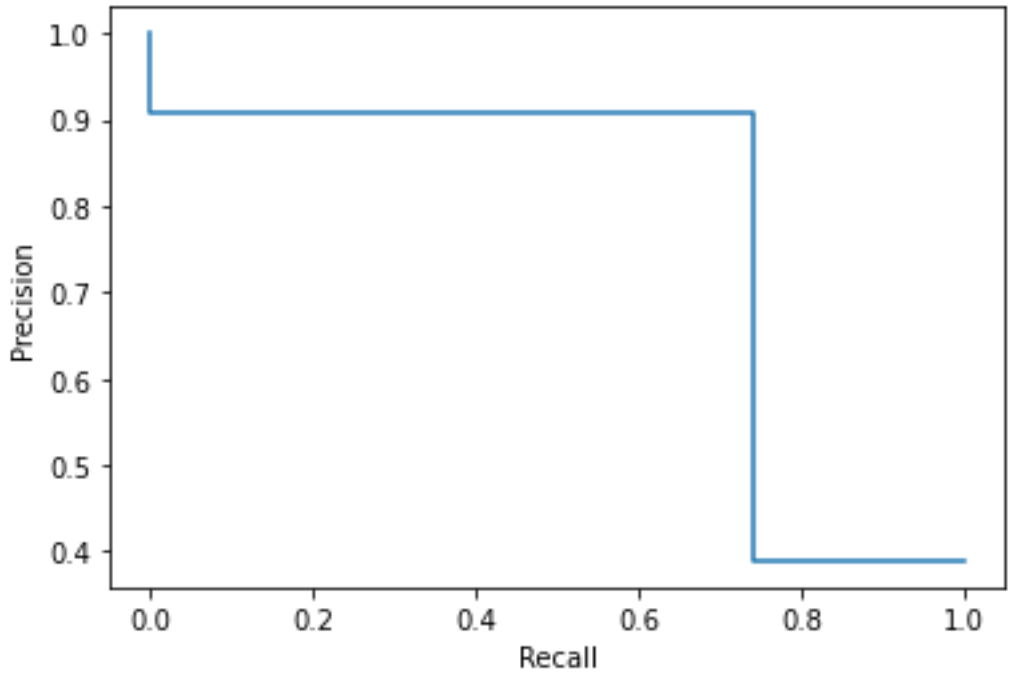

Fig 1.13: Precision-Recall Curve for Random Forest on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

AdaBoost: AdaBoost or Adaptive Boosting, this is an example of ensemble learning. This approach works on the principle of *boosting* the accuracy or building a strong classifier from a collection of weak classifiers. Unlike Random Forest AdaBoost can be used with any Machine Learning algorithm, although the rule of thumb is to have a collection of weak classifiers.

The driving principle behind this approach is to build classifiers in such a way that each subsequent classifier tries to correct its predecessor. This results in a strong and robust classifier.

Algo:

1. Dataset is divided in a way that each data point has equal weight.
2. The initial dataset is then fed into the model.
3. The misclassified points are identified and fed into the model till all the points are correctly classified.
4. Another stopping criteria is that the max number of models have been trained.

Following tables show the evaluation metrics values on both the datasets for *AdaBoost*.

Attack Class	Precision	Recall	F1-Score	Support
Benign	65%	81%	72%	9711
DoS	67%	60%	63%	7636
Probe	52%	71%	60%	2423
R2L	57%	1%	1%	2574
U2R	1%	3%	2%	200

Table 1.10: *Evaluation metrics for each class obtained from AdaBoost on KDD dataset.*

Attack Class	Precision	Recall	F1-Score	Support
Normal	77%	96%	86%	73495
Attack	90%	55%	69%	46841

Table 1.11: *Evaluation metrics for TON IoT Telemetry Dataset obtained from AdaBoost.*

Following figures show the confusion matrices on both the datasets for *AdaBoost*.

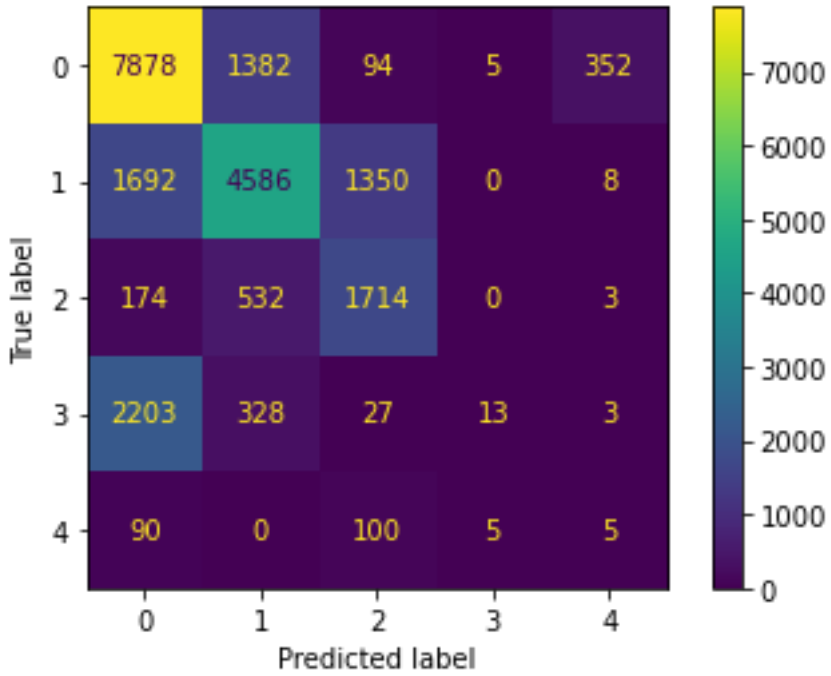

Fig 1.14: Confusion Matrix for Adaboost on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)

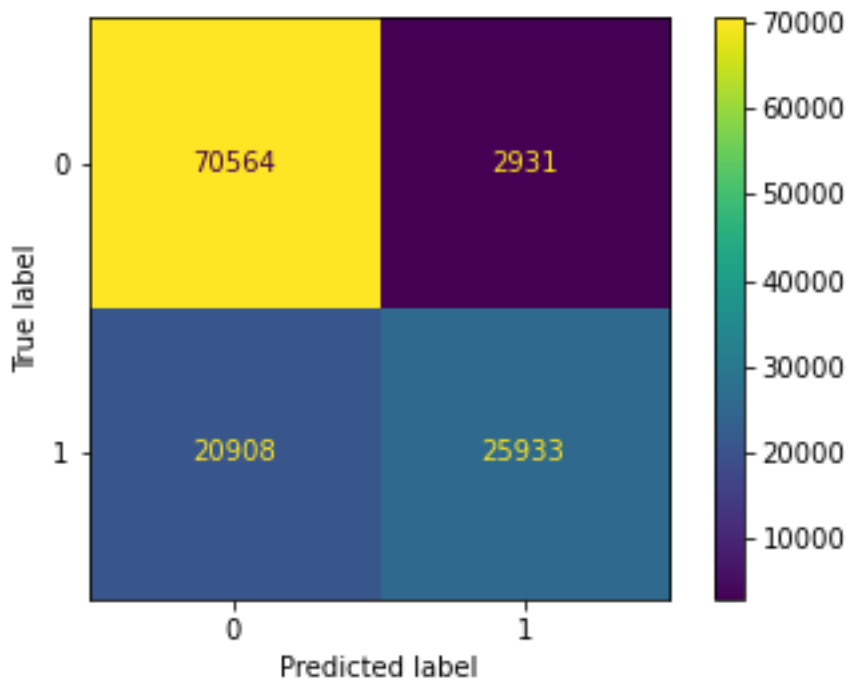

Fig 1.15: Confusion Matrix for Adaboost on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

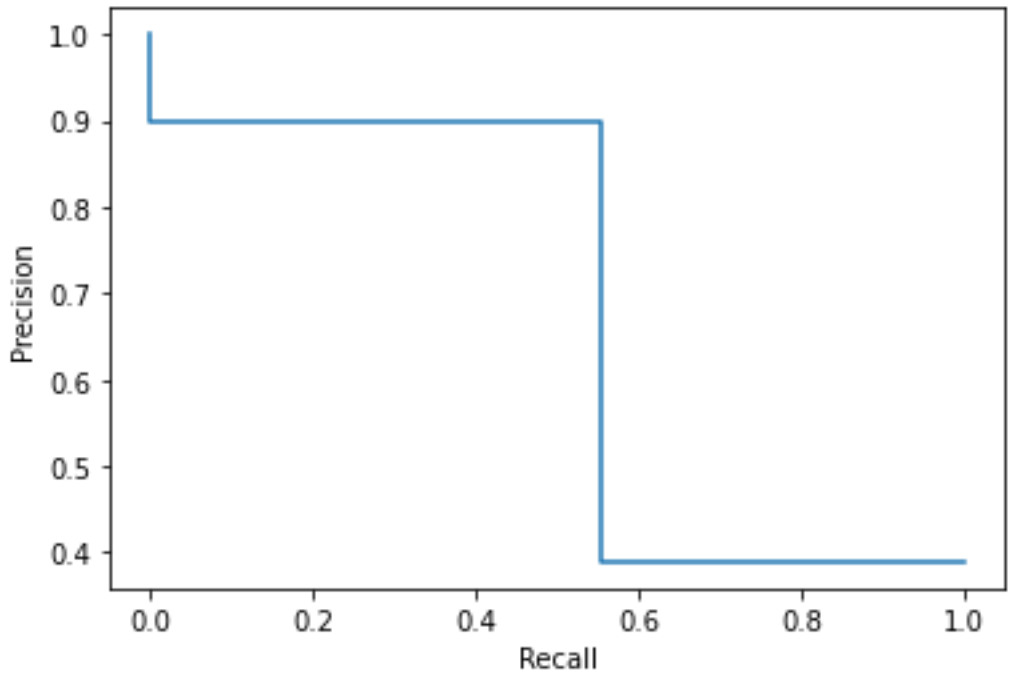

Fig 1.16: Precision-Recall Curve for Adaboost on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

MLP: Multi-Layer perceptron is a supervised learning algorithm which can be used for both classification and regression tasks. This model can be used to learn nonlinear boundary functions. The driving principle behind this and other Deep Learning model is that the weights are adjusted according to the defined loss function during the learning process and non-linearity is introduced via activation

function. This means that the model can approximate any nonlinear function as long as the input data is a complete representation of the boundary function. This makes MLP a powerful tool as it abstracts the feature extraction step and can be generalised to solve an array of problems with the help of raw data.

Algo:

1. Forward Pass: During this we pass the input datapoint through the model with randomly or previous state of the weights.
2. Loss calculation: In this step the predicted values are checked against the ground truth labels and the distance is calculated to check how far from the ground truth the predictions are.
3. Backward Pass: The calculated loss is propagated back into the layers; this is where the weights are updated.
4. **Key Params:**
    a. Hidden Layers: Controls the depth of MLP by controlling the number of layers.
    b. Activation Function: Introduces nonlinearity in the model.
    c. Solver: Determines which optimization technique will be used.
    d. Learning Rate: Determines the speed of convergence or the weight updating process.

Following tables show the evaluation metrics values on both the datasets for KNN.

Attack Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign	67%	93%	78%	9711
DoS	88%	74%	80%	7636
Probe	81%	73%	77%	2423
R2L	87%	14%	24%	2574
U2R	0%	0%	0%	200

Table 1.12: *Evaluation metrics for each class obtained from MLP on KDD dataset.*

Attack Class	**Precision**	**Recall**	**F1-Score**	**Support**
Normal	71%	95%	82%	73495
Attack	85%	40%	54%	46841

Table 1.13: *Evaluation metrics for TON IoT Telemetry Dataset obtained from MLP.*

Following figures show the confusion matrices on both the datasets for MLP.

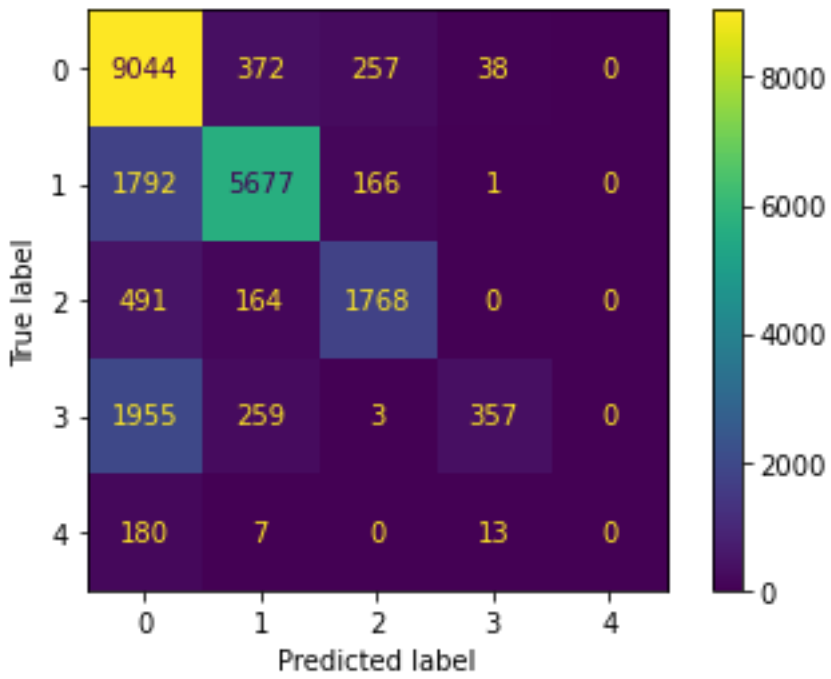

Fig 1.17: Confusion Matrix for MLP on KDD dataset. (0 – Benign, 1 – DoS, 2 – Probe, 3 – R2L, 4 – U2R)

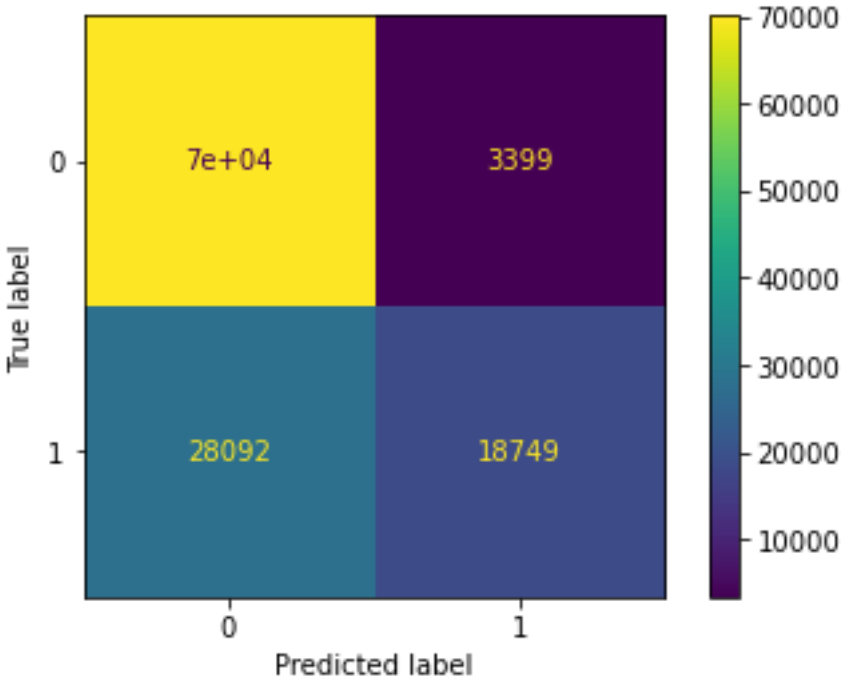

Fig 1.18: Confusion Matrix for MLP on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

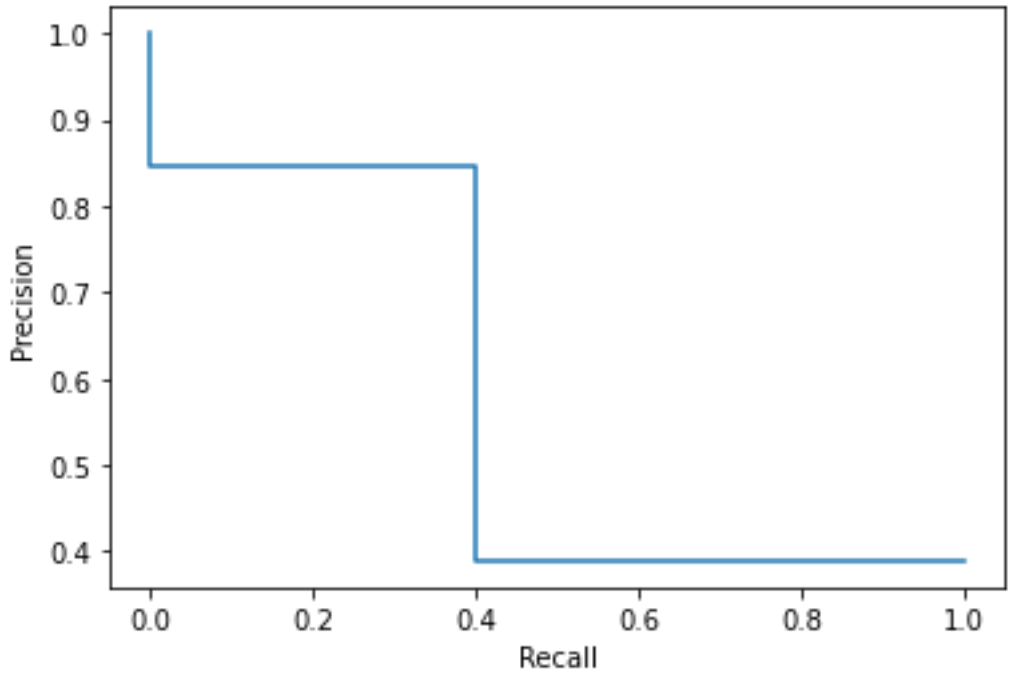

Fig 1.19: Precision-Recall Curve for MLP on TON IoT Telemetry dataset. (0 – Normal, 1 – Attack)

3 – Results

For KDD Dataset, Decision Tree scores highest in accuracy, recall and FPR and second most efficient in prediction time. KNN scores highest in accuracy along with Decision Trees and scores highest in precision, while Logistic Regression is the most efficient in prediction time.

For TON_IoT Telemetry Dataset, Random Forest scores highest in accuracy, precision, recall and F1-score. LR has the least FPR and MLP has the most efficient prediction time. The prediction time of MLP stays similar when the dataset is changed to a larger dataset.

Results on KDD Dataset

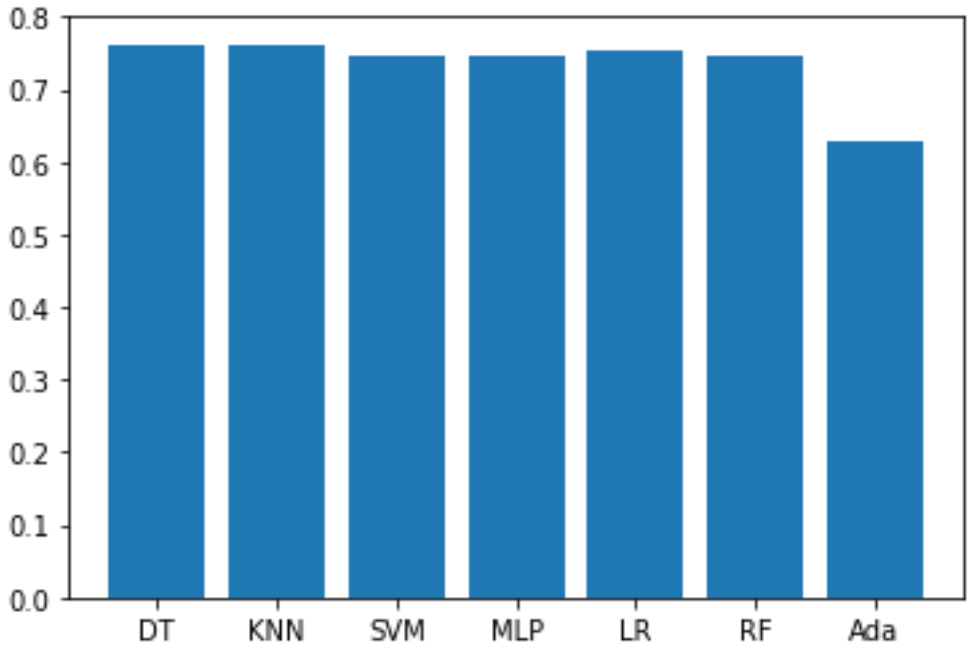

Fig 2.1: Accuracy comparison between the classifiers used on KDD Dataset.

TABLE 6. Result of each algorithm on KDDTest+.

Algorithms	Accuracy	Precision	Recall	F1	Time(S)
DeciTree	79.71%	83.51%	79.72%	77.31%	6.34
RanForest	76.64%	81.85%	76.64%	72.17%	1.86
kNN	75.51%	80.97%	75.51%	71.41%	86.49
LR	73.58%	74.65%	73.58%	69.13%	43.77
SVM	74.09%	80.91%	74.09%	70.38%	1785.2
DNN	81.6%	84%	81.6%	80.18%	227.8
Adaboost	76.02%	81.82%	76.02	72.12%	265.1

Table 2.1.1: *Performance as reported by [1]*

Algorithm	Accuracy	Precision	Recall	F1-Score	FPR	Train Time (s)	Test Time (s)
DeciTree	76%	81.48%	76.21%	73.05%	5.9%	2.6	0.02
RanForest	75%	81.1%	74.5%	70.7%	6.3%	12.43	0.2
kNN	76%	81.69%	76.19%	72.53%	5.9%	0.21	66.92
LR	75%	80%	75.3%	71.5%	6.1%	9.8	0.01
SVM	75%	81.5%	74.6%	71.3%	6.3%	87.13	21.3
MLP	75%	77.2%	74.7%	71.8%	6.3%	27.74	0.03
AdaBoost	63%	63.01%	62.9%	59.27%	9.2%	31.8	0.98

Table 2.1.2: *Performance after applying the list of classifiers on KDD Dataset. The best value for every metric is highlighted.*

Results on TON_IoT Telemetry Dataset

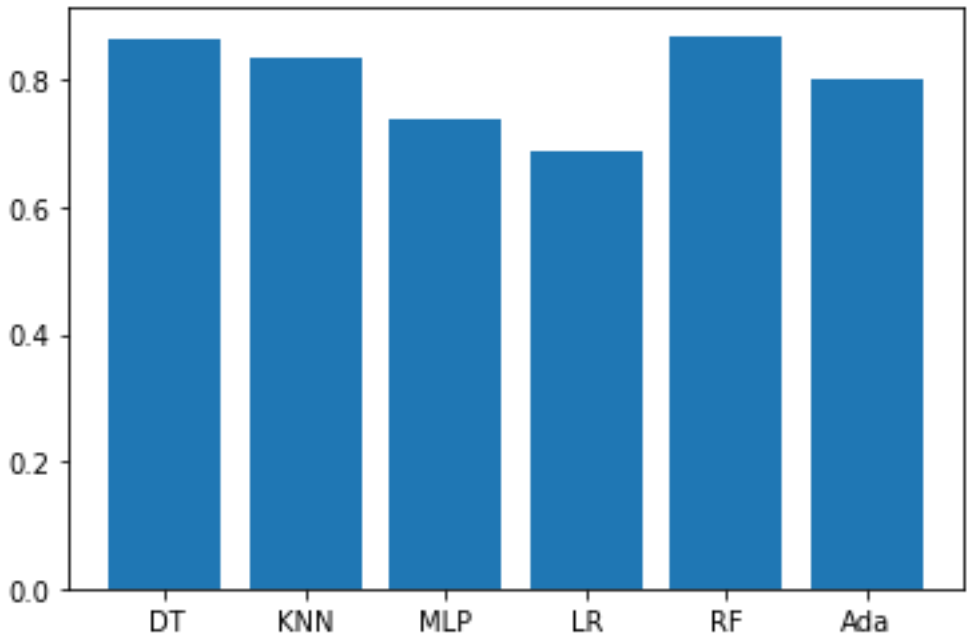

Fig 2.2: Accuracy comparison between the classifiers used on TON_IoT Telemetry Dataset.

Models	Accuracy	Precision	Recall	F-score	Train Time	Test Time
LR	0.61	0.38	0.62	0.47	30.778	0.014
LDA	0.62	0.46	0.63	0.51	1.947	0.017
*k*NN	0.72	0.71	0.73	0.70	18.262	79.02
RF	0.71	0.69	0.72	0.67	8.233	0.17
CART	**0.77**	**0.77**	**0.77**	**0.75**	5.048	0.023
NB	0.54	0.59	0.51	0.52	0.738	0.093
SVM	0.60	0.37	0.61	0.46	10526.148	965.971
LSTM	0.68	0.64	0.68	0.63	1375.305	5.928

Table 2.2.1: *[Performance as reported by [2]]*

Algorithm	Accuracy	Precision	Recall	F1-Score	FPR	Train Time (s)	Test Time (s)
LR	69%	85.8%	23.8%	37.3%	2.4%	2.8	0.1
kNN	83%	86%	68.26%	76.11%	7%	0.59	688.4
RF	87%	90.6%	74.2%	81.6%	4.8%	43.3	2.58
DeciTree	86%	89.36%	74%	81%	5.6%	1.62	0.03
SVM*							
AdaBoost	80%	63.01%	62.9%	59.2%	9.2%	15.5	1.6
MLP	74%	84.6%	40%	54.3%	4.6%	19.12	0.025

Table 2.2.2: *Performance after applying the list of classifiers on TON_IoT Telemetry Dataset. The best value for every metric is highlighted.*

**SVM figures are not calculated due to computational restraints.*

There are few highlights from the results of this experiment, this experiment shows how tree-based models are very efficient in such problems of attack classification and intrusion detection. On the KDD dataset the best model was Decision Tree and on TON_IoT Telemetry Dataset the best dataset was Random Forest. If we observe the performance of these two models on the datasets where they are not clear winners, we can see that the performance is efficient. This suggests that tree-based models would be the best solution for this problem statement.

This experiment also brings out the flaw in sklearn SVM, where it could not be trained on TON_IoT Telemetry Dataset and takes 87 seconds for training in KDD dataset and 21.3 seconds for prediction. This shows the high computation power and efficiency overhead involved in training SVM pipelines. This makes the algorithm sub optimal for the given dataset and problem statement.

The strength of KNN is also highlighted as it performs quite well except for the prediction time overhead involved. The large prediction time shows the drawback of instance-based models as they take comparatively higher time to perform prediction, which is one of the crucial factors when deploying large scale machine learning systems on scale. Although this can be used for quick proof of concept pipelines as it has low training cost and can give an idea of potential achievable metrics.

Another key highlight was the low prediction time of MLP, although the low numbers across other metrics make this and AdaBoost a suboptimal approach. Logistic Regression performs well on KDD dataset but has a low recall of 23% for positive class, which makes it a sub optimal approach for this problem.

Another insight that the experiment provides is, due to low number of samples present in KDD dataset for U2R class, no algorithm can score high recall for U2R class. This highlights an important aspect of class imbalance as it can affect the model performance critically. Therefore, it is crucial to address class imbalance issues either artificially by resampling the data or revisiting the data collection process.

Hyper parameter tuning

Hyper parameter tuning is the process of selecting best set of parameters for which a model will yield best value for evaluation metric against which it is being optimized. Here, I have taken accuracy as the evaluation metric against which the parameters are tuned.

To demonstrate how hyper parameter tuning affects models' performance I choose the top two algorithms (Decision Tree and Random Forest) and perform Grid Search against accuracy to select the best set of parameters to generate evaluation metrics. It can be observed that accuracy remains

similar once optimized parameters are used for both the classifiers. The ranking of classifiers does not change when models with optimized parameters are used. This highlights how hyper parameter tuning can be used to boost performance across the desired metric by iteratively retraining and testing the model for the best set of parameters, however here the metrics are already at their near best.

Algorithm	Accuracy	Precision	Recall	F1-Score	FPR
DeciTree	76%	81.48%	76.21%	73.05%	5.9%
RanForest	75%	81.1%	74.5%	70.7%	6.3%

Table 2.3.1: *Performance on KDD Dataset after hyper parameter tuning.*
Best Params: Decision Tree: {Max_depth: 30}
Random Forest: { 'criterion': 'entropy', 'max_depth': 30, 'n_estimators': 160}

Algorithm	Accuracy	Precision	Recall	F1-Score	FPR
DeciTree	87%	93.08%	71.08%	80.6%	3.3%
RanForest	88%	97.7%	70.2%	82.1%	1.02%

Table 2.3.2: *Performance on TON_IoT Telemetry Dataset after hyper parameter tuning.*
Best Params: Decision Tree: {Max_depth: 60}, 1% gain in accuracy.
Random Forest: { 'criterion': 'gini', 'max_depth': 40, 'n_estimators': 260} }
Metrics in green show the gain after hyper param optimization.

Note that the optimized parameters would change if they are optimized against other scoring metrics like precision or recall, which would lead in gain in the respective metrics but could yield in a drop of other metrics. Therefore, the algorithms must be tuned for the metric which has to be maximized.

In conclusion Random Forest seems the best model with Decision Trees a close second. Random Forest is chosen because of their ability to prevent overfitting as compared to Decision Trees and hence providing better generalisability. The third best model would be KNN for the comparable performance and low training overhead.

4 – References:

[1] X. Gao, C. Shan, C. Hu, Z. Niu and Z. Liu, "An Adaptive Ensemble Machine Learning Model for Intrusion Detection," in *IEEE Access*, vol. 7, pp. 82512-82521, 2019, doi: 10.1109/ACCESS.2019.2923640.

[2] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood and A. Anwar, "TON_IoT Telemetry Dataset: A New Generation Dataset of IoT and IIoT for Data-Driven Intrusion Detection Systems," in *IEEE Access*, vol. 8, pp. 165130-165150, 2020, doi: 10.1109/ACCESS.2020.3022862.

[3] J. Han, J. Pei and M. Kamber, Data Mining: Concepts Techninqs, Amsterdam, The Netherlands: Elsevier, 2011

[4] X. Wu, "Top 10 algorithms in data mining", *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1-37, 2008.

[5] S. Mukherjee and N. Sharma, "Intrusion detection using naive Bayes classifier with feature reduction", *Procedia Technol.*, vol. 4, pp. 119-128, Oct. 2012.

[6] M. S. Mok, S. Y. Sohn and Y. H. Ju, "Random effects logistic regression model for anomaly detection", *Expert Syst. Appl.*, vol. 37, no. 10, pp. 7162-7166, Oct. 2010.