Intrusion Detection using Machine Learning

Isha Chaudhary
Dept of Electrical Engineering
Indian Institute of Technology Delhi
Delhi, India
Isha.Chaudhary.ee318@ee.iitd.ac.in

***Abstract*—This study is aimed at demonstrating the applicability and efficiency of appropriately trained Machine Learning Models in predicting attacks on computer networks. With the rapidly growing population of Cyber-Physical Systems and increasing number of online transactions, real-time, continuous and accurate prediction of attack on a component of the network becomes extremely important. This use-case is suitably addressed by the common Machine Learning models available to Software Developers and continuous research has shown the effectiveness of such ML models applied to this problem. This study is one such attempt to identify the best performing ML models in the development of Intelligent secure cyberspaces. The ML models tested in this study were trained on the real network observations from the 2019 IEEE BigData Cup Challenge.The models tested under this study were KNN, Naive Bayes, Support Vector Classifier, Random Forests Classifier, and Multilayer Perceptrons. Appropriate data pre-processing was performed to extract the best possible performance from the ML models.**

***Index Terms*—Cybersecurity, Machine Learning, data analysis, Intrusion Detection Systems**

I. Introduction

Intrusions and attacks on computers are continuously evolving and protection against them is very important to prevent loss of valuable resources. Hence it becomes imperative to have reliable and real-time Intrusion Detection Systems. This gains even more importance in these times of virtual lifestyles of a majority of people. With numerous transactions and confidential interactions happening over the web, an improvised Intrusion Detection System is needed to make the web more secure. To give enhanced robustness, firewalls are often combined with Intrusion Detection Systems (IDSs). Traditionally, IDSs have been based on predefined activities, described by experts. But with the highly dynamic atmosphere, ML tools can be to classify activities as routine or attacks. To enhance the robustness of such data-driven IDSs, ML models along with inputs of human expert are used.

In the 2019 IEEE BigData Cup Challenge, the participants had to use network traffic data, which was labeled by Security Operations Center team members of a company, to detect attacks in a unlabelled test data. The performance of a model was evaluated on the basis of Receiver Operating Characteristics AUC (Area Under Curve) scores achieved by the different models. While handling such a dataset, feature engineering plays a crucial role. It requires some amount of expertise to figure out the more important features for the prediction task. This requirement can be leveraged to some extent by using feature extraction using Principle Component Analysis. This study aimed at analyzing several ML models on the datasets provided in the 2019 IEEE challenge. AUC was used as a metric to evaluate the performance of the ML models used.

The first part of this study utilized the cybersecurity_train dataset provided in the challenge. The performance of the different models was obtained with it. Then the localized_alerts dataset was merged with the cybersecurity_train dataset and used for the evaluation.

The major challenge which arises out of such a fusion of ML into IDSs is the enormous amount of data that is continuously generated by computers and networks. It remains a challenge to scale ML models to huge datasets and extensive feature engineering is required. This results in such ML empowered IDSs not being a part of common Personal Computers and hand-held devices. The enormous computation power demanded by such systems often may not turn out to be ergonomic.

The remainder of this paper is structured as follows. Section II briefly presents the literature review done to perform the study. Section III expands on the Methodology adopted, which includes an elaboration on the data pre-processing steps and model selection. Section IV demonstrates the experiments, results and conclusions derived from them. Section V concludes the paper and provides suggestions for future work.

II. Literature Review

An intrusion is often an attempt to disrupt the normal functioning of a system and cause data and monetary loss. Intrusion detection can be broadly classified as Misuse detection and Anomaly Detection. Misuse detection keeps track of the system vulnerabilities to detect intrusions. Anomaly detection looks out for unusual signals and patterns in the data received from the system being secured.

The approach adopted in this study is to develop data-driven IDSs, which can scale to meet the requirements under a highly dynamic intrusion atmosphere.

Various attempts have been made at utilizing various models to get higher AUC scores with the dataset used in this study. The study conducted in [1] primarily uses a tree-based approach of classifying attacks. They extend their study to test the performance of Neural Networks on this dataset.

III. Methodology

This study initially utilized the *cybersecurity_train* dataset to evaluate several ML models.

A. Description of the dataset and data pre-processing

Dataset Description

- The dataset contains 39427 samples with 62 features.
- Being a labeled training dataset, one of the columns called 'notified' contains the labels for the given sample.
- A label '0' signified the absence of an attack, whereas '1' signified an attack.
- Approximately 5.77% of the dataset is composed of samples of attacks. The remaining samples are of non-attacks.
- Thus this dataset shows a heavy bias towards the non-attack samples, which is expected in an actual situation, wherein attacks are not occurring commonly.
- Thus, taking the bias into account, the AUC metric is chosen for the ML model evaluation, as was also done by the challenge organizers on the test dataset.
- The column containing alert_ids is like the primary key of the dataset.
- The merged dataset containing entries from the localized_alerts dataset combined with the cybersecurity_train dataset is a huge dataset with 6567968 samples.
- The number of features in the merged dataset is 27.
- The experiments will not be carried out on the merged dataset, owing to its huge size.
- The results demonstrated below are only on the basis of pre-processed cybersecurity_train dataset.

Data Pre-processing procedure

- The cybersecurity_train dataset contains some empty entries. So the empty entries were detected and filled with 0 values.
- The dataset containes features of three types: int64, float64, and object.
- The object type features include datatypes such as string too.
- To make the data fit to be training an ML model, the object type features were converted to numeric type features.
- After the conversion, 'alert_ids' and 'client_code' features were dropped from the dataset. This is because, they couldn't be converted to numeric values and they also did not convey any additional information on the classification of events.
- The labels (column with title 'notified') were next separated from the features.
- The training and testing datasets were carved out of the original labeled dataset provided. The 7:3 train:test split was used.
- The features were standardized in the training and testing datasets created.

B. Deploying ML: Considerations and Hyperparameters

The various models tested in this study are detailed next. The models used were K-Nearest Neighbors, Naive Bayes, Random Forests Classifier (ensemble learning version of Decision Trees), Support Vector Machine Classifiers, Neural Networks (Multi-Layer Perceptrons).

1) K-Nearest Neighbors:

- The K-Nearest Neighbors Algorithm is one of the simplest classification algorithms to implement.
- Theoretically, 1-Nearest Neighbor classification shows nearly half the optimality of a Bayesian classifier, along with its ease to implement and use.
- The basic idea of the algorithm is to assign a sample the label carried by a majority of k of its nearest neighbor samples in the space of observations.
- The KNN algorithm used in the experimentation has been borrowed from Scikit Learn's implementation under *sklearn.neighbors*.
- To test the variation of the AUC values with the hyperparameter, K, a plot between the AUC values for different values of K was made.

2) Naive Bayes:

- The Naive Bayes Algorithm is a simplified version of Bayesian Classification, which is the most optimal classification algorithm known.
- The features are assumed to be conditionally independent of each other given the observation (sample).
- The Naive Bayes algorithm used in the experimentation has been borrowed from Scikit Learn's implementation under *sklearn.naive_bayes*.

3) Random Forests Classifier:

- This classification technique is an ensemble classifier, based on Decision Trees.
- The number of estimators is a hyperparameter, over which we can optimize and get the best results.
- A plot between the AUC and the number of estimators was made to show the best value for the hyperparameter.
- The Random Forests Classifier algorithm used in the experimentation has been borrowed from Scikit Learn's implementation under *sklearn.emsemble* called *RandomForestClassifier*.

4) Support Vector Machine Classification:

- The Support Vector Machine Classifier tries to add the maximum margin between the line separating the samples with different labels, by appropriately adjusting the weight parameters of the line. The line is in the space of the features and is actually a linear combination of the features in the most simple design of the algorithm.
- Kernel functions can be used to model non-linear relations between features and labels.
- The Support Vector Machine Classifier algorithm used in the experimentation has been borrowed from Scikit Learn's implementation under *sklearn.svm* called *SVC*.

5) Multi-Layer Perceptron (MLP):

- A Perceptron is a computational machine to mimic a neuron and assign binary labels (actually gives a probability, which can be used for the classification) to the samples received by it, after applying a specified activation function to the weighted sum of features of the sample.
- The weights form the parameters of the model. The activation function is a hyperparameter.

- A Multi-Layer Perceptron has one/more hidden layers, which may create very basic representations of the sample received, by using various functions of the features, to enable the machine to classify the sample.
- The Multi-Layer Perceptron algorithm used in the experimentation has been borrowed from Scikit Learn's implementation under *sklearn.neural_network* called *MLP-Classifier*.
- Some hyperparameters of the MLP algorithm were also cross-validated, using Grid Search CV. Grid Search CV is an algorithm to take all the combinations of the hyperparameter values passed to it, run the algorithm specified for the chosen hyperparameters and return the best performing set of hyperparameters for the given algorithm. An implementation of this algorithm from Scikit Learn Library was used, under *sklearn.model_selection* called *GridSearchCV*. 3 Cross Validations were performed.

C. *Evaluating the Models*

- The main parameter used for the evaluation of all the models tested in this study was the ROC-AUC, which the Receiver Operating Characteristics Area Under the Curve.
- The ROC curve plots the True Positive Rate against the False Positive Rate, in the implementation in Scikit Learn. In other implementation, ROC curves can be drawn between any combinations of TP, FP, TN, FN.
- A higher AUC value indicates higher success in the classification task.
- The calculation of AUC for the different algorithms is done using functionality in *sklearn.metrics.roc_auc_score*.

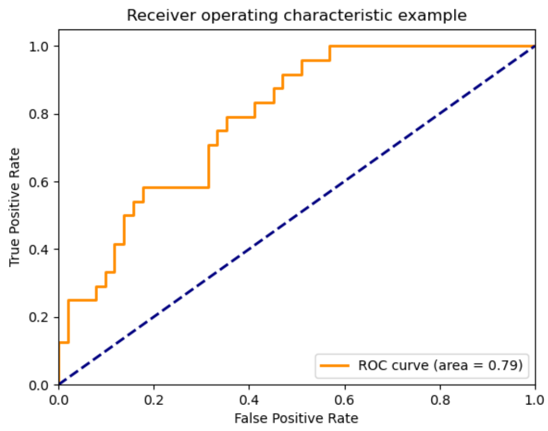

Fig. 1. An example of an ROC

IV. Experiments, Results and Inferences

A. *Experimental Design*

The experiments were carried in a Google Colab notebook. The most basic, free version of Google Colab was used, with limited RAM. This posed a constraint in applying the models to bigger datasets.

B. *Cross Validation for Hyperparameter Tuning*

1) MLP: A Grid Search CV procedure was applied to determine the best AUC for a set of possible hyperparameters. The hyperparameters which were tuned for the best AUC were:

1) Hidden layer configuration
2) The activation function used (ReLu vs Logistic)
3) alpha value

C. *Performance Metrics for the Various models used*

The variation of the AUC metric with the hyperparameters is shown for the following models:

1) KNN: For KNN, the hyperparameter k was varied. The AUC values for different values of k were obtained and plotted as shown in Figure 2. The best AUC score (0.806) is obtained

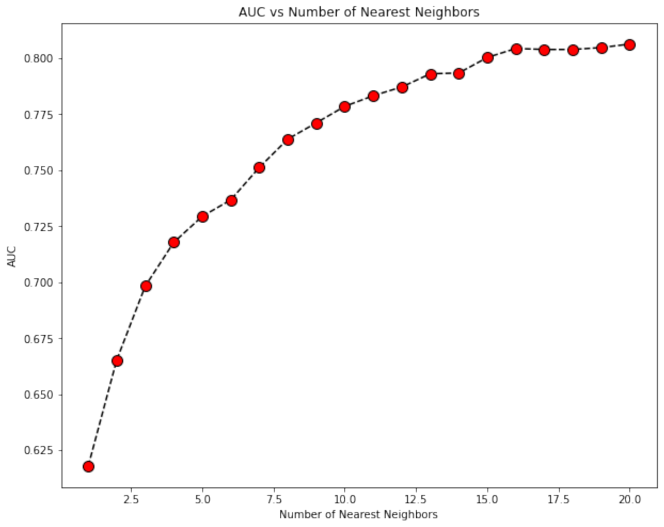

Fig. 2. AUC vs k for KNN Algorithm

with 20 neighbors.

2) Random Forests Classifier: For Random Forests, the hyperparameter, number of estimators was varied to show the variation of AUC. A plot demonstrating how AUC varies with the number of estimators is shown in Figure 3.

The best AUC score (0.918) is obtained with 950 estimators.

3) Tabulating the Performance: The AUC, Precision, Recall, F1 Score metrics were used to evaluate the performance of the various ML models used. The performance of best models after cross-validation is shown in Table I.

The Figure 4 clearly indicates the superiority of Random Forest Classifier over other classifiers used for this particular dataset.

V. Conclusion and Further Scope

The various experiments conducted with the selected ML models indicate a superiority of Random Forests Classifier over the other models. Several other studies listed in the references have adopted a ensemble tree-based approach for

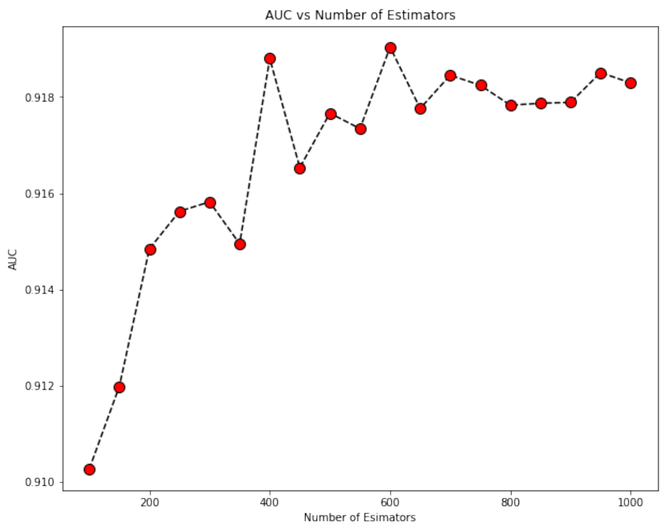

Fig. 3. AUC vs number of estimators for Random Forests Algorithm

TABLE I
PERFORMANCE OF ML MODELS

ML Model	Performance			
	AUC	***Precision***	***Recall***	***F1 Score***
KNN	0.806	0.942	0.999	0.969
SVM	0.778	0.941	0.999	0.969
RFC	0.918	0.947	0.998	0.972
NB	0.775	0.991	0.430	0.6
NN	0.845	0.95	0.989	0.969

[a] The Precision, Recall, F1 Score values are for class 0 as positive.

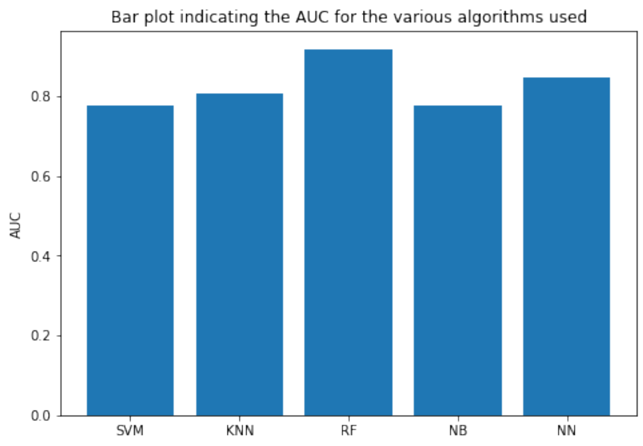

Fig. 4. An AUC based comparison of the ML models used

this problem.
Neural Networks too exhibit good performance when the hyperparameters are properly tuned using extensive Cross-Validation over possible combinations.

A. *Further Scope*

- Similar studies can be performed on more powerful machines to get better AUC values and to perform more cross-validation to better tune the hyperparameters. A variety of other ML algorithms can be tested in this way and the best performing algorithm can be deployed on real-time IDSs. More log data can also be processed by increasing the capabilities of the machine.
- Deep Learning Architectures can be experimented with and the results can be compared against other ML models.
- ML models can be customized to be less power demanding and be small enough to be incorporated in mobile IDSs.
- The ML models can be customized to operate on a stream of inputs, to be functional in real time systems. The models can then constantly update themselves with the evolving attack patterns.

VI. ACKNOWLEDGMENT

This study was performed for the term project of the course SIV810, Special Module in CyberSecurity course at IIT Delhi. The author is highly grateful to the course Instructor Prof. Anupam Joshi, for his teaching, constant guidance and support throughout the project.

REFERENCES

[1] A. F. M. Huang, Y. Chi-Wei, H. Tai, Y. Chuan, J. J. C. Huang and Y. Liao, ”Suspicious Network Event Recognition Using Modified Stacking Ensemble Machine Learning,” 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 5873-5880, doi: 10.1109/BigData47090.2019.9006391.
[2] D. Sartiano, G. Attardi, L. Deri and M. Martinelli, ”Suspicious Network Event Recognition Leveraging on Machine Learning,” 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 5915-5920, doi: 10.1109/BigData47090.2019.9006178.