Leveraging Machine Learning for Network Intrusion Detection

Ayoub El Ourrak

NOVA School of Science and Technology (FCT NOVA)

Introduction

The goal of this project is to develop a network intrusion detection system that can accurately differentiate between malicious connections, referred to as intrusions or attacks, and legitimate normal connections. An Intrusion Detection System (IDS) is a software application that employs machine learning algorithms to detect network intrusions. By monitoring network activity, an IDS safeguards computer networks from unauthorised access, including potential insider threats.

The main objective of the intrusion detection learning task is to construct a predictive model, typically a classifier, that effectively discriminates between ”bad connections” (intrusions/attacks) and ”good (normal) connections.”

Effectively detecting different attack types requires a combination of diverse techniques rather than relying on a single approach. The IDS must possess a range of skill sets to effectively handle various types of attacks. Many researchers have conducted direct comparisons, evaluating different machine learning techniques and combinations of techniques, to determine the most effective approaches. Some of the challenges highlighted include the effectiveness of detecting attacks without generating excessive false alarms, the adaptability of IDS to changing environments, speed in dealing with high-performance networks, diversity of network environments, fault tolerance, inter-operability, ease of use, and timeliness in handling large amounts of data.

Objective

The goal of the project is to develop a ML model able to classify the type of a network connection by differentiate between benign and attack connection and also specifying which type of attack. The main challenge will be to train efficiently the system despite the imbalanced dataset exploring different techniques such as under-sampling or using penalized models.

Dataset

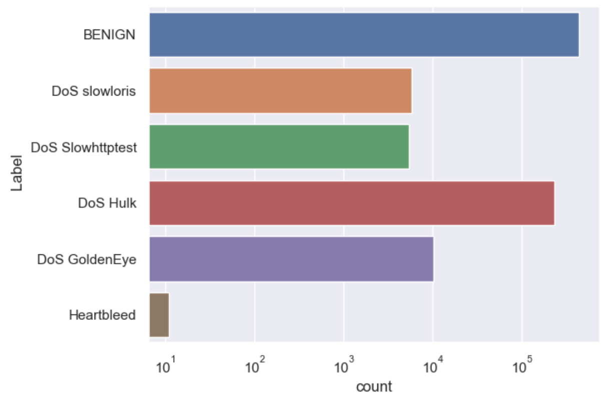
The dataset that will be used is part of CICIDS2017 dataset that contains benign and the most up-to-date common attacks, which resembles the true real-world data. It also includes the results of the network traffic analysis using CICFlowMeter with labelled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack.

The data capturing period is Wednesday July 5 2017, recording both good connections and malicious ones, namely several DoS and Heartbleed attacks. It is composed of 692703 elements and each one of them is characterised by 84 features extracted from the generated network traffic and a label indicating the type of the connection.

As shown in the figure 1 the dataset is highly imbalanced with 63.5% of the traffic being ”benign” and even after considering only the malicious connections the ”DoS Hulk” constitutes 91.5% of the attacks.

Preprocessing

To provide a more suitable data for the neural network classifier and the other techniques, the dataset is passed through a group of preprocessing operations. First of all we removed features useless for training the ML model such as ”Flow ID”, ”Source IP”, ”Source Port”, ”Destination IP”, ”Destination Port”, ”Protocol”, ”Timestamp” that are only used to identify the connections. It is important to remove such information to provide unbiased detection, where using such information may results in overfitted training toward socket information. However, it is more important to let the classifier learn from the characteristics of the packet itself, so



	count
BENIGN	440031
DoS Hulk	231073
DoS GoldenEye	10293
DoS slowloris	5796
DoS Slowhttptest	5499
Heartbleed	11

Figure 1: Network connection classes

that, any host with similar packet information is filtered out regardless to its socket information. Moreover Exploratory Data Analysis (EDA) has been performed first on the whole dataset and then on the subset composed only by the ”bad connections” which resulted in 10 features (”Bwd PSH Flags”, ”Fwd URG Flags”, ”Bwd URG Flags”,”CWE Flag Count”,”Fwd Avg Bytes/Bulk”, ”Fwd Avg Packets/Bulk”, ”Fwd Avg Bulk Rate”, ”Bwd Avg Bytes/Bulk”, ”Bwd Avg Packets/Bulk”, ”Bwd Avg Bulk Rate”) being constant on both datasets. After removing those features from the dataset we replaced missing and infinity values, converted string values representing numbers to numeric and encoded the multi-class labels so that the classifier can learn the class number that each tuple belongs to.

Feature selection

After the preprocessing phase, we proceeded with the selection of the best features. To reduce the dimensions of the dataset we removed the highly correlated features. That is because if two features are highly correlated then the information they contain is very similar, and it is likely redundant to include both the features. So it is better to remove one of them from the feature set. Following we used **SelectKBest** to select the best 20 features and the result was this list of features: (”Flow IAT Max”, ”Fwd IAT Std”, ”Bwd Packet Length Mean”, ”Flow IAT Std”, ”Bwd Packet Length Std”, ”Bwd Packet Length Max”, ”Packet Length Std”, ”Max Packet Length”, ”Packet Length Mean”, ”Packet Length Variance”, ”Flow Duration”, ”Fwd IAT Mean”, ”Flow IAT Mean”, ”ACK Flag Count”, ”FIN Flag Count”, ”Bwd IAT Std”, ”Bwd IAT Max”, ”Bwd IAT Mean”, ”min_seg_size_forward”, ”Flow Packets/s”)

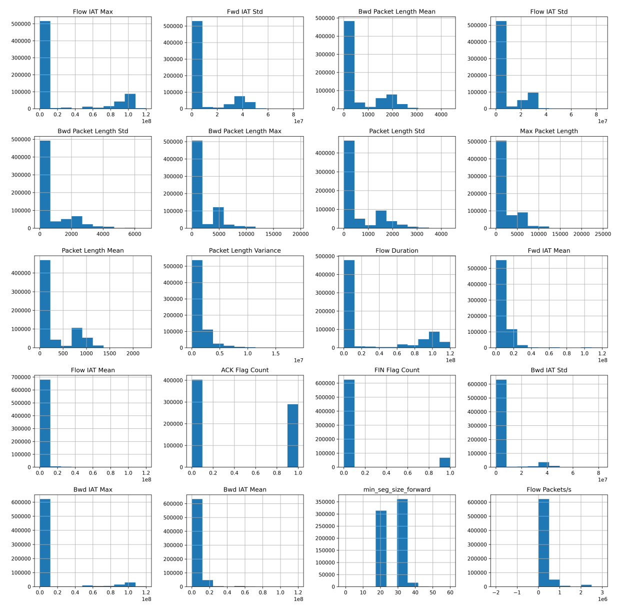


Figure 2: features distribution

Modeling

The next step is to find the best model that can fit the dataset and provide optimum results, to do so we had to experiment with different types of models and techniques.

Models

Several models were used to fit the dataset, some of them are usually used for binary classification but given that the problem is a multi-class classification we used three techniques to fit the binary classifiers to the dataset:

- **One-vs-One (OvO) multiclass strategy**, which consists in fitting one binary classifier for each pair of classes;
- **One-vs-the-rest (OvR) multiclass strategy**, which consists in fitting one binary classifier per class;
- **Output-code (OC) multiclass strategy**, which consist in representing each class with a binary code. At fitting time, one binary classifier per bit in the code book is fitted. At prediction time, the classifiers are used to project

new points in the class space and the class closest to the points is chosen;

The models used were:

- **Hoeffding Adaptive Tree classifier**, which uses ADWIN to monitor performance of branches on the tree and to replace them with new branches when their accuracy decreases if the new branches are more accurate;
- **Adaptive Random Forest classifier**, which induce diversity through re-sampling and randomly selecting subsets of features for node splits. It also has drift detectors per base tree, which cause selective resets in response to drifts;
- **Aggregated Mondrian Forest classifier**, which implementation is truly online, in the sense that a single pass is performed, and that predictions can be produced anytime;
- **Logistic regression** for binary classification;
- **Stochastic Gradient Tree** for binary classification.

The metric we used to measure the performance of the models is the **F1 score**, defined as the harmonic mean of precision and recall:

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This metric works well with imbalanced datasets and as we can see in table 1 the Adaptive Random Forest classifier is the model with the best F1 score.

Models	Weighted avarage score
HA Tree	99.10%
AR Forest	99.73%
AM Forest	99.66%
Logistic Regression (OvO)	99.03%
Logistic Regression (OvR)	99.01%
Logistic Regression (OC)	98.90%
SG Tree (OvO)	45.25%
SG Tree (OvR)	73.92%
SG Tree (OC)	68.26%

Table 1: F1 Score of the models

Imbalance

In this step we used different techniques to solve the problem of the imbalance of the dataset, the techniques are the following:

- **Random over-sampling**, which will train the provided classifier by over-sampling the stream of given observations so that the class distribution seen by the classifier follows a given desired distribution;
- **Random under-sampling**, which will train the provided classifier by under-sampling the stream;
- **Random sampling by mixing under-sampling and over-sampling**, which will train the provided classifier by both under-sampling and over-sampling the stream;
- **Cost sensitive learning**, which will weight differently residuals from minority and majority class.

We used the first three technique on the Adaptive Random Forest model meanwhile we used Cost sensitive learning on the Logistic Regression (OvR). As we can see on the table 2 the results show that the Adaptive Random Forest model has the best score even without any method that remove the imbalance.

Technique	F1 score	ROCAUC
AR Forest	99.56%	99.91%
Over-sampling	98.34%	99.89%
Under-sampling	88.29%	99.96%
Random sampling	93.94%	99.95%
Logistic Regression (OvR)	96.15%	99.43%
Log loss	96.99%	99.82%
Focal loss	96.66%	99.88%
Over-sampling and Focal loss	94.35%	99.95%

Table 2: Score of the techniques

Hyperparameter

The final step was to select the best hyperparameters of the best model, Adaptive Random Forest. The parameters that we took into consideration were:

- **n_models**, which is the number of trees in the ensemble;
- **max_features**, which is the maximum number of attributes for each node split;
- **metric**, which is the metric used to track trees performance within the ensemble;
- **split_criterion**, which is the function to measure the quality of a split;
- **leaf_prediction**, which is the prediction mechanism used at leafs;
- **max_depth**, which is the maximum depth a tree can reach.

Successive halving algorithm for classification was used to explore the several combinations of the parameters, this algorithm is made for performing model selection without having to train each model on all the dataset. At certain points in time (called ”rungs”), the worst performing will be discarded and the best ones will keep competing between each other. The results show that the best model is:

```
forest.ARFClassifier(
    n_models=15,
    max_features='sqrt',
    metric=metrics.F1(),
    split_criterion='gini',
    leaf_prediction='nba',
    max_depth=None)
```

Conclusion

In the table 3 we can see how the final model fit the dataset, it's interesting to see that the score is more or less proportional to the number of observations of the classes so a possible solution is to increment the observations of the minority classes. Nevertheless the ML model shows great results even if the class *Heartbleed* had so few observations.

Class	F1 score	Count
BENIGN	99.86%	440030
DoS GoldenEye	99.46%	10293
DoS Hulk	99.81%	231073
DoS Slowhttptest	99.53%	5499
DoS slowloris	97.84%	5796
Heartbleed	95.24%	11
Weighted average	99.82%	-

Table 3: F1 Score of the final model