Network Intrusion Detection Evaluation Using Machine Learning Models on Big Data

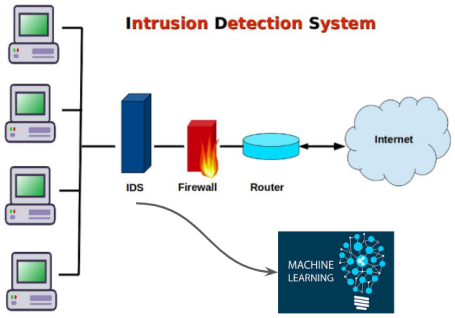
Savvas Kastanakis[1], Dimitris Karathanasis[1], Manos Chatzimpyrros[1]
[1]*Computer Science Department, University Of Crete, Greece*
{*csdp1096, csd3547, csd3137*}*@csd.uoc.gr*

Keywords: Intrusion Detection Systems, Big Data Applications, Machine Learning, Network Monitoring, Security

Abstract: Anomaly detection has been the main focus of many researchers due to its potential in detecting novel attacks. However, its adoption to real-world applications has been hampered due to system complexity as these systems require a substantial amount of testing, evaluation, and tuning prior to deployment. Running these systems over real labeled network traces with a comprehensive and extensive set of intrusions and abnormal behavior is the most idealistic methodology for testing and evaluation. To that end, we have designed and implemented a Network Intrusion Detection Tool using Machine Learning Models over real-world traces. We have parsed 50Gb of raw network traffic (PCAP files) and we tuned a plethora of models, to find the one instance that learns most out of the provided data. Our analysis has shown that Random Forest Classifier with 100 Decision Trees performs the best in our case study, achieving an 84% Cross Validation score on the Validation and Test Set respectively.

1 INTRODUCTION

In the world scenario, concerns with security and privacy regarding computer networks are always increasing. Computer security has become a necessity due to the proliferation of information technologies in everyday life. The increase in the number of Internet accesses and the emergence of new technologies, such as the Internet of Things, are accompanied by new and modern attempts to invade computer systems and networks. Companies are increasingly investing in studies to optimize the detection of these attacks. Institutions are selecting intelligent techniques to test and verify by comparing the best rates of accuracy. This research, therefore, focuses on rigorous state-of-the-art literature on Machine Learning Techniques and Intrusion Detection for computer network security. The work aims to deal with several intelligent techniques and their applied intrusion detection architectures in computer networks and machine learning.



Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) are the most important defense tools against the sophisticated and ever-growing network attacks. We have implemented a Machine Learning Intrusion Detection System, able to classify malicious/benign traffic, with an 84% Cross Validation Score.

2 AWS ENVIRONMENT SETUP

Amazon Web Services (AWS) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality. In detail, we have used the EC2, S3 Services of AWS.
Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity. Amazon Simple Storage Service (S3) is storage for the Internet. It is designed to make web-scale computing easier for developers.
Amazon S3 has a simple web services interface that

you can use to store and retrieve any amount of data, at any time, from anywhere on the web. Our setup for this project included:

- 8 64bit CPUs running Linux
- 32Gb of RAM
- Up to 5Gb Network Bandwith
- 100Gb Hard Disk Storage

Type	vCPUs	Memory (GiB)	Network Performance
t3a.2xlarge	8	32	Up to 5 Gigabit

3 DATA PREPROCESSING

For this project, we have use the CICIDS2017 dataset, which can be found in the following link: *https://www.unb.ca/cic/datasets/ids-2017.html* **CICIDS2017** dataset contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the corresponding labeled flows of each one of the packets included in the PCAPs.
The data capturing period started at 9 a.m., Monday, July 3, 2017 and ended at 5 p.m. on Friday July 7, 2017, for a total of 5 days.
Monday is the normal day and only includes the benign traffic.

3.1 Raw Network Packets

We have parsed the raw network packets, using the python library Scapy. For each packet parsed, we have monitored 6 Network Protocols:

- IPv4
- IPv6
- TCP
- UDP
- ICMP
- ARP

For each of the above protocols, we monitored the existent values, and assigned them to the respective variables. In the case where there weren't any values present (because of absence of the protocol field in the packet) a default value was assigned.

3.2 Labeled Network Traffic

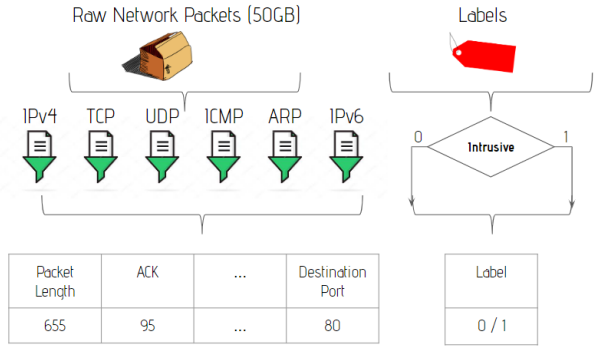
Each label corresponded to one packet from the PCAP files. Labels were the following:

- Benign
- Brute Force FTP
- Brute Force SSH
- DoS, DDoS
- Heartbleed
- Web Attack
- Botnet

We applied a **Transformation to Binary**, such that we have a binary classification problem to cope with. To that end, we assigned a label 0 for each Benign packet and a label 1 for any of the malicious categories.

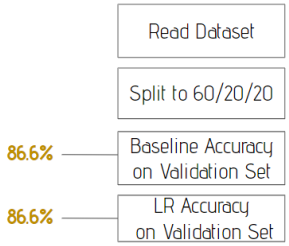
3.3 One Data Point

The following figure shows how we collected a single data point from raw network traffic and labels. After the completion of this procedure, we ended up with a dataset of 1.287.877 entries/datapoints.



4 THE FIRST ML PIPELINE

Proceeding with the implementation, we began constructing our first ML models.
We used a series of algorithms chained, composed, and scrambled together in some ways to process our data.
We need to choose an appropriate model from several rivaling approaches(Baseline-Linear Regression), so we split the database in three sets using 60% for the Training set, 20% for the Testing set and 20% for the Validation set and we continue with three main phases.



- Training phase,implements pairing the input with the expected output
- Validation phase makes our models observable and result in selecting the best performing approach using the validation data
- Test phase estimates the performance of the selected approach

We scored the same accuracy (86.6%) both for the baseline and LR model on the validation set. This behaviour though, is abnormal, since Logistic Regression should outperform a naive random label picking approach.

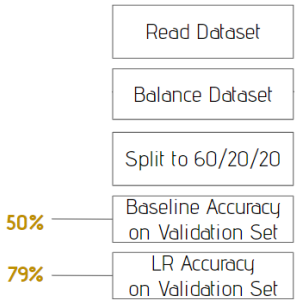
5 BALANCING THE DATASET

How is it possible to achieve the same accuracy for both Baseline and LR models?
This is happening because many ML algorithms are designed to maximize overall accuracy. We observe in the previous figure, that LR model ignores the minority class in favor of the majority class. So, the previous LR model has 87% because it's predicting only one class, the majority. For the reasons mentioned above, we should take into consideration two major problems:

1. Balancing the Dataset
2. Picking an Appropriate Performance Metric

In order to deal with the imbalanced dataset, we Up-sampled the Minority class. Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce it's signal. There are several heuristics for doing so, but the most common way is to simply resample with replacement. We trained another LR model, this time on the balanced dataset. We're getting 79% accuracy on LR and 50% on the Baseline model. Now the model is no longer predicting just one class.

While the accuracy took a nosedive, it's now more representative as a performance metric.

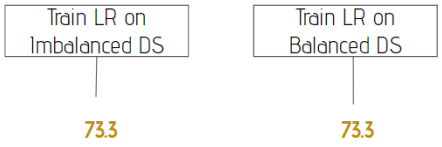


6 PICKING AN APPROPRIATE PERFORMANCE METRIC

Imbalanced classes put accuracy out of business. This is a surprisingly common problem in machine learning (specifically in classification), occurring in datasets with a disproportionate ratio of observations in each class.

Standard accuracy no longer reliably measures performance, which makes model training much trickier. As we discussed earlier, many machine learning algorithms are designed to maximize overall accuracy. So far, we've looked at one way of addressing imbalanced classes by resampling the dataset.

Next, we'll look at using other performance metrics for evaluating the models. For a general-purpose metric for classification, we recommend Area Under ROC Curve (AUROC). Intuitively, AUROC represents the likelihood of our model distinguishing observations from two classes. In other words, if you randomly select one observation from each class, what's the probability that your model will be able to "rank" them correctly? To calculate AUROC, we used predicted class probabilities instead of just the predicted classes. As we can see in the following figure, we achieve the same ROC AUC score both for the imbalanced and the balanced dataset, which proves that picking an appropriate performance metric is crucial, such that you can have representative results.



Since we were on a stable path, we decided to train a Random Forest classifier on the initial imbalanced dataset. Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes.

In modern applied machine learning, tree ensembles (Random Forests, Gradient Boosted Trees, etc.)

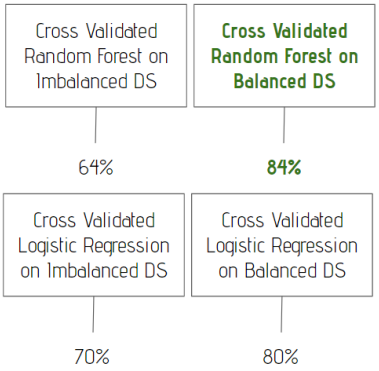
almost always outperform singular decision trees, so we jumped right into those.

7 HANDLING OVERFITTING

The ROC AUC score of 97% was surprisingly high, since it gave a profound picture of an overfitted model. The main reason for this behaviour is that we simply trained a default RF Classifier, without tuning any parameters, and the score came to a head in an instance. To avoid overfitting, we used K-fold Cross Validation. Cross Validation in general, is primarily used to estimate the skill of a machine learning model on unseen data. This method follows the next steps:

- Shuffles the dataset randomly.
- Splits the dataset into k groups (5 in our implementation)
- For each unique group:
  - Takes the group as a hold out or test data set
  - Takes the remaining groups as a training data set
  - Fits a model on the training set and evaluates it on the test set
  - Retains the evaluation score and discards the model
- Summarizes the skill of the model using the sample of model evaluation scores

So we retrained the previous models to observe any remaining overfitting issues, and we took our new measurements. This time we achieved a successful generalization of the models, as the numbers now are more ”down to earth”. The most satisfying result was the Random Forest Classifier trained on the Balanced (Upsampled) Dataset scoring an 84% Cross Validation Score.



8 HYPERPARAMETER TUNING

In machine learning, **hyperparameter optimization or tuning** is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process.

The figure that follows, shows the hyperparameters we took into consideration and tuned their values, for the Random Forest Classifier. The same approach was followed for the Linear Regression Models, the hyperparameters of which, can be found in: ”https://scikit-learn.org/”

Parameters	Values	Action
n_estimators	int	The number of trees in the forest
criterion	'gini, 'entropy'	The function to measure the quality of a split
bootstrap	boolean	If false, the whole dataset is used to build each tree
class_weight	'Balanced', None	Weights associated with classes
random_state	123, None	Random Seed
max_depth	int, None	Maximum depth of a tree
n_jobs	int, None	Number of CPU cores used

The above figure shows us some of the most important settings are the number of trees in the forest (n_estimators) and the max number of levels in each decision tree (max_depth). An efficient use of Hyperparameter Tuning is just to try out a wide range of values and see what works.

We adjusted/combined the following set of hyperparameters:

- n_estimators = number of trees in the forest
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

The optimal set of parameters that achieves the best CV score on the dataset is the following:

```
clf_3 = RandomForestClassifier(
        bootstrap=True, class_weight=None, criterion='entropy',
        max_depth=2, max_features='auto', max_leaf_nodes=None,
        min_impurity_decrease=0.0, min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
        oob_score=True, random_state=0, verbose=0, warm_start=False
        ).fit(X, y)
scores = cross_val_score(clf_3, X, y, cv=5)
print("RF Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
# 84%
```

We notice that we have set **n_jobs** to -1 such that all instances in the EC2 cluster can generate decision trees and then send them to the master node for aggregation.

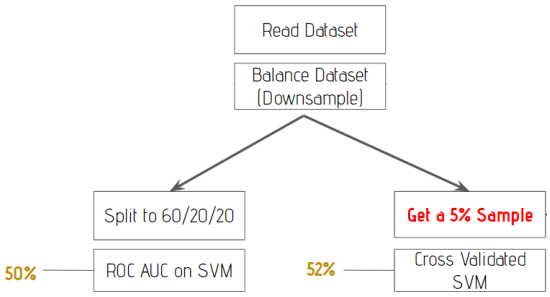
9 USING COST SENSITIVE MODELS

One of our tactics was to use penalized learning algorithms that increase the cost of classification mistakes on the minority class. A popular algorithm for this technique is Penalized-SVM. During training, we used the argument **"class_weight='balanced'"** to penalize mistakes on the minority class by an amount proportional to how under-represented it is. We trained our model using Penalized-SVM **on the original imbalanced dataset**. Due to the heavy and complex computations that this technique performs internally to reach to a result, we ran out of credits on AWS platform (for a single account). We took some actions in order to cope with this limitation. First of all, we downsampled the dataset, such that each class is represented by approximately 160 thousand datapoints (in contrast with the upsampling which enlarged the dataset instead of shrinking it). Proceeding, we tested two cases:

1. Cross Validation SVM on 5% of the downsampled dataset (such that we can achieve convergence in a reasonable amount of time)
2. Split to Train/Validation/Test and feed SVM with just one instance of the data and not Cross Validated (to avoid high convergence times)

What we can observe is:

1. The first approach led us to a 52% F-1 score. Considering we only used 5% of the Down-sampled dataset, it's not a representative result over the original dataset
2. Our second approach was to split the dataset into Train/Validation/Test, with 60/20/20 assigned respectively to each set. We trained the SVM model and used ROC AUC as the performance metric. The result was again 50%, which was discouraging
3. Using cost sensitive models might outperform our previous best results, but due to the occurrence of limitations and constraints, we could not thoroughly test and reach to a representative conclusion



10 CONTRIBUTIONS

We designed and implemented an Intrusion Detection System using Machine Learning Models and Techniques, and managed to classify traffic as intrusive or benign, with an 84% prediction score.
Our work highlights the following contributions:

- Balancing a Dataset can reinforce the signal of a minority class
- Picking a representative Performance Metric for your models is crucial
- Using Cross Validation can Prevent Overfitting

We have followed the aforementioned techniques, and ended up with a Random Forest Classifier as our best model, trained on a balanced dataset using upsampling and k-fold cross validation. Our work, seems as a promising entryway to Anomally Detection and Intrusion Detection research fields.

ACKNOWLEDGEMENTS

- This research was supported by Big Data Applications Course, CS543 @ CSD, UoC, Greece
- We want to thank the Teaching Assistants of the course for comments that greatly improved the manuscript
- We would also like to show our gratitude to Prof. C. Kozanitis who taught us all the 'Must Haves' when dealing with Big Data and Machine Learning

REFERENCES

[1]https://elitedatascience.com/
[2]https://www.unb.ca/cic/datasets/ids-2017.html
[3]https://thepacketgeek.com/scapy-p-04-looking-at-packets/
[4]https://scikit-learn.org/
[5]https://www.awseducate.com/
[6]https://towardsdatascience.com/practical-tips-for-class-imbalance-in-binary-classification-6ee29bcdb8a7
[7]https://insidebigdata.com/2017/11/30/10-tips-building-effective-machine-learning-models/
[8]https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624
[9]https://machinelearningmastery.com/automate-machine-learning-workflows-pipelines-python-scikit-learn/