Final Project Report

CS5350 Machine Learning

TYLER JONES AND VIJAY BAJRACHARYA

GitHub Repository: https://github.com/vijaymanbajracharya/cybersecurity-ml

1 INTRODUCTION

1.1 Problem Statement

Traditional cybersecurity stacks consist of multiple layers of protection. Firewalls, endpoint protection, secure gateways, and authentication protocols are a few of the many tools used to protect a company's network. Despite these efforts, the ever-evolving cybersecurity landscape continues to plague even the largest of companies in the world. To make monitoring and detection tasks easier, many businesses have started adopting intelligent and autonomous solutions into their security stacks. For our project, we want to perform a comparative analysis of various machine learning algorithms on public cybersecurity data. With so much concern over data privacy in recent years, it is important now more than ever to find an efficient solution to this problem.

1.2 Motivation

Our main motivation behind this project is to assess the viability of intelligent approaches to cyber-threat management. Moreoever, this project is part of a larger capstone project that is being sponsored by L3Harris. With this project, we are hoping to reduce the time spent on repetitive monitoring tasks and instead incentivize more important tasks such as root cause analysis and preventive actions of cybersecurity threats. Additionally, this exploratory project will serve as a baseline of comparision for neural network models that we have been developing in our capstone project.

2 OUR APPROACH

2.1 Research

We were heavily inspired by the strategies used in Li et al. (2020) so we tried our best to mimic their results as well as add onto it. One extension that we made to their approach was to make multi-class classifiers instead of just performing binary classifications. In order to detect network

intrusions, we had to first familiarize ourselves with the most common types of network attacks. Since an entire network architecture has too many components to cover in one project, we decided to focus solely on network packet data for this exploratory project. For packet data, we searched for pre-existing public datasets that had specific attack types associated with network traffic. This is where we discovered the UNSW_NB15 dataset. Following the collection of data, we used various machine learning algorithms to perform classification tasks.

2.2 Data Collection

All of the algorithms that we used in this project utilizes the UNSW_NB15 dataset. The dataset consists of raw network data packets that were generated in the Cyber Range Lab of the Australian Centre for Cyber Security based on real and synthesized attack patterns. It contains 100 gigabytes of network traffic but we will only be using a small subset of this data that was pre-sampled into testing and training data. We will be training our models on 82,332 labelled examples with 49 different features such as protocol, service, time to live, source to destination bytes etc. Moreover, the dataset is divided into 10 different attack families - normal, DoS, fuzzer, backdoor, exploit, reconaissance, analysis, generic, shellcode and worms.

The training and testing sets are stored in the form of csv files. Due to the large size of these files, we had to compress the data using gzip. Every time an analysis is run, our program will decompress and extract the csv files into a local folder and use the relative paths to train the model and calculate classification errors.

protocol	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	sload	dload	spkts	dpkts	...	attack_cat
tcp	FIN	0.668831	1008	12548	62	252	6	10	-	11387.03223	143845.0156	18	24	...	Exploits
tcp	FIN	0.668831	1008	12548	62	252	6	10	-	11387.03223	143845.0156	18	24	...	Exploits
tcp	FIN	0.668831	1008	12548	62	252	6	10	-	11387.03223	143845.0156	18	24	...	Exploits
tcp	FIN	0.626161	564	354	254	252	2	1	-	6490.34375	3960.642822	10	8	...	Recon
udp	INT	0.000011	168	0	254	0	0	0	-	61090908	0	2	0	...	Recon
tcp	FIN	0.038981	3806	51808	31	29	7	26	-	768990	10471563	64	66	...	Normal
udp	CON	0.001809	520	304	31	29	0	0	-	1724709.75	1008291.875	4	4	...	Normal
tcp	FIN	0.383638	1516	66749	62	252	2	25	http	29652.95508	1366142	16	54	...	Generic
udp	INT	0.00001	168	0	254	0	0	0	-	67200000	0	2	0	...	Recon
udp	INT	0.000003	1354	0	254	0	0	0	-	1805333248	0	2	0	...	Generic

Figure 1: Raw Network Packet Data

2.3 Data Preprocessing

The packet data contains a mixture of both numerical and categorical data. Our first goal was to unify the dataset by mapping each categorical feature to a binary vector. To achieve this, we created a pandas dataframe from our csv files and split the data into feature vectors and ground truth labels. Then, we selected all feature vectors that were categorical in nature and used the sk-learn one-hot encoder to provide a numerical representation for categorical features. The converted features were then inserted back into the training set.

2.4 Evaluation Metrics

To determine the robustness of our classifiers, we used the F_1 Score metric. We chose to use this metric to integrate the imbalanced distribution of labels into our evaluation score. Simply using accuracy as a measure yielded misleading results because of the large number of "Normal" data in our dataset. Therefore, the F_1 Score better captured the accuracy of our classifiers because it accounted for the types of misclassifications that were made. The F_1 Score was calculated using the sk-learn library which uses the following formula:

$$F_1\,Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

An F_1 Score falls within the range of 0.0 to 1.0 where 1.0 is the maximum possible score and 0.0 is the worst possible score. A classifier with a score of 1.0 means that it classified all the true positive and true negative datapoints correctly.

2.5 Models and Architectures

Each method was implemented using the scikit learn library, except for the Balanced Random Forest which used imbalanced learn. We settled on the hyper-parameters for each algorithm listed below using **KFold Cross Validation** with 5 folds as well as some manual testing. In the testing of the parameters sets, we considered generalization performance and run-time. Here is comprehensive list and descriptions of the classifiers we used to perform the comparative analysis:

1. **Decision Tree:** Our decision tree uses the CART algorithm which uses the gini-index to calculate information gain. For hyper parameters, we used a max-depth of 20 for our decision tree. We also set the min_samples_split parameter to be 40 and the min_samples_leaf parameter to be 8.

2. **Balanced Random Forest:** We used the random forest algorithm from the imbalance-learn library specifically because it allowed balanced bootstrap sampling. For hyper parameters, we set the max-depth of 3 for the estimators with a total of 100 estimators. Furthermore, we use the square root of the max number of features to be used by each estimator.

3. **Linear SVM:** The linear SVM solution in our project used the squared hinge loss function. It also solved the primal form, rather than the dual form because the dual form took a very long time to converge on our dataset. The maximum number of iterations was 1000.

4. **Naive Bayes:** We used both Gaussian Naive Bayes and Multinomial Naive Bayes variants with our data and selected the Gaussian Naive Bayes because it outperformed in all preliminary experiments.

5. **K-Nearest Neighbours:** The algorithm used for our KNN implementation was KD Tree. The classifier weighted points by the inverse of their distance. In other words, closer neighbors of a point within the same neighborhood had a greater affect on the classification of that point. The number of neighbors used for classification was 5. We also chose to set the hyper parameter leaf size to 15 to speed up the algorithm's run time.

3 RESULTS

3.1 Overall Analysis

Here is a bar-chart containing the F1 scores for each classifier for both binary (malicious vs normal) and multi-class classifications (labelled by attack family).

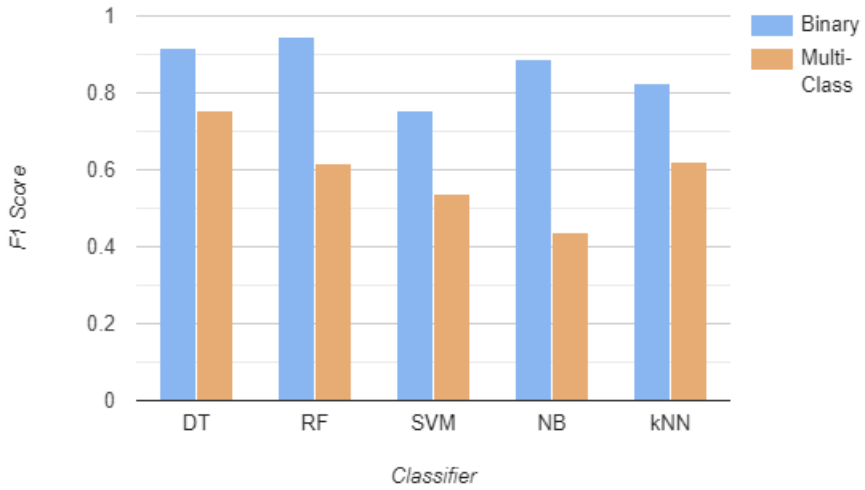

Figure 2: F1 Scores for binary and multi-class classifications

3.2 Decision Tree

Aggregate Testing Error: 0.2474
Aggregate Training Error: 0.1133
F1 Score - Multi-Class: 0.7525
F1 Score - Binary: 0.9173

Label	Correct	Incorrect	Error
Normal	54621	1379	0.024625
Backdoor	99	1647	0.943298969
Analysis	0	2000	1
Fuzzers	2104	16080	0.884293885
Shellcode	497	636	0.561341571
Reconnaissance	7341	3150	0.300257363
Exploits	20360	13033	0.390291378
DoS	7505	4759	0.388046314
Worms	61	69	0.530769231
Generic	39373	627	0.015675

Figure 3: Error rates for decision tree

3.3 Random Forest and Linear SVM

Algorithm: Balanced Random Forest
Aggregate Testing Error: 0.3847
Aggregate Training Error: 0.4054
F1 Score - Multi-Class: 0.6152
F1 Score - Binary: 0.9470

Label	Correct	Incorrect	Error
Normal	44270	11730	0.209464286
Backdoor	197	1549	0.887170676
Analysis	1116	884	0.442
Fuzzers	6900	11284	0.620545535
Shellcode	951	182	0.160635481
Reconnaissance	3233	7258	0.691831093
Exploits	11667	21726	0.650615398
DoS	310	11954	0.974722766
Worms	113	17	0.130769231
Generic	39125	875	0.021875

(a) Balanced Random Forest

Algorithm: Linear SVM
Aggregate Testing Error: 0.4617
Aggregate Training Error: 0.3703
F1 Score - Multi-Class: 0.5382
F1 Score - Binary: 0.7560

Label	Correct	Incorrect	Error
Normal	53475	2525	0.045089286
Backdoor	0	1746	1
Analysis	0	2000	1
Fuzzers	177	18007	0.990266168
Shellcode	0	1133	1
Reconnaissance	0	10491	1
Exploits	1348	32045	0.959632258
DoS	20	12244	0.998369211
Worms	0	130	1
Generic	39360	640	0.016

(b) Linear Support Vector Machine

Figure 4: Error rates for random forest and linear svm

3.4 Naive Bayes and k-Nearest Neighbors

Algorithm: Gaussian Naive Bayes
Aggregate Testing Error: 0.5625
Aggregate Training Error: 0.5911
F1 Score - Multi-Class: 0.4374
F1 Score - Binary: 0.8870

Label	Correct	Incorrect	Error
Normal	29275	26725	0.477232143
Backdoor	0	1746	1
Analysis	0	2000	1
Fuzzers	2249	15935	0.876319842
Shellcode	18	1115	0.984112974
Reconnaissance	4884	5607	0.534458107
Exploits	795	32598	0.976192615
DoS	30	12234	0.997553816
Worms	11	119	0.915384615
Generic	39438	562	0.01405

(a) Gaussian Naive Bayes

Algorithm: k-Nearest Neighbors
Aggregate Testing Error: 0.3785
Aggregate Training Error: 0.0675
F1 Score - Multi-Class: 0.6214
F1 Score - Binary: 0.8267

Label	Correct	Incorrect	Error
Normal	50516	5484	0.097928571
Backdoor	9	1737	0.994845361
Analysis	50	1950	0.975
Fuzzers	1710	16474	0.905961285
Shellcode	135	998	0.880847308
Reconnaissance	3680	6811	0.649223144
Exploits	8845	24548	0.735124128
DoS	4932	7332	0.597847358
Worms	2	128	0.984615385
Generic	39093	907	0.022675

(b) k-Nearest Neighbors

Figure 5: Error rates for naive bayes and knn

3.5 Conclusions

From the results above, we can see that even though most classifiers were expressive enough to separate malicious and normal network traffic, identifying individual attack families is still a difficult task. The poor results from the linear SVM show us that the data is highly non-linear. This coupled with the long run-times leads us to the next logical step of using neural networks instead of traditional machine learning algorithms. It is worth mentioning that the decision tree and the balanced random forest classifiers performed fairly well on both binary and multiclass classifications.

4 PROJECT CONTINUATION

Given more time to work on this project, one of the first improvements we would make to our current experiments is to balance the datasets. While we tried both undersampling and oversampling methods, we were not able to get competitive scores with the balanced data. With more time, we would be able to research more about how we should be resampling the data to fix the disproportionate amount of minority data in our dataset. Another avenue to explore would be feature selection using statistical methods like chi-squared test or variance thresholds. Currently, we are using all of the features in the dataset but some features which have low variance might not be useful for the classifier and those features could be omitted entirely.

Since this exploratory project is related to our work in the capstone project, we will be continuing to use the results from this project in the future. Eventually, our goal is to use graph neural networks for node classification and link prediction on this dataset.

REFERENCES

1. Li, Kun, Huachun Zhou, Zhe Tu, and Bohao Feng. "CSKB: A Cyber Security Knowledge Base Based on Knowledge Graph." In International Conference on Security and Privacy in Digital Economy, pp. 100-113. Springer, Singapore, 2020.

2. Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In 2015 military communications and information systems conference (MilCIS), pp. 1-6. IEEE, 2015.

3. Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set." Information Security Journal: A Global Perspective 25, no. 1-3 (2016): 18-31.

4. Sarhan, Mohanad, Siamak Layeghy, Nour Moustafa, and Marius Portmann. "Netflow datasets for machine learning-based network intrusion detection systems." arXiv preprint arXiv:2011.09144 (2020).