Classifying Botnet Traffic on IoT Devices

Jake Nachlas, James Dugle, Ancelmo Polanco

CS 334 - Machine Learning

Emory University

I. Abstract

The purpose of this study is to analyze various machine learning models by comparing their ability to accurately classify botnet attacks on Internet of Things (IoT) devices. Being able to classify these botnet attacks accurately and quickly is important to a society with increasing security risks. We implemented Decision Trees, Random Forests, Naive Bayes, Logistic Regression, XGBoost, KNN, and Neural Networks. In addition to comparing each of these models, we also compared various pre-processing techniques for each of the models including scaling the data, feature extraction, and feature selection.

The results showed us that XGBoost, Random Forests, and Decision Trees were the best performers with the dataset we used across the board using the raw, unprocessed data. Feature preprocessing did, however, improve the models on average. The neural network greatly improved when the data was preprocessed, for example. We concluded from our results that both benign and malicious attacks can not only be detected, but easily classified. It is important to continuously develop these machine learning models to be able to predict these attacks before they happen or a lot of valuable data can get into the hands of the wrong people.

II. Introduction

As technology advances, all of our personal devices are becoming more and more connected to each other. This can be beneficial to many people as the connectedness of our devices allows for things such as smart homes, powerful voice assistants, and personal automation. With this increased connectivity there are many security risks. We are aware of the maliciousness of hackers, whether you've heard the news surrounding Russia's involvement in the election, or Snowden releasing sensitive files to the public. From large scale events like these to the Target or Capital One incidents to getting one-click phishing emails, we need to be more aware of the threats that are out there. These incidents can cost people thousands and companies hundreds of thousands if not more. And if it's not just money, it is highly sensitive information that should have been protected very carefully. At a country-wide level, corporate level, and individual level, we all need to take measures to increase our own safety. Soon there will be a reality where our cars, homes, and security systems are all vulnerable to attacks from a malicious entity.

We plan to attack this problem by using machine learning algorithms to be able to identify the malicious entities on the Internet of Things (IoT) devices before the attacks happen.. The Internet of Things is a complex system of interconnected devices that allows data to be transferred from device to device to bypass both human-to-computer interaction as well as human-to-human interaction. The reason we are testing multiple models is because there is no perfect model, so by looking at a variety of algorithms with a variety of preprocessing techniques, we hoped to be able to find the best solution for predicting both benign and malicious activity of botnets. The key contribution we hope to make is a classifier that can very highly accurately predict the botnet activity by both intent and class type. If successful, we will have made an important step towards cyber safety because being able to quickly and accurately classify botnet attacks that compromise IoT devices can be hugely beneficial to making our world a safer place.

III. Background

The basis of our study is drawn from *N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Encoders* by Meidan et al. The purpose of their study was to develop a new method for detecting attacks from compromised IoT Devices. The authors opted to use autoencoders, SVM, isolation forests, and LOF models to classify the data, while we are using Decision Trees, Random Forests, Naive Bayes, Logistic Regression, XGBoost, KNN, and Neural Networks. The authors of this paper performed their evaluation with real traffic data as opposed to simulated data. They infected nine commercial IoT devices to evaluate the autoencoding method. They used deep autoencoders as well as maintaining a separate model for each IoT device as their base anomaly detector.

Dollah et al examines HTTP IoT attacks in their research paper, *Machine Learning for HTTP Botnet Detection Using Classifier Algorithms.* They implemented machine learning algorithms to be able to detect HTTP botnets. One takeaway from this study is that KNN, Decision Trees, and Naive Bayes models are highly effective in being able to detect botnets attacks. We will be implementing each of these models in our own study.

Lastly, Ayush Kumar and Teng Joon Lim define EDIMA and demonstrate its value in their research, *EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques.* EDIMA is a distributed modular solution to detecting IoT malware network activity. They utilized machine learning algorithms for edge devices' traffic classification, a policy module, an optional packet sub-sampling module, and a packet traffic feature vector database. Their classifier collects incoming traffic samples and classifies them based on their model. They utilize Naive Bayes, Decision Trees, and Support Vector Machines.

IV. Methods

1. Preprocessing

We utilized three preprocessing techniques. The reason we did so is because many machine learning models perform better or converge more quickly when the features are on similar scales or are close to normally distributed. We wound up with four datasets. One with each of the following preprocessing techniques performed on the raw data, and the fourth is the raw dataset with no preprocessing done.

The first technique we used is a standard scaler which standardizes a feature by subtracting the mean of that feature and then scaling to unit variance. The unit variance is calculated by dividing all the values by standard deviation. Scaling the data this way results in a distribution of the feature where the standard deviation is equal to 1 and variance is equal to 1. The reason we used a scaler is because the data has a lot of numerical features but the range of numbers amongst features differs greatly so we needed to scale the features.

The dataset we used has a total of 115 features. This is a great number of features and some of them are likely to be highly correlated to each other. Therefore, we needed to reduce the number of features. For our preprocessing we performed two major techniques. The first technique was feature elimination. We decided to use a Pearson correlation matrix to calculate the correlation between all the features. Then we went through all pairs of features and if the correlation value was equal to or greater than 75% we would randomly drop one of the features. A correlation value of 75% was chosen because after viewing the heatmap of the raw data and looking at all the correlation values between the features, a good amount of the features had correlation values at this threshold and above. This threshold helped us achieve our goal of maximizing the reduction of dimensionality in order for our models to perform better.

The last preprocessing technique we used was feature extraction. For this technique we used a principal component analysis. Principal component analysis (PCA) transforms a set of possibly correlated variables into a set of linearly uncorrelated variables that are combinations of the old variables. This new set of variables is referred to as the principal components. The PCA algorithm works by first calculating a matrix that first summarizes how variables are related to each other, then breaks down the matrix into direction and magnitude. We applied PCA to our dataset and retrieved the number of principal components that gave us and explained variance of about 95%.

We performed all these preprocessing techniques/methods because we realized the different preprocessing techniques perform better on different models. We didn't want to introduce bias into our study, so we tried to include a variety of preprocessing techniques that performed well on a majority of our models.

2. Models

We then implemented seven machine learning models for this project. We utilized a 75-25 split for the training and testing datasets for each model.

The first is a Decision Trees model. The Decision Trees model implements an algorithm that is based on a greedy algorithm that continuously devices the feature space into rectangles. It does this by calculating which feature split will lead to the greatest loss in entropy, splitting the data into two groups, then repeating. We chose to implement this model because decision trees are robust, interpretable, and fast to predict.

The second model we implemented is Random Forests. Random Forests is a bagged classifier using many decision trees, where each tree only considers a random group of the features. The final prediction is obtained by aggregating the predictions of all of the trees within the forest. We chose to use this model because it is one of the most accurate general purpose learners available, it is easy to train and tune, and it is less prone to overfitting than decision trees.

Third, we implemented Naive Bayes. Naive Bayes is a classifier that applies Bayes Theorem along with the assumption of feature independence to make predictions. The idea behind this is that by assuming feature independence, we can calculate the joint probability distribution of all of the features by simply treating the prediction as a linear combination of the feature space. We chose this because it generally performs very well in most scenarios, is fast to train and classify, and is not sensitive to irrelevant features.

Logistic regression was the next model we used. Logistic regression combines a logistic function with cross entropy loss in order to calculate the probability of a certain class. In other words, it gives us the probability of an event occurring. In our case, it utilizes the probability of one of the specific classes in our dataset occuring. We chose it because we wanted to see how linear classifiers performed on the data.

We implemented XGBoost next, as it is one of the leading state of the art classifiers based on the idea of gradient boosting. Gradient boosting is the application of many small and weak regression trees, which are trained on the residuals of the tree prior to them. Combined, they produce a powerful committee by the end of the chain. We chose this because it is a powerful algorithm used by most of the top learning models nowadays, and is also very fast compared to other industry leading classifiers.

Next we implemented K Nearest Neighbors (KNN), which is an algorithm that simply bases its predictions off of a given input by looking at the instances in the training data that most resemble our input. We chose this because KNN is a very simple and straightforward learner, which gives us a baseline to compare the performance of other models.

Lastly, we used a Neural Network. Neural networks are classifiers inspired from the neurons in the human brain. They are composed of multiple interconnected layers of neurons. The data is fed into one side, passed through the layers, and will output an answer depending on the many weights in between. They are trained by adjusting the weights after every correct and incorrect classification. We chose these because neural networks are very good with nonlinear boundaries, are fast to classify, and are most effective when trained on large amounts of data (which we have).

V. Experiments/Results

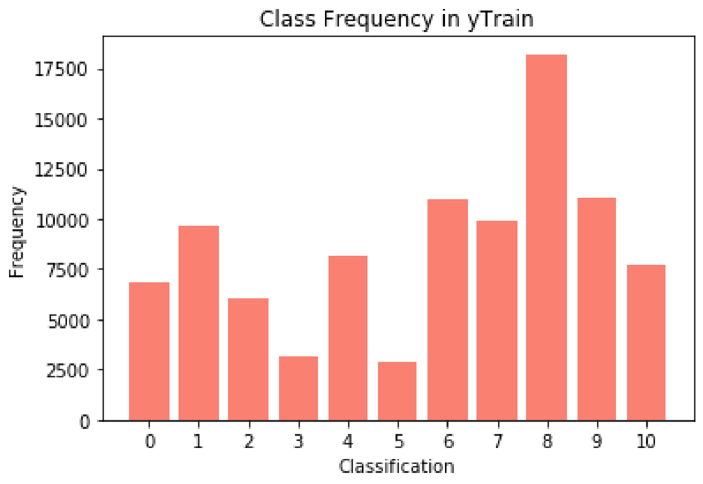
1. Data Description

The dataset comes from the University of California, Irvine. Whereas prior experimental studies on the detection of IoT botnets or IoT traffic anomalies typically relied on emulated or simulated data, this dataset allows us to use real traffic data, gathered from nine commercial IoT devices infected by authentic botnets from two families in an isolated network. The two botnets are Mirai and BASHLITE, which are two of the most common IoT-based botnets. Mirai is a botnet that turns IoT devices that run Linux into remotely controlled bots that can be controlled on a wider network of bots. This network of bots can be used for large scale attacks. The primary targets of Mirai are online consumer devices like IP cameras and home routers. BASHLITE also infects Linux systems, but in this case it utilizes the network to launch distributed denial-of-service (DDoS) attacks.

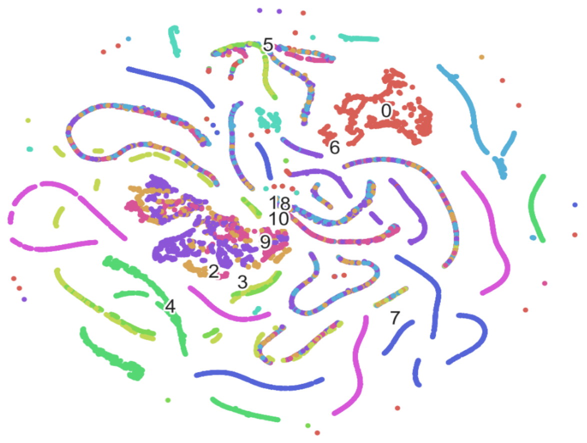
As previously mentioned, the Mirai and BASHLITE attacks were carried out on nine IoT devices. These nine devices are Danmini doorbell, Ecobee thermostat, Ennio doorbell, Philips B120N10 baby monitor, Provision PT 737E security Camera, Provision PT 838 security Camera, Samsung SNH 1011 N webcam, SimpleHome XCS7 1002 WHT security camera, and SimpleHome XCS7 1003 WHT security camera.

There are 115 total features in the dataset. Some examples of the features are a stats variable that summarizes recent traffic from the packet's host, another feature of the stats summarizing recent traffic from the packet's host to the packet's destination host and a summary of the jitter of the traffic going from the packet's host to the packet's destination host. It also contains other common statistic features like standard deviation, mean, weight of the stream (number of items observed in recent history), the covariance between two streams, the root sum squared of the two stream's variances, and more.

Below is a figure that shows the distribution of classifications in the training dataset. There are 11 total classifications-- five classifications for the Mirai attacks, five classifications for the BASHLITE attacks, and one classification for benign traffic. We recognize that there is a slight class imbalance, it is not great enough of an uneven distribution to cause concern.

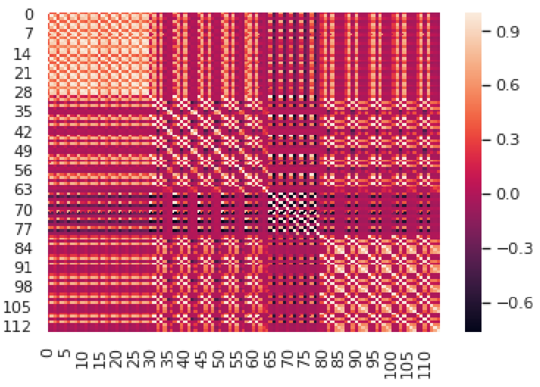


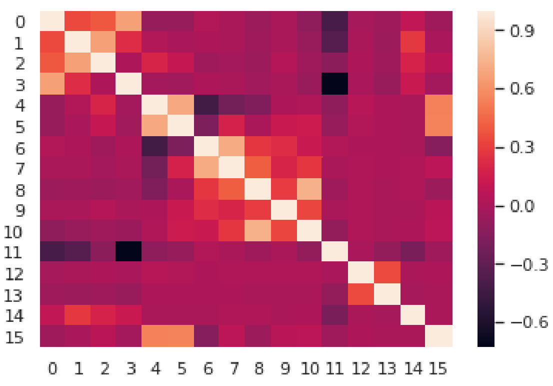
Here is a preliminary t-SNE graph of our data. We can see here that a lot of these strands are homogenous in color which implies that there is distinguishability between the classes.



2. Preprocessing, Feature Extraction, Feature Selection

The first step in preprocessing the data was to create different methods to handle all the different preprocessing techniques that were gonna be used in the study. To standard scale the data the sklearn module was loaded in and the data was transformed into the scaled data. After the transformation, the data was exported with new names that conveyed the preprocessing technique applied to it. Next, for the feature elimination, the first step was to load the sklearn module and use it to calculate the Pearson correlation matrix shown below on the left. Then, we decided to examine each value in the correlation matrix and mark the column false if the correlation value was over 75%. We chose 75% because we tested it at 25%, 50%, and 75%, and 75% in our preliminary testing performed the best. Once all the necessary columns were marked, we would delete them from the original dataset and export the transform data with the appropriate name. The resulting Pearson correlation matrix is shown below on the right. For the final preprocessing step, we loaded the sklearn model and loaded the PCA module with the variance set to 95%. We fitted the PCA to the training data and then transformed both the training and testing data using the PCA model. Once we finished transforming the data, we export the data with the appropriate preprocessing technique performed on it.





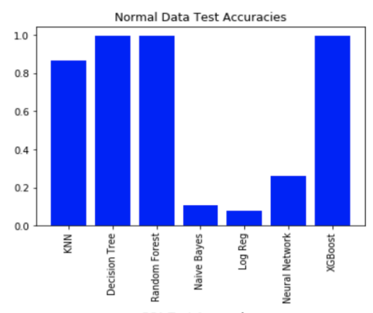
For parameter tuning, we have to alter our approach for each model separately. During hyperparameter tuning, we further split the training dataset 75/25 into a training and validation set to evaluate hyperparameter performance. For each individual model, we manually evaluated which parameters best affected that specific model. After we decided on which parameters we were going to tune, we did a manual search for the best ranges for each hyperparameter of a single model. We then modularized the parameter tuning by running a model over the decided range for each specific hyperparameter. We recorded some general stats such as training accuracy, test accuracy, macro f1 score, and time taken, but we also recorded model specific hyperparameters. We repeated this process for each model and adjusted accordingly. For example, let's say the model we were using was a decision tree. The hyperparameters we decided to tune were criterion, max depth, and min leaf samples. We would write a program to run a single instance of the model over a decided range for each of the hyperparameters mentioned previously. Finally, for each instance, a model was run we would record the criterion, max depth, min leaf samples, and the general stats described previously.

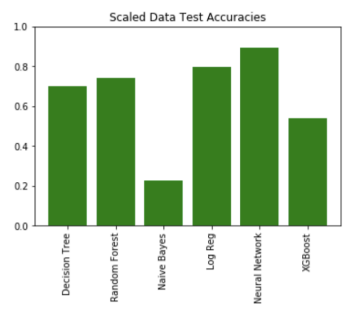
3. Model Choices

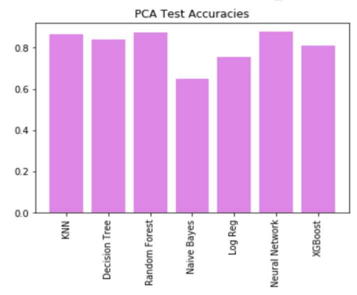
The models that we selected as the best were evaluated using test accuracy and macro F1 score, as well as the average classification time of the models. The parameters were determined using the above method. The top three models that performed best were actually on the raw data. The decision tree had a test accuracy of 0.996 and a macro f1 score of 0.995. The random forest model had a test accuracy of 0.996 and a macro f1 score of 0.997. Lastly, the XGBoost model had a test accuracy of 0.996 and a macro f1 score of 0.994. Although all 3 of these models are very similar in terms of accuracy and F1 score, when looking at the classification and the time it takes to run, it is easy to see that the decision trees are our overall top performer. The reason for this is because decision trees are very simple which is why they had the fastest classification time which is very important for real time monitoring. We believe the decision tree using the raw data likely performed the best because of the size of our dataset. Decision trees are prone to overfitting, but with massive amounts of data this can be combated. An honorable mention should also be given to the neural networks, which consistently performed the best in every preprocessed dataset. Although the models using the preprocessed data were not among the best performers, all three of the preprocessing techniques improved the performances of the neural networks, logistic regression, and naive bayes.

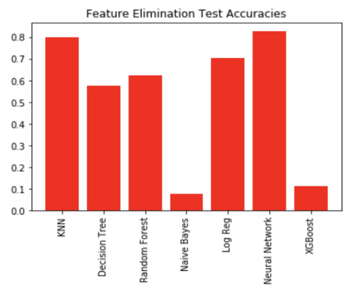
4. Empirical Results

Below are the results from each of the models run on each of the preprocessed datasets.

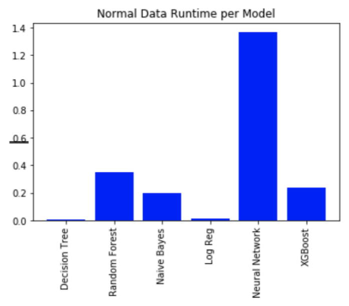


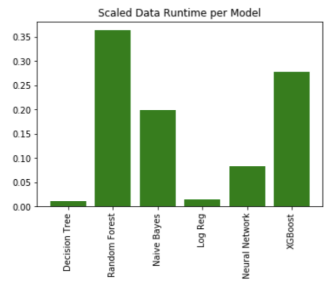


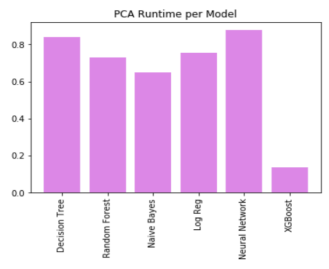


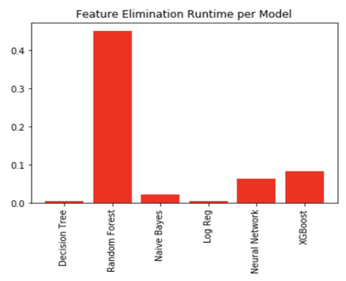


We can see here that on average, the PCA dataset yielded the best results, but they were not as high as the best results from the raw dataset. As mentioned before, the Decision Trees, Random Forests, and XGBoost models performed the best using the raw data. The way we decided on decision trees for the top performer was using the runtimes shown below.









In the top left visualization it is clear that the Decision Trees model runs more quickly amongst the three top performers. We did not include KNN in these runtime comparisons because it took a substantially longer time to run and made it harder to compare the rest of the models. The reason the visualizations have different scales is because it is easier to see the comparisons within datasets.

VI. Discussion

A key finding from our experiment is that the best performing model was the Decision Trees model when used on the raw, unprocessed data. It had a 99.5% test accuracy and ran in less than 0.01 seconds. This goes to show that it is possible to detect both benign and malicious botnet activity at least for these ten classes provided in the dataset with very high accuracy and at a very quick computational speed. Another key finding from our experiments was that preprocessing the data with one of our preprocessing techniques did raise the test accuracies on average. This shows that although the decision tree model on the raw data did outperform the rest of the models, preprocessing is still valuable for the most part in making the data more valuable and usable for a good analysis.

Lastly, it is important to note the speed at which our models ran. With the exception of the KNN model, our algorithms ran in a very quick time. This is important because the goal of this experiment was to be able to detect these malicious entities in real time. The reason this is important is because security often takes a back seat to convenience nowadays when talking about consumers and the excitement that comes with shiny new technology. We have shown with our experiments that it is possible to monitor and classify these benign and malicious entities in real time, which means we can keep the convenience of our interconnectedness while making sure our safety is not compromised.

Of course, with a test accuracy as high as 99.6%, it is important to consider the prospect of data leakage. Data leakage is the idea that advantageous information that should not be included in the training set gets included, leading to artificially and unrealistically high model performance. To test for data leakage, we examined several scenarios. The first scenario is including the test data in the training data. The reason this is not an issue is because we used the industry-standard holdout technique to make sure that test and training data were always separated. Additionally, we did not tune the parameters based on the test dataset, instead further splitting the training data into training and validation datasets for parameter tuning. We also made sure to choose our principal components and most relative features based on the training data, not the test data.

Moving forward, we would like to see how our models perform on the traffic of other IoT devices. Our training data comes from a variety of devices so it would be very interesting to see how well it generalizes. We would like to see how our models perform on binary classification of benign traffic versus botnet attacks that we have not trained the models on. Our models will obviously not be able to classify the type of attack like what was done in this experiment, but may be able to distinguish unknown attacks from benign traffic. Finally, since our T-SNE visualization seems to suggest some distinguishability, we also plan to use other nonlinear dimensionality reduction techniques and measure how they affect model performance.

We could also use ensembles in an attempt to further improve classification accuracy. However, we question the necessity for this. Our decision tree performs remarkably well and will be much faster than any ensemble, as ensembles are bottlenecked by having to run multiple classifiers as well as the overhead cost of aggregating these together. We value fast learners, and when we are doing real time monitoring, an algorithm that can give us 10 predictions at 99.6% confidence is much more valuable than one that can give 1 prediction at a marginally higher confidence in the same timeframe.

VII. Contributions

Ancelmo's primary contribution was to research and implement preprocessing techniques to get the datasets we would feed into the models. James focused on the implementation of the models. Jake's contributions were the visualization of the models and the results from the models, early data analysis, and leading the report.

VIII. Code

https://github.com/apolanco18/ml-final-project

IX. Sources

Meidan, Yair. "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders." *IEEE Pervasive Computer,* Vol. 13, No. 9, July - September 2018.

Dollah, Rudy Fadhlee Mohd. "Machine Learning for HTTP Botnet Detection Using Classifier Algorithms." *Semantics Scholar,* Vol. 10, No. 1-7, 2017.

Kumar, Ayush. “EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques.” *Arxiv.* June 24, 2019.