**Malicious URL Identification**

Mark Gerken

Dr. Kevin Scannell

Machine Learning – CSCI 4930

**Project Description**

This project strove to achieve the highest accuracy possible in the identification of malicious URLs. The dataset used for training and testing was retrieved from <https://www.sysnet.ucsd.edu/projects/url/>. The URLs are contained in an uncompressed archive of an anonymized 121-day dataset consisting of a total of about 2.4 million URLs and up to 3.2 million features.

**Data Exploration**

The data archive contains 121 independent files. Each is a .svm file corresponding to one day of the dataset, and labeled DayX.svm (where X is an integer from 0 to 120). URLs are labeled +1 if they are malicious, and -1 if they are benign. Each SVMlight file consists of a Scipy sparse matrix for the data and a Numpy ndarray containing the corresponding labels for each data vector. The features are reported to correspond to different values such as if the URL has been reported on blacklists, properties gathered from the associated IP address, WHOIS information, domain name, and even geographic properties.

I reviewed three different research papers (appropriately sourced at the end of this report) to see what I could learn about the data set features. Unfortunately, while all three papers generally mention the features, none give specifics on each feature’s exact meaning. I explored the possibility of trying to weight certain features higher than others, but with over 3.2 million, it became quickly apparent this would not give any worthwhile results.

Additionally, upon manual review of the data, I saw that the data had already been normalized as well. Every feature falls within the range between 0 and 1, many appearing as binary features and some appearing as high precision floats. From this examination, I decided to not perform any scaling or transformations to the vector.

**Evaluation Process**

The dataset was immediately segregated for training and testing. The first 90 days (Day0 – Day89) of the data was taken for training. The files corresponding with days 90-120 were moved to a different location to be used exclusively for testing. Most statements of accuracy in this report should be taken as a close estimate for the performance, unless explicitly stated otherwise. Due to the sheer size of the working dataset, I was unable to run full tests, as in training on all 90 files and testing on all 31 files. Instead, training and testing was done by taking stochastic subsets of the data. I selected and reduced the first five files of both the training and test data to only include the first 3000 data points, which I will refer to as my *sample* files. This reduction drastically reduced runtime, and as such were the files used for minor tweaks and experimentation. Periodic larger tests were also performed in which two different arrays were randomly generated with a uniform distribution: one integer array to select training files and another integer array to select testing files. These arrays were used to select which (full-sized) files would be used. This randomness allowed me to test various files over many trials with minimal favoritism or bias.

**Process and Results**

**SVM Classifier**

The first objective was to establish a bare minimal baseline. I first tested using the sample files to test the different base SVM classifiers with default parameters.

Training Data: (Sample) Day0, Day1, Day2

Testing Data: (Sample) Day90

RBF: 68.22%

Linear: 97.03%

Poly (5 degree): 69.23%

Sigmoid: 69.23%

From these results, pursuing a linear classifier further was obviously the best choice. I wanted to get a better breakdown of where the linear classifier was making its mistakes, so I reran the classifier training against the full Day0, Day1, and Day2 file. I then tested against the full Day90 file, and created a confusion matrix based on those predictions. Columns are based on true values while rows are based on classifier prediction.

Linear SVM Classifier

Training Data: Day0, Day1, Day2

Testing Data: Day90

Accuracy: 97.86%

|  |  |  |
| --- | --- | --- |
|  | MALICIOUS | BENIGN |
| POSITIVE | 13322 | 218 |
| NEGATIVE | 210 | 6250 |

**KNN**

I then experimented with a KNN implementation. Stand alone, the KNN classifier with 15 neighbors gave the following results (modifying the neighbors from 5-20 had nominal effect).

Training (Sample): Day0, Day1, Day2

Testing (Sample): Day 90

Accuracy: 90.73%

|  |  |  |
| --- | --- | --- |
|  | MALICIOUS | BENIGN |
| POSITIVE | 2005 | 72 |
| NEGATIVE | 206 | 717 |

Seeing a degradation of performance, I didn’t pursue this much further. Interestingly enough, the KNN classifier produced a much higher rate of false negatives than false positives. I intended to explore if combining the KNN classifier with some other method could potentially help mitigate that bias.

**Forests of Randomized Trees**

I decided to step away from the single classifiers and test the data against some ensemble methods using scikit-learn’s forest of randomized trees classifier.

Random Forest Classifier

Training (Sample): Day0, Day1, Day2

Testing (Sample): Day 90

RFC (5 estimators): 90.53%

RFC (7 estimators): 93.43%

RFC (15 estimators): 92.70%

As can be seen from the data, the RFC actually performed worse as I increased the amount of random estimators past a certain threshold. I also tried a quick experiment with “extremely randomized trees”, similar to the forest of randomized trees but instead draws threshold splits at random instead of computing a best split. However, this experiment performed under identical parameters and with 8 estimators performed slightly worse with an 88.67% success rate, so no further experimentation with that classifier was attempted. At this point, now equipped with three different reasonably functioning classifiers (Linear SVM, KNN, and Randomized Trees), I wanted to try a different type of ensemble method and implement a voting classifier. The confusion matrices for both KNN and Randomized Trees consistently yielded significantly more false negatives than false positives giving me doubt a vote would mitigate this.

**Voting Classifier [SVM, KNN, RT]**

The voting classifier was configured to utilize the Linear SVM, KNN, and Randomized Trees classifiers. Due to the high rate of false negatives in the KNN and RT classifiers, I made the voting classifier use a “soft vote”. The linear classifier was parameterized to report probabilities instead of binary classification. Finally, the voting classifier was tested on the same sample files, giving twice the weight to the SVM compared to KNN and RT. This yielded a promising result of 93.44% accuracy, with a reduced amount of false positives. I wanted to test this model further and opted to run this classifier on the full SVM files, but giving a weight of three to the SVM. Unfortunately, after a full hour elapsed, I ended the trial. I ultimately tested the sample files again but with the higher weight on the SVM classifier.

Voting Classifier

Soft Vote - [SVMx3, KNNx1, RTx1]

Testing Data (Sample): Day0, Day1, Day2

Training Data (Sample): Day0

Accuracy: 96.43%

While initially reassuring to see a rise in accuracy, I realized continuing this any further would be pointless. Any further increase of the weight of the SVM classifier would essentially reduce the voting classifier to be equivalent to the linear classifier as it dominated the others. So I decided to make one last attempt to beat the standard linear SVM classifier with a stochastic gradient descent classification.

**Stochastic Gradient Descent**

From brief experimentation with the stochastic classifiers, I was able to get best results from a stochastic gradient descent classifier using the Huber loss function with a max iteration count of 1000. The predictions result in a float between -1 and 1, so I had to round these results to a label of either -1 or 1. Rounding from a threshold of zero yielded a significantly higher rate of false positives. So I decided to tweak this hyperparameter and round anything below 0.5 to -1 and anything above 0.5 to 1. This classifier yielded the following results:

Stochastic Gradient Descent

loss: Huber, max\_iter: 1000

Activation: > 0.5

Training (Sample): Day0, Day1, Day2

Testing (Sample): Day90

Accuracy: 96.87%

|  |  |  |
| --- | --- | --- |
|  | MALICIOUS | BENIGN |
| POSITIVE | 2031 | 46 |
| NEGATIVE | 48 | 875 |

After seeing these results, I decided I wanted to reattempt the voting classifier combining my two best performing classifiers.

**Voting Classifier [SVM, SGD]**

I integrated both the linear SVM classifier and the stochastic gradient descent classifier into an evenly weighted vote. I wanted to test this on the full file sizes. I started with training on five full sample files and testing two. Unfortunately after several unsuccessful attempts (i.e. insufficient hardware and processing capabilities), I eventually reduced to training and testing on all the sample files. The results of this final trial are below.

Voting Classifier

Probabilistic “Hard” Vote – [SVM, SGD]

Training: (Sample) Day0, Day1, Day2, Day3, Day4

Testing: (Sample) Day90, Day91, Day92, Day93, Day94

Accuracy: 97.27%

|  |  |  |
| --- | --- | --- |
|  | MALICIOUS | BENIGN |
| POSITIVE | 9936 | 165 |
| NEGATIVE | 244 | 4655 |

**Conclusion**

I tested the linear classifier once more with the exact conditions I tested the SVM and SGD voting classifier. The test yielded an accuracy of 97.23%. From these results, it seems the two classifiers perform quite similarly. Unfortunately, I was not able to vastly improve upon the linear classifier simply because it performed so well from the beginning. From the research papers, they reported accuracy of over 99% using methods of linear regression. I did not make any major efforts to recreate their work, but rather used them for inspiration for the different algorithms I wanted to try. To conclude, the most effective classifier my experiments yielded was a voting classifier consisting of two different estimators. One estimator was a linear SVM and the other was a stochastic gradient descent classifier with a Huber loss function.