Anomaly Detection Challenge 5

Vishal Bhalla Matriculation No: 03662226
Nithish Raghunandanan Matriculation No: 03667351

Malware Classification

January 18, 2016

1 Abstract

The goal of this challenge was to implement a machine learning algorithm (Multi-class Classification) based on behavioral data (function call stacks) to classify malware into predefined families (0...9) or to determine that a sample does not belong to any family (10). The samples are identified by the md5 signature.

2 Data Preprocessing & Analysis

The data consisted of the output of cuckoo on the samples.

2.1 Structure of Function Call Stacks

Each sample can have a varying amount of function calls. There are 5729 different function calls in the training set. Each sample can have from 0 to 9353 of these function calls in a wide variety of combinations.

2.2 Handling Empty Training Samples

There were 257 training samples that were empty. They were not removed as there were similar samples in the test samples. Instead they were encoded as null vectors.

2.3 Handling Missing Test Samples

There were 30 samples in the test data that did not have any information. We tried to obtain the samples from virustotal and malwr. But we could not get any data for the samples except for one sample which was not the cuckoo results. So, we checked the presence of these samples

in the Challenge 3 data sets. Out of the 30 samples, 24 samples were present in the test data. So, we used the best classification output results(with a classification accuracy of 96.778% from the Challenge 3. After the initial classification, the missing test cases outputs were replaced by the ones from Challenge 3.

3 Feature Engineering

There were many qualitative features in the data set which we had to map to numerical features.

3.1 Bag of Words

In document classification, a bag of words is a sparse vector of occurrence counts of words; that is, a sparse histogram over the vocabulary. Here, we represent each sample by the counts of the occurrences of the functions in it. This leads to a 5729-dimensional vector for each sample. This leads to the loss of the order of function calls though.
Example: A->C can be represented as [1, 0, 1] where the total number of functions is 3.

3.2 Term Frequency-Inverse Document Frequency(TF-IDF)

TF-IDF, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Here, we represent each sample by the frequency of different functions in it normalized by its occurrence across other samples. This also leads to a 5729-dimensional vector for each sample. This also leads to the loss of the order of function calls though.
Example: A->C can be represented as [1/Occ(A), 0, 1/Occ(C)] where the total number of functions is 3 and Occ(X) represents the occurrence of X across different samples.

3.3 State Transitions

Here, we represent each function by a state from 1. Each sample is represented by a sequence of states(functions in the call stack). This preserves the order of function calls in the sample. The vectors have the dimensionality of the largest function call stack. In our case, it was 9353. The remaining fields apart from the state transitions are filled by 0s.
Example: A->C can be represented as [1, 3, 0, 0] in a vector of size 4.

3.4 Stacking of Features

We also tried stacking a combination of the different features together like stacking the bag of words with the state transitions. This leads to an increased dimensionality, but possibly with more information.

3.5 N Gram

N Gram represents contiguous sequence of n items from a given sequence of text here, which are basically the function calls. We tried Mono, Bi, Tri Grams and this comprised of a total of 78310 words in the vocabulary. It did not make much of a difference in the results in comparison to all the above features. Moreover it gave a memory error when the n-grams were increased. In hindsight, we should have considered taking the intersection over the train and test datasets which could have drastically reduced our vocabulary and hence compute time.

4 Model

The essential structure of our data involved a stack trace of function calls and hence can be viewed as a statistical learning problem. Some of the methods that have been developed within the machine learning research community for addressing such problems include sliding window methods, recurrent sliding windows, hidden Markov models, conditional random fields, and graph transformer networks [1]. We tried incorporating the above models to our dataset using 3rd party available libraries in Python, but failed because of lack of knowledge on the different input data parameter format and lack of proper documentation, examples and tutorials.

We tried fitting different models to our data viz. Random Forest, Ada Boost, Naive Bayes, Multi Class SVM and KNN Classifiers. Some of the observations and conclusions during Model Selection over the cross-validation dataset are as below:

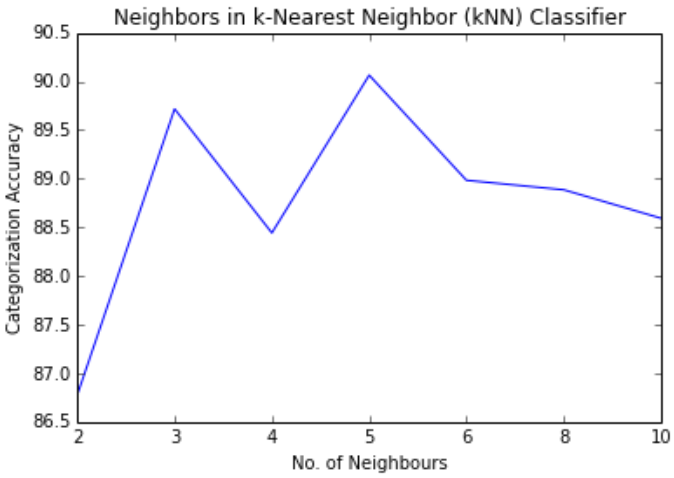
1. Random Forests
   We used Random Forest (fig. 1c) with Entropy and 50 trees as a criterion. It performed better with the bag of words as features.

2. k Nearest Neighbors
   We tried k Nearest Neighbors (kNN) (1a) and found 5 to be the optimal number of neighbors. It worked better on TF-IDF as features.

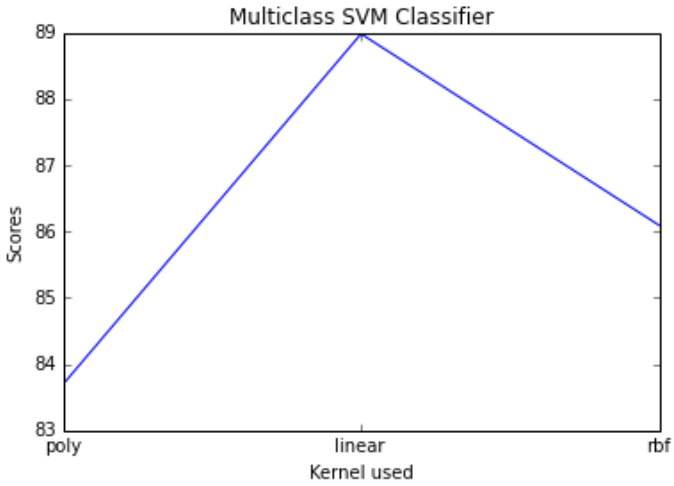
3. Multi Class SVM
   For a Multi Class SVM (1b) model, the linear kernel gave the best results over all the parameters we tried. This was due to the fact that the features were linear in nature.

4. Adaboost
   The Adaboost classifier scores performed poorly with TF-IDF as features but we got near about same results as compared to kNN and Random Forests with Bag of words as features.

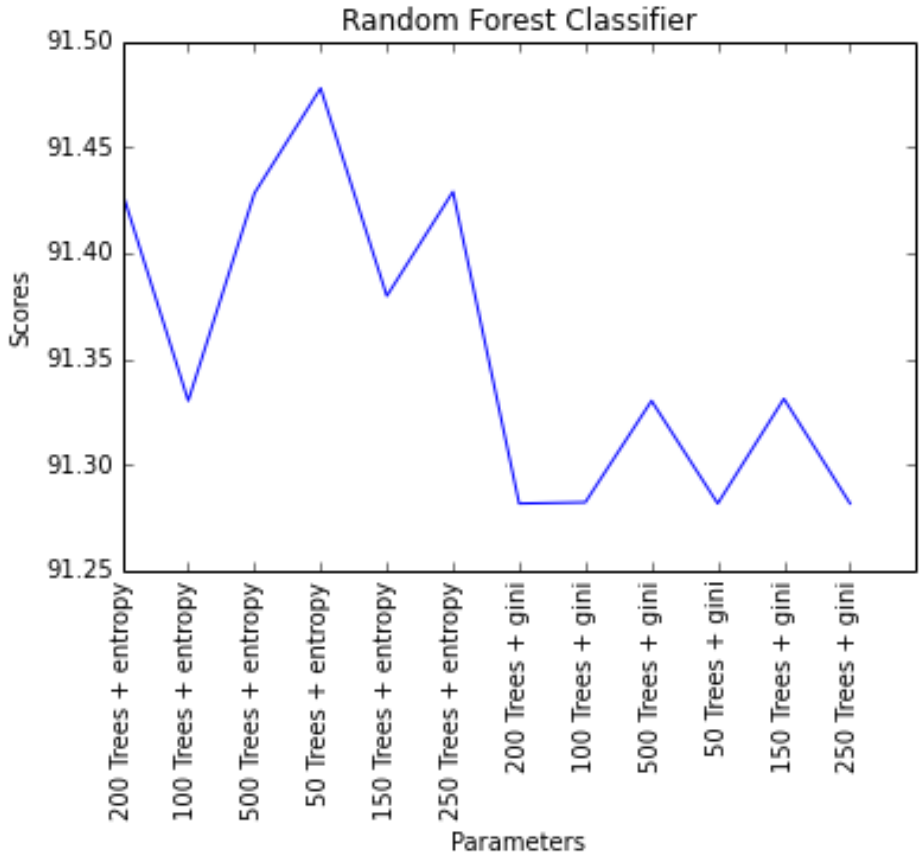
5. Neural Networks
   With Neural Networks we got near about same results as compared to kNN and Random Forests with both TF-IDF and Bag of words as features. We found Tanh activation function, adagrad as the rule and 100 units to give the best results.



(a) kNN



(b) SVM



(c) Random Forest

Figure 1: Classification Models

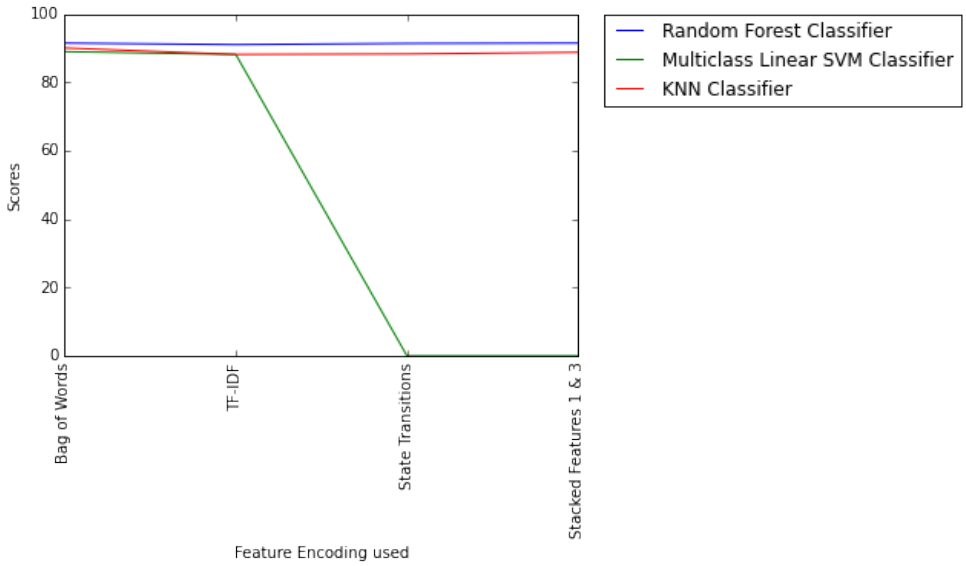


Figure 2: Model Comparison

6. Stacking Features
   We got almost the same results on kNN and Random Forest by stacking the bag of words/TF-IDF with the state transitions as features.

7. Use of Challenge 3 Data
   The use of the classification outputs from Challenge 3 helped the in improving the accuracy by 2-3% across all classifiers.

8. Splitting the training samples in the ratio of 4:1 and using Stratified K-Fold Cross-Validation as a model evaluation metric as depicted in fig. 2, we infer kNN to be the best classifier.

5 Results

The performance is evaluated by computing the Classification Accuracy, i.e. the percentage of correct predictions. The final evaluation results from Kaggle using different models on the dataset are depicted in the graph fig. 3

6 Conclusion

By trying out different models on our training set, best results were observed for the classification by KNN(k=5) along with the application of challenge 3 data on the TF-IDF as a feature with a Public classification accuracy of 81.586% (83.286% Private Score). It had

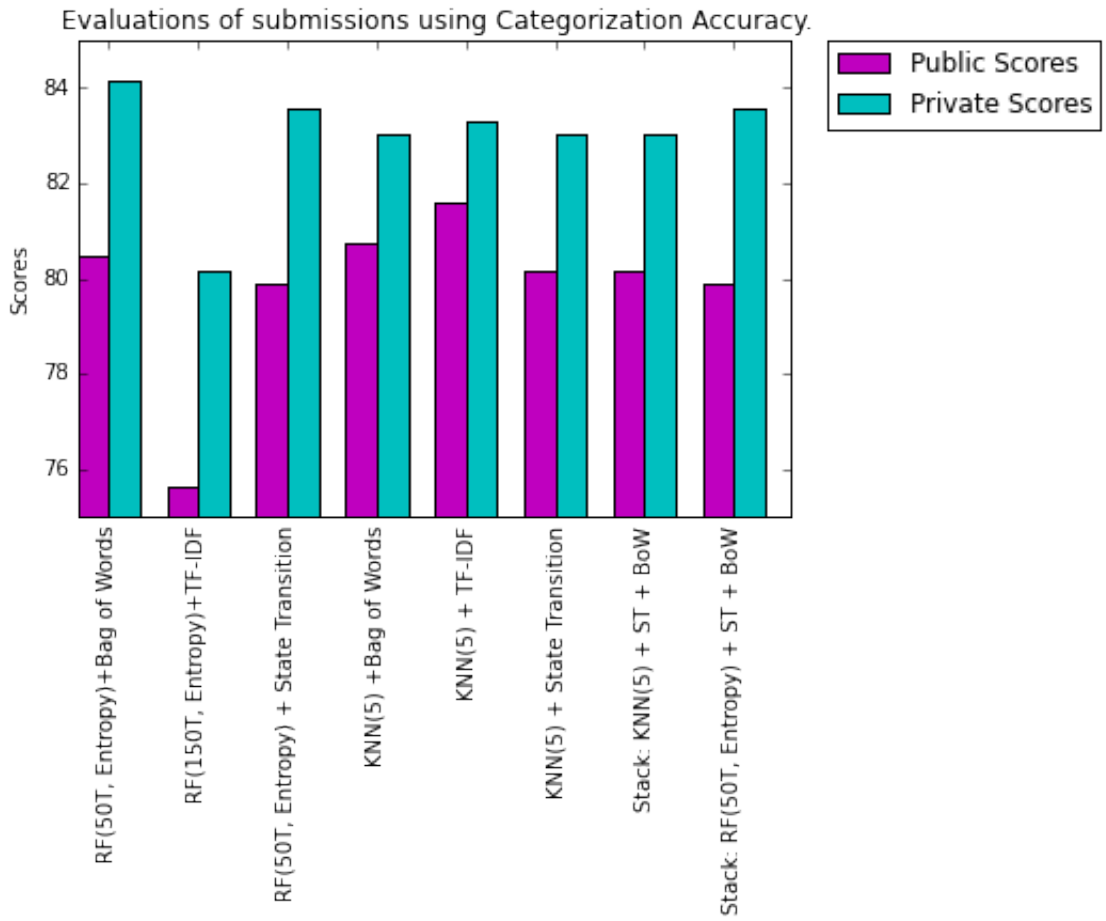


Figure 3: Kaggle Evaluation Results on different models : Public vs Private Scores

a comparable score to Random forests in the cross validation of about 90%. Stacking of features gave almost similar scores.

Random Forest on the bag of words gave the best accuracy of 91.477% on the cross-validation dataset. However it was highly over fitted as verified by its Kaggle score of 80.453% (84.136% Private Score).

The reason why KNN gave better results compared to the other models was that the data contained a lot of repeated patterns which can be expected from the behavior of the malware belonging to the same class. KNN was better at recognizing repeated patterns from other samples.

References

[1] Dietterich, Thomas G. "Machine learning for sequential data: A review." Structural, syntactic, and statistical pattern recognition. Springer Berlin Heidelberg, 2002. 15-30.