Malware Classification Using Weak Classifier Ensemble Learning

Adeel Javed
Michael Hwang
Ryan Tran

Department of Computer Science
San José State University

May 18, 2020

1 Terminology and Approach

In this paper we try to classify multiple families of malware at once using multiple weak classifiers and compare how they perform individually and stacked together using support vector machines. A weak classifier is one that performs very poorly but at least better than random guessing. We chose to use Hidden Markov Models, Random Forest, and K Nearest Neighbors. These classifiers while weak in performance have the advantage of great speed.

We take 53 samples from each of the 7 families, using 80% for training and 20% testing. We start by training a set of 9 Hidden Markov Models for each of 7 families and use Support Vector Machines to apply binary one vs all classification to determine which families perform the best. One vs All classification separates the samples into two categories, in the family or one of the other 6 families. Then we remove models from the stack 2 at a time to see their contribution to the overall performance. We combine all 63 of the models together using Support Vector Machines to classify all families at once.

We then train Random Forest and K Nearest Neighbor models and compare their performance with the Hidden Markov Models. Finally we stack the scores from these two models with the Hidden Markov Models, and assess their accuracy.

2 Applying Hidden Markov Models

We test each of the families for their individual ability to be classified. We train 9 Hidden Markov Models for each family one vs all to judge how consistently each can be classified. These models were trained by looking at the sequence of opcodes in each malware sample. The hyper parameters of these models will follow those in Table 2.1. In our experiments we consider the number of hidden states the model uses, the size of opcode tuples, and how many distinct opcodes should be considered. We grouped the opcodes in each sequence using a sliding window of size n to form n-tuples of opcodes, each of which was mapped to a number that the HMM could process. To prevent processing of noise, we only considered only m distinct opcode tuples, denoted by column 3 of Table 2.1. Each tuple less frequent than the m most common tuples across all samples were mapped to the same number to represent an insignificant opcode in the sequence.

Training a Hidden Markov Model is a hill climb process so it can only find a local minimum. We used 100 random resets on each model trained and chose the best model to be the representative model for its set of hyperparameters.

After training these models we stacked them using Support Vector Machines with polynomial kernels of varying degrees. We compared the degree with the accuracy of the model to determine the best hyperparameter for classifying each family. We found zeroaccess, smarthdd, winwebsec, and zbot to be the best performing families, Fig. 2.1 - 2.4. We considered these to be our strongest families

Number of Hidden states	Tuple Size	Opcode Cut off
2	1	30
2	2	40
2	3	50
2	4	70
3	4	70
4	4	70
2	5	70
3	5	120
6	5	120

Table 2.1: Hidden Markov Model Hyper Parameters

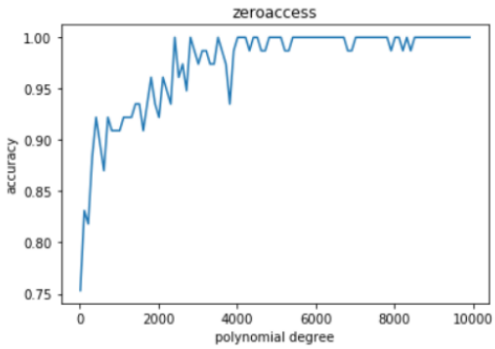

Fig 2.1

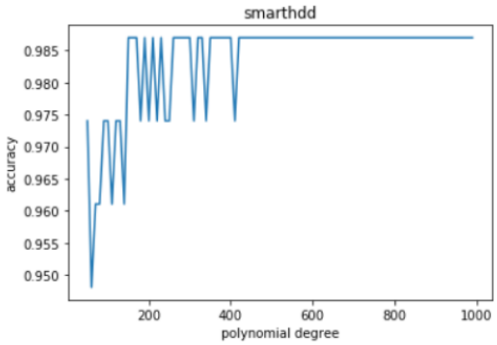

Fig 2.2

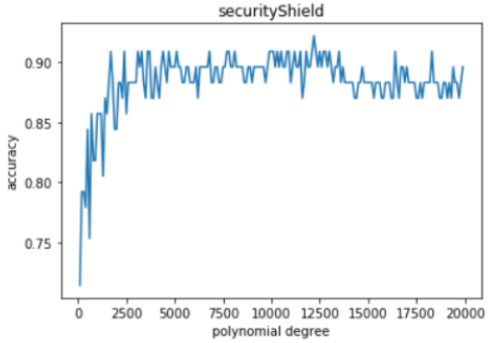

Fig 2.3

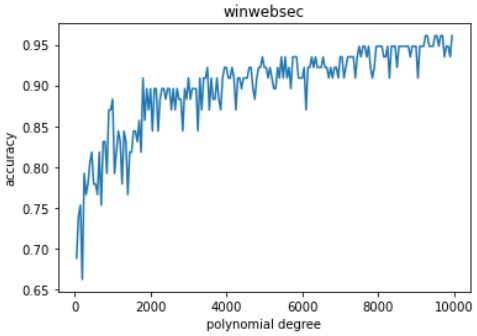

Fig 2.4

and will use them as our baseline when testing on subsets of the families. Harebot, securityshield, and cridex performed much worse, Fig. 2.5 - 2.7, with large variance in the cridex family, and will be added one at a time during testing.

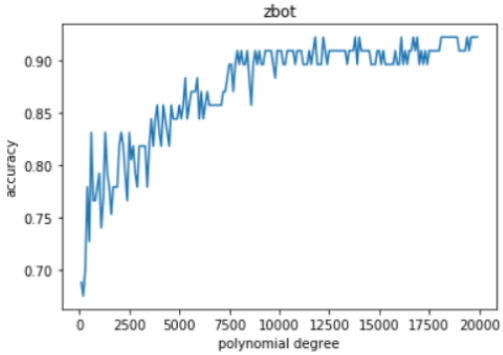

Fig 2.5

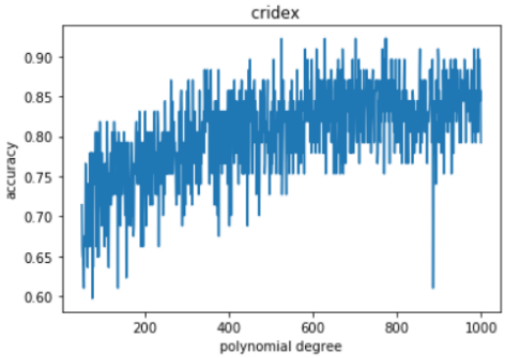

Fig 2.6

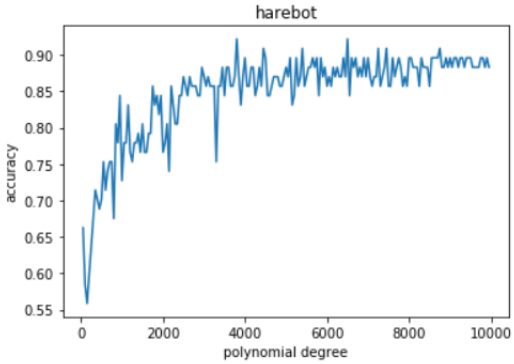

Fig 2.7

3 Classifying Multiple Families

We are start classifying multi-family by choosing the 3 best performing families and stacking each of their 9 models with SVM. In Fig 3.1 we compare the degree of our kernel to the model's overall accuracy. We then add more models and retrain the SVM, following the order of best performing families found in section 2.

Our models obtained 100% accuracy for 3 and 4 family classification, Fig. 3.1 and 3.2. Upon adding our 5th, 6th, and 7th families the accuracy dropped beyond acceptable results, Fig. 3.3 - 3.5. In contrast with binary classification in Fig. 2.1 - 2.7, increasing the degree of the kernel beyond a certain point resulted in worse accuracy. We believe when classifying multiple families, increasing the degree introduces more noise into the model. In general introducing more families into the model decreases the point at which the degree introduces more noise.

To further test our model, trained a model on all families removing pairs of HMMs from the stack from the bottom of Table 2.1 up. In Fig 3.6 - 3.8 test higher polynomial degrees then our original experiments. From theses results we verified that our additional models do not introduce any noise and can be kept for further experimentation. Furthermore, we found while the accuracy of the models drop off quickly, there is a second broad bump in accuracy with much higher degrees.

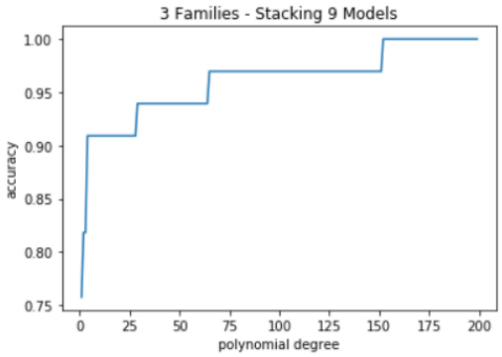

Fig 3.1

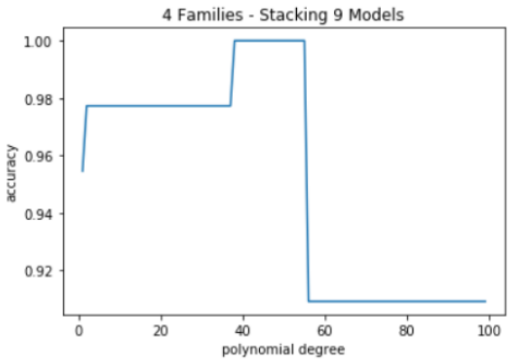

Fig 3.2

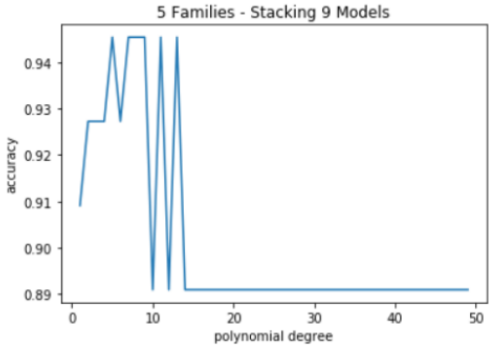

Fig 3.3

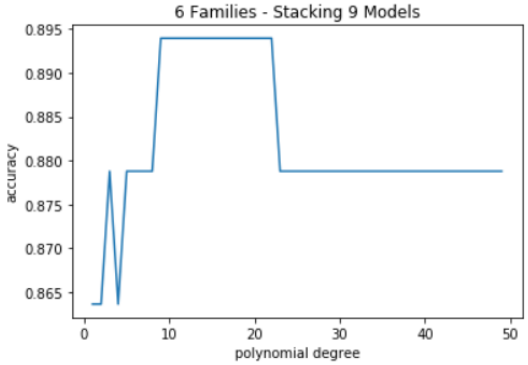

Fig 3.4

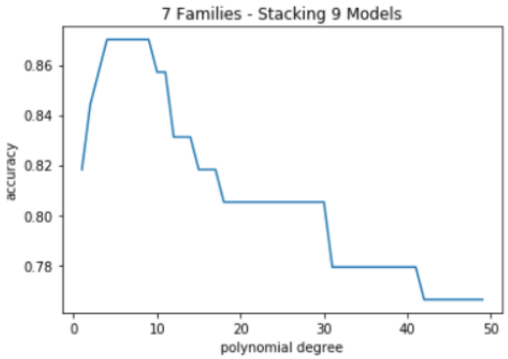

Fig 3.5

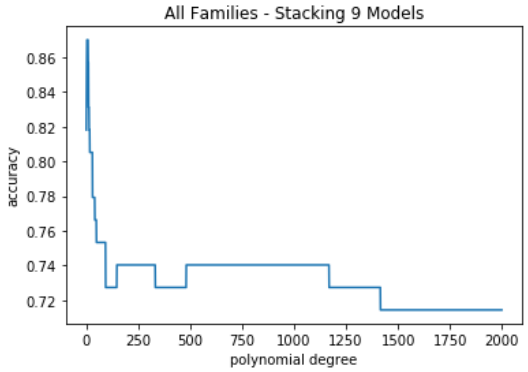

Fig 3.6

This bump is much smaller the more models we stack. However, this second bump is lower than the initial peak, and as such future experimentation will use smaller polynomial degrees.

In general, we found that all 9 HMMs are useful for our overall model and our low polynomial degree is likely to produce the best results moving forward. In order to improve our model, we turn to stacking additional weak classifiers.

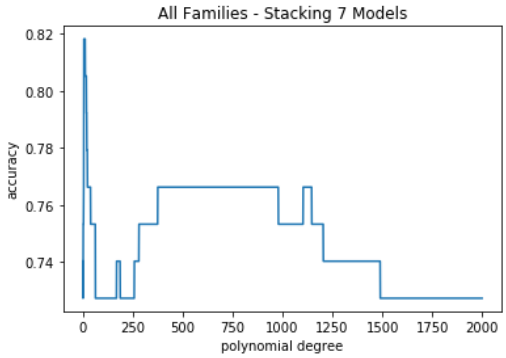

Fig 3.7

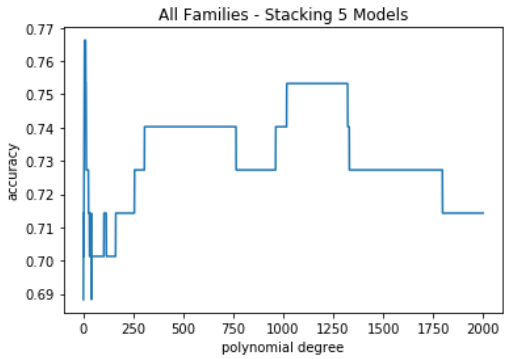

Fig 3.8

4 Applying Addition Classifiers

From all the additional to add to our HMMs, we chose Random Forest and K Nearest Neighbors. The motivation for choosing these models are two fold, These models, while weak in predictive power, are incredibly fast. In addition to their speed, their classification techniques are based on different features than HMMs. As a result, when stacking with SVM, our overall model can combine multiple approaches resulting in better accuracy than either alone.

We processed our samples by looking at 7 features. We first considered the overall file size of each sample. Then we again grouped the opcode sequences into tuples following the procedure in section 2. From these tuple sequences, we looked at the distinct number of opcode tuples.

Finally we applied a modified version of shannon entropy on the opcode sequences. This standard equation processes the file bit by bit. As each bit can be one of 2 states, the base of the log is 2.

$$H(X) = -\sum p_i log_2 p_i$$

We processed our file tuple by tuple and thus modified the base of the equation. However, if we simply change the base to the number of possible tuples of size n, the entropy of should be independent of tuple size thus extracting no new information from higher tuple sizes. Rather than considering all possible tuples of length n, we considered only tuples that exist in at least one of the files already. By modifying the entropy equation, the base of our log varies relative to the tuple size in a way that extracts extra information from our samples.

We fed these 7 features into Random Forest and KNN models starting with classifying 4 families. When training our Random Forest models, we varied the number of estimators to find the optimal model. In general, any additional estimators beyond 25 produced no meaningful improvements beyond random noise.

Just as our HMMs performed perfectly on the 4 best families, Random Forest correctly classified all samples among these families, Fig 4.1. However, when adding additional families the models quickly dropped in accuracy, Fig. 4.2 - 4.4.

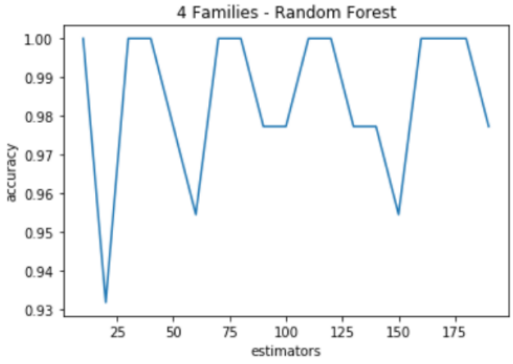

Fig 4.1

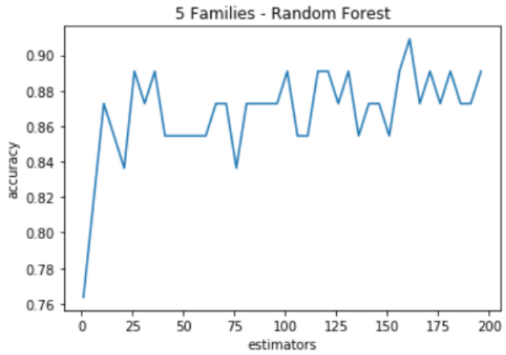

Fig 4.2

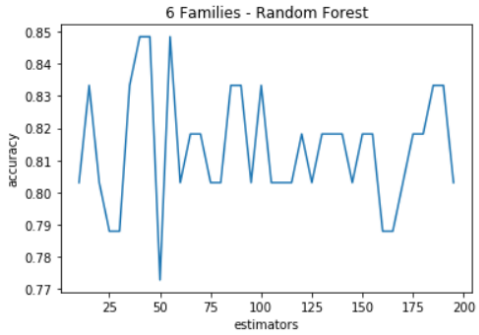

Fig 4.3

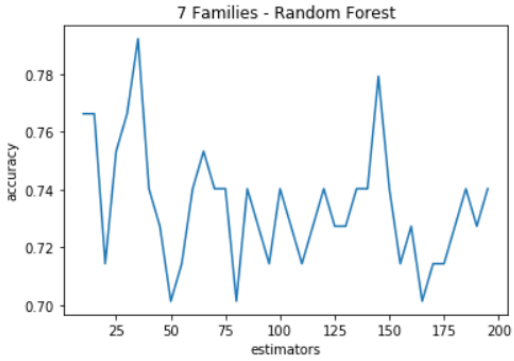

Fig 4.4

The benefit of additional entropy scores calculated for different tuple sizes drops off as well when classifying more families. When removing all but one of the entropy scores, Random Forest only correctly classified 90% of the samples from 4 families, Table 4.1. When removing these features for models trained on 7 families, the difference is at most 4%. Despite the diminishing returns, we kept these features as they did not appear to add any noise that hinders the models.

In contrast with the relatively good results from Random Forest, KNN fails to properly classify even 80% of the samples for even 3 families, Table 4.2. Despite these results we chose to use these models for stacking because they are still better than random chance.

	Number of Families			
Features	4	5	6	7
7	100.00%	85.45%	81.82%	75.32%
6	100.00%	85.45%	78.79%	74.03%
5	97.73%	85.45%	81.82%	74.03%
4	95.45%	83.64%	78.79%	75.32%
3	90.91%	81.82%	78.79%	71.43%

Table 4.1: Random Forest Scores

Number of Families	Accuracy
3	75.76%
4	72.73%
5	69.09%
6	60.61%
7	51.95%

Table 4.2: KNN Scores

5 Stacking All Classifiers

Finally, after training our Hidden Markov Models, Random Forest and K Nearest Neighbors models we compared our overall model's ability to classify 6 and 7 families. When comparing the result with Random Forest and KNN to those without, we found the use of RF and KNN created a 4%-5% boost in accuracy, Fig 5.1 and Fig. 5.2 (see Fig 3.4 and Fig 3.5). This brings 6 family classification within an acceptable range of accuracy. However, our models still fail to classify all 7 families with the desired results.

We found that for both 6 and 7 family classification, harebot and securityShield tend to produce the most false positives, Table 5.1 - 5.4. Our models conversely fail to recognize zbot, cridex, and winwebsec most frequently, confusing them for the other families.

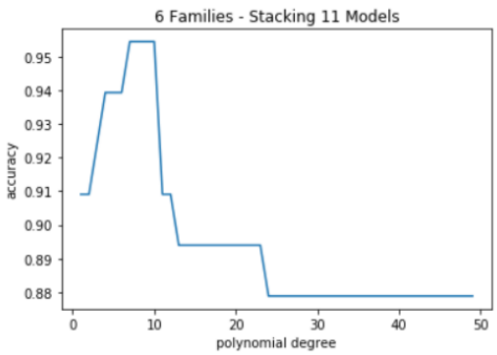

Fig 5.1

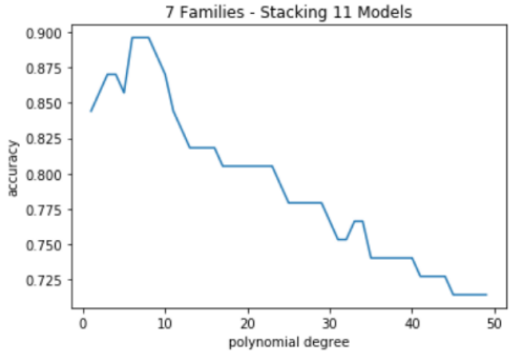

Fig 5.2

Family	harebot	securityShield	smarthdd	zbot	winwebsec	zeroaccess	False Positive
harebot	11	0	0	3	1	0	26.67%
securityShield	0	11	0	1	2	0	21.43%
smarthdd	0	0	11	0	0	0	0%
zbot	0	0	0	7	0	0	0%
winwebsec	0	0	0	0	8	0	0%
zeroaccess	0	0	0	0	0	11	0%
False Negative	0%	0%	0%	36.36%	27.27%	0%	

Table 5.1: HMM Alone - 6 Families Confusion Matrix

Family	harebot	securityShield	smarthdd	zbot	winwebsec	zeroaccess	False Positive
harebot	11	0	0	3	0	0	21.43%
securityShield	0	11	0	1	1	0	15.38%
smarthdd	0	0	11	0	0	0	0%
zbot	0	0	0	7	0	0	0%
winwebsec	0	0	0	0	10	0	0%
zeroaccess	0	0	0	0	0	11	0%
False Negative	0%	0%	0%	36.36%	9.09%	0%	

Table 5.2: HMM with RF and KNN - 6 Families Confusion Matrix

Family	harebot	cridex	securityShield	smarthdd	zbot	winwebsec	zeroaccess	False Positive
harebot	11	3	0	0	0	1	0	26.67%
cridex	0	7	0	0	3	0	0	30.00%
securityShield	0	0	11	0	1	1	0	15.38%
smarthdd	0	0	0	11	0	0	0	0%
zbot	0	0	0	0	7	0	0	0%
winwebsec	0	0	0	0	0	9	0	0%
zeroaccess	0	1	0	0	0	0	11	8.33%
False Negative	0%	36.36%	0%	0%	36.36%	18.18%	0%	

Table 5.3: HMM Alone - 7 Families Confusion Matrix

Family	harebot	cridex	securityShield	smarthdd	zbot	winwebsec	zeroaccess	False Positive
harebot	11	3	0	0	0	0	0	21.43%
cridex	0	8	0	0	3	0	0	27.27%
securityShield	0	0	11	0	1	1	0	15.38%
smarthdd	0	0	0	11	0	0	0	0%
zbot	0	0	0	0	7	0	0	0%
winwebsec	0	0	0	0	0	9	0	0%
zeroaccess	0	0	0	0	0	0	11	0%
False Negative	0%	27.27%	0%	0%	36.36%	9.09%	0%	

Table 5.4: HMM with RF and KNN - 7 Families Confusion Matrix

6 Future Work

Our final results indicate that classify all 7 families requires further research. The accuracy of model could be boosted by adding addition classifier or extracting other features to be used for Random Forest and K Nearest Neighbors.

Malware creators can defeat these models through various means of obfuscation. Further research need to done on using ensemble learning to classify obfuscation techniques.