Malware Classification with Ensemble Learning

Everette Li

May 2020

Abstract

Ensemble methods are vastly used technique in Machine Learning[2]. In order to obtain the best performing model, common ensemble methods like Bagging, Boosting and Stacking are applied in model selection. In this article, a malware classification model will be discussed. This model is a pipeline that combines HMM and SVM together. Experiments are conducted, including ensemble learning methods for HMM, SVM selection and pick of hyper-parameters, to understand and to analyze this pipeline model.

1 Introduction

In order to classify different malware families, a pipeline has been built. This pipeline put HMM and SVM together. The HMM is good at working with sequence, while the SVM is strong in classification. In this study, sequentially represented malware files are being classified. Which is why such HMM-SVM combination are introduced. To enhance the performance of this pipeline model, ensemble learning techniques are used in selecting HMM and SVM. More specifically, random restart for HMM selection and boosting for SVM. Experiments are then performed to analyze this pipeline method.

The section "Methodology and Dataset" goes through the pre-processing on the raw malware data. The section "Experimental Setup" introduces how the experiments are designed and why. The "Experimental Results" shows and discuss the outcomes. Conclusion and future work will be mentioned in the end.

2 Model Design

The goal of this pipeline model is clear and simple - classify malware file. That is, given a malware file, label the file with the correct malware family. In this study, malware from 3 families are given - *Zbot*, *Winwebsec* and *Zero Access*. The model have to know which family of the three a file belongs to. Serving this purpose, a pipeline model has been designed. The pipeline model connect HMM and SVM together. The HMM will score the file then the SVM will classify it based on its HMM score.

Shown in the Figure1, the pipeline model for malware classification is composed with 2 layers. The first layer uses HMM. The HMM layer transform malware file from opcode text into numerical data by applying the scoring function. Each malware file will be scored by all three HMMs to obtain a vector.

$$< H_{zbot}, H_{win}, H_{zero} >$$

The second layer uses SVM. Three one-vs-rest Support Vector Classifiers are combined into one multi-class classifier, which gives the final label.

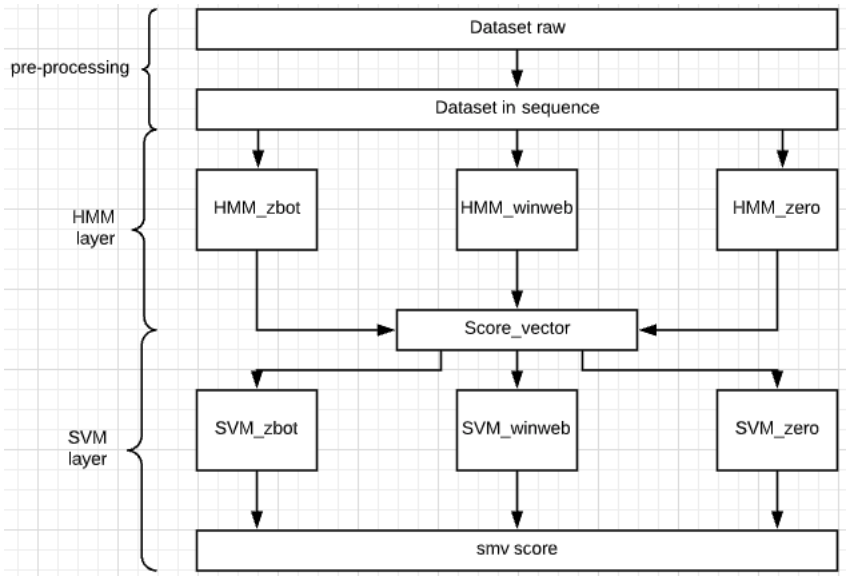

Figure 1: Pipeline Model Design

2.1 HMM Model

Three HMM models are used in this design, one for each family. Each HMM will be trained on FOC(family observation sequence) and become specialist in detecting that family. In this way, when a malware file, say zbot file, is scored by all three HMMs, the Zbot HMM will give it a high score whereas the other two are not. This can help the SVM in the next step.

2.2 SVM Model

This study use SVM from scikit learn. According to scikit learn, if the SVM object is declared for multi-class classification, the scikit learn library will implement one-vs-rest SVM for each class under the hood while, appearing to user, have only one SVM object showing in code. This feature make the implementation of SVM layer much easier.

3 Methodology and Dataset

The HMM and the SVM consumes different kind of data. Therefore, the pre-processing of this pipeline model have to address two parts. One for HMM and the other for SVM.

3.1 Pre-processing for HMM

The initial malware file contains a list of opcodes. These opcodes are the assembly code used by the malware program. To extract the feature out of malware opcode file, the token-frequency of each opcode is considered. The following steps are used to process raw opcode text file. After the last step each opcode file will be represented by 5 opcodes(5 is the best pick from last study[1]) that has the most token-frequency and the family observation sequence will be made.

3.1.1 Step 1: Get token-frequency

Let *length* be the length of a file, *count* be the number of time this opcode appears in this file. Then the opcode token-frequency, TF_{opcode}, is as below.

$$TF_{opcode} = \frac{count}{length}$$

3.1.2 Step 2: Five opcodes for each file

After computing TF_{opcode} for all opcodes, sort the frequency and pick top 5. These five opcodes are the ones that appears the most in this file and will be use to represent the file.

3.1.3 Step 3: Convert all the files

Repeat the following to all the files. Now a file is list of five opcodes. And the malware family is a long list of such file list.

3.1.4 Step 4: Make dataset and family observation sequence

FOS(family observation sequence) is a long list of opcodes resulting from file concatenation. FOS of a malware family will be used to train the HMM of that family. Files in the FOS will not be include in dataset.

$$FOS = file_1[op_1, op_2...op_5] :: file_2[op_1...op_5]... :: file_n[op_1...op_5]$$

The dataset is a set of file-target pair as below.

$$Dataset\{([op_1...op_5], family_1), ([op_1...op_5], family_2), ...\}$$

3.2 Pre-processing for SVM

The "raw data" for SVM are the scoring vectors from HMM layer. They're almost clean enough. However, due the design of HMM model there will be some NaN or $-Infinity$ in the scoring. Some opcode only appear in one family but not others, this make 0 in transaction matrix, the B matrix. Therefore, when computing the score for some file, 0 will be hit and lead to $\frac{1}{0}$ in the program. This 0 in B matrix can be interpret as "it is impossible for this observation sequence to be in this family", since it contains some opcode that never even shows in this family. Based on this assumption, all the NaN and $-Infinity$ will be replaced by 0.

4 Experimental Setup

The experiments are design based on layer - HMM related experiments and SVM related ones. For HMM layer, the performance is matured by observation sequence scoring, the higher the better. For the SVM layer, the performance is matured by ROC curve and AUC score. All the results can be find in the Experiment Results section accordingly.

4.1 HMM Experiment

There are many hyper-parameters that can be tuned in HMM in general. This study is focusing on the Table1. For 1-3, control variable experiment will be done, then random restart will be applied based on the

1. number of state	number of hidden states
2. FOC length	length of family observation sequence
3. iteration	number of iterations in training
4. number of restart	restart times in random restart

Table 1: Tuning parameters

best 1-3 parameter to see if future optimization is possible.

4.1.1 HMM: number of state

The experiment set up follows this Table2 The result of this experiment can be found in Table10

number of state	FOC	iteration	number of restart
2	2500	50	0
5	2500	50	0
10	2500	50	0
50	2500	50	0
100	2500	50	0

Table 2: number of state: HMM initial setup

4.1.2 HMM: FOC length

The experiment set up follows this Table3 The result of this experiment can be found in Table11

number of state	FOC	iteration	number of restart
50	2500	50	0
50	1500	50	0
50	500	50	0
50	100	50	0
50	50	50	0

Table 3: FOC length: HMM initial setup

4.1.3 HMM: Iteration

The experiment set up follows this Table4 The result of this experiment can be found in Table12

4.1.4 HMM: Random Restart

The experiment set up follows this Table5 The result of this experiment can be found in Table13

number of state	FOC	iteration	number of restart
50	2500	10	0
50	2500	50	0
50	2500	100	0
50	2500	200	0
50	2500	300	0

Table 4: Iteration: HMM initial setup

number of state	FOC	iteration	number of restart
50	2500	50	0
50	2500	50	5
50	2500	50	10
50	2500	50	20
50	2500	50	30

Table 5: Random Restart: HMM initial setup

4.1.5 HMM: Random Restart

4.2 SVM Experiment

The SVMs are constructed based on the HMM score vector. Therefore, before starting the experiment on SVM, we should take a look at the score vector from HMM. The score vector dataset has attributes as Table6

length	training set length	test set length
2400	1920	480

Table 6: Score Vector Set

4.2.1 Understanding Score Vector

Though SVM can splits the data in high dimension, it is interesting to visualized the scoring vector and see if any pattern exist. According to Figure2 and Figure3, one class is clearly separable from the other two. For the orange and the blue class, though they are very close to each other, they are not overlapping with one another.

For SVM, the experiment will test on different kernel functions and compare the results with ROC curve and AUC score.

4.2.2 SVM: RBF Kernel

RBF(Radio Basis Function) function, or Gaussian function is a bell curve. The closer the point to the center of the bell the higher confidence the classification is. In Python Sklearn library, the formula of RBF Kernel is given as follow.

$$exp(-\gamma(\|x - x'\|^2))$$

Figure7 is the initial setup for SVM with RBF Kernel. Here the C penalty varies, the higher it is the stricter the boundary is.

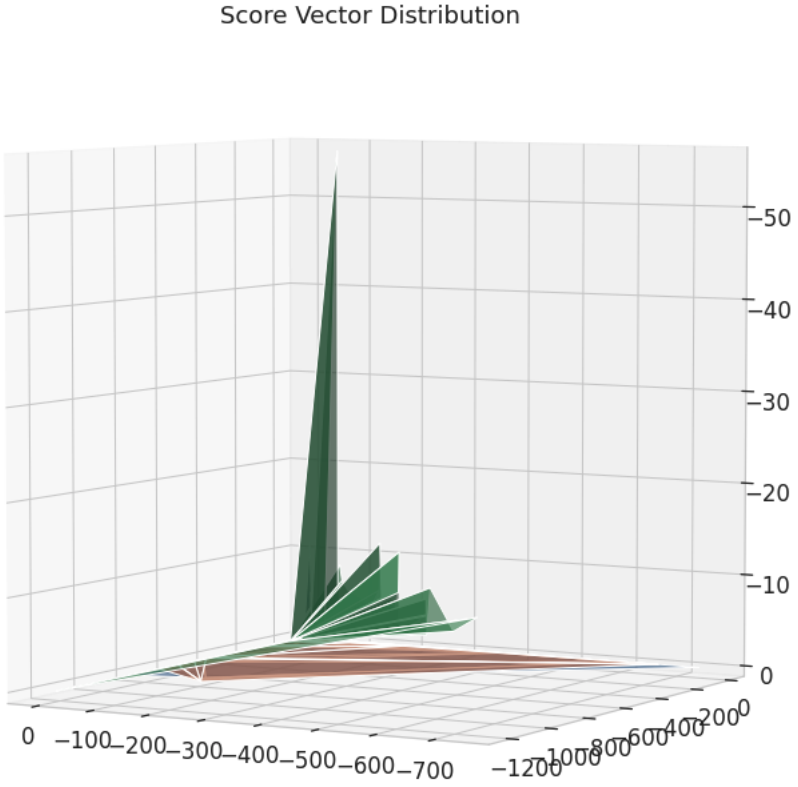

Figure 2: 3D Score Vector 1

C penalty	Kernel	gamma
0.5	RBF	0.00052
1	RBF	0.00052
1.5	RBF	0.00052

Table 7: SVM RBF kernel

4.2.3 SVM: Polynomial Kernel

Polynomial Kernel enable SVM to draw complicated boundary around each clusters. The initial setup is given in Table8 The input data has been scaled for Polynomial Kernel since the un-scaled version runs

C penalty	Kernel	gamma
0.5	Poly	0.1
1	Poly	0.1
1.5	Poly	0.1

Table 8: SVM Polynomial kernel

forever.

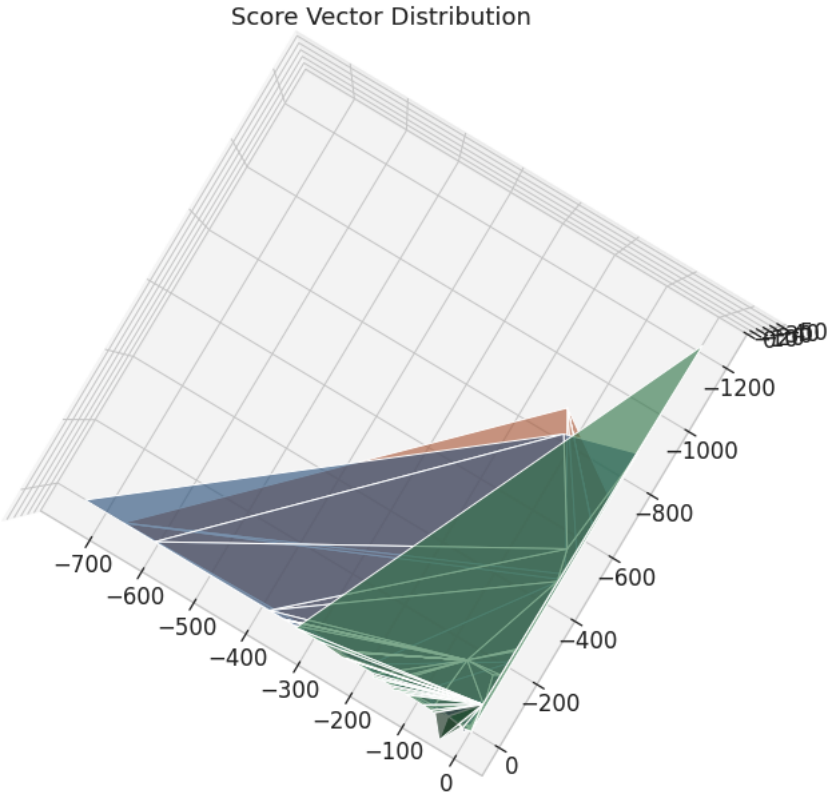

Figure 3: 3D Score Vector 2

4.2.4 SVM: Linear Kernel

Linear Kernel is the most basic kernel option for SVM. The initial setup is as the Table9 Table8

C penalty	Kernel	gamma
0.5	Linear	0.1
1	Linear	0.1
1.5	Linear	0.1

Table 9: SVM Linear kernel

5 Experimental Results

This section contains the results from the experiments and the discussion on them.

5.1 HMM Results

5.1.1 Result - HMM: number of state

Table10 is the resulting table. As Table10 shows a great amount of increase can be observed. The observation

number of state	2	5	10	50	100
Zbot	-5018.869	-2812.086	-2449.828	-1843.90	-1632.624
Winwebsec	-4825.609	-2435.885	-2334.885	-1272.208	-1243.585
Zero Access	-4477.955	-3722.264	-1994.247	-1376.274	-1344.455

Table 10: number of state: HMM result

sequence score is monotonously increasing as the number of number of hidden state increase. Shown in Figure4 is the curve of observation score. According to Figure4, a sharp increase in score can be seen from

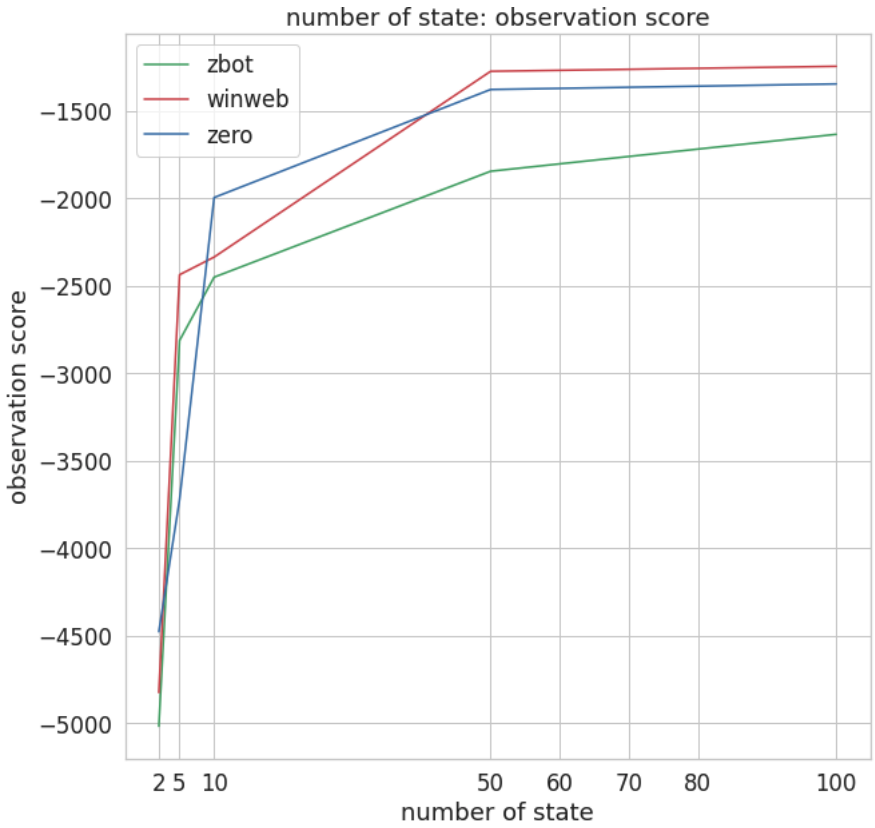

Figure 4: number of state: HMM score

2 to 5. Then this increasing trend become less aggressive with it pass 50. From the result 50 hidden state is a good number of hidden states.

5.1.2 Result - HMM: FOC length

Table11 is the resulting table. The result is quite against intuition. At this point it is hard to say if this an actual increase in performance or a result of over-fitting. It can also simply because shorter sequence is easier for HMM to "study". Figure5 is the plot of observation score. From the Figure5 a very unnatural

FOC length	2500	1500	500	100	50
Zbot	-1857.230	-1131.471	-305.099	-41.422	-11.090
Winwebsec	-1272.021	-743.678	-248.214	-39.919	-16.997
Zero Access	-1371.159	-830.911	-255.950	-33.335	-14.222

Table 11: FOC length: HMM result

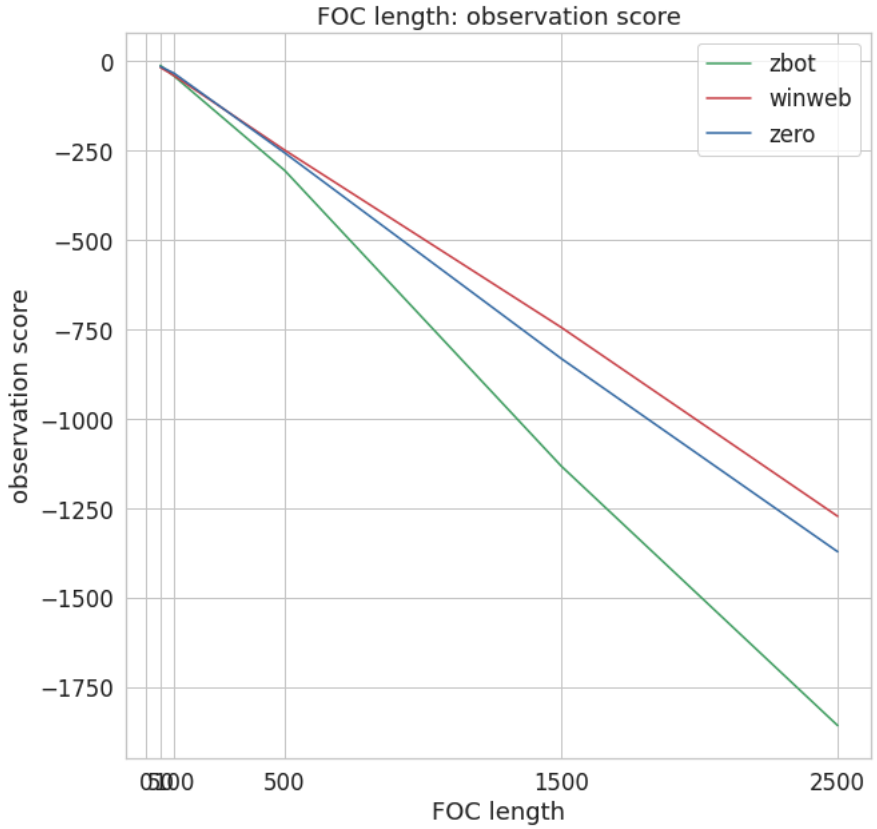

Figure 5: FOC length: HMM score

decrease can be observed. Which may lead to the conclusion that the length of observation sequence alone cannot determine whether the overall performance is increased or not.

To confirm this assumption, the scouring vector is check. As a result, when the FOC is less than 1500, there will be too many 0 in the scoring vector and cannot be use for the SVM step. This may because the sequence is too short for 50 hidden states.

5.1.3 Result - HMM: Iteration

Table12 is the resulting table. Figure6 is the plot of observation score. From the Figure6, an obvious increase can be observed at 50 iterations. The increasing rate then drop after 50. Meanwhile, the training time of the model is vastly increased. As a result, 50 iterations is a reasonable pick.

Iteration	10	50	100	200	300
Zbot	-4928.383	-1867.758	-1644.015	-1595.407	-1580.565
Winwebsec	-4793.657	-1269.744	-1258.228	-1243.544	-1240.597
Zero Access	-4774.461	-1365.463	-1345.802	-1333.611	-1337.593

Table 12: Iteration: HMM result

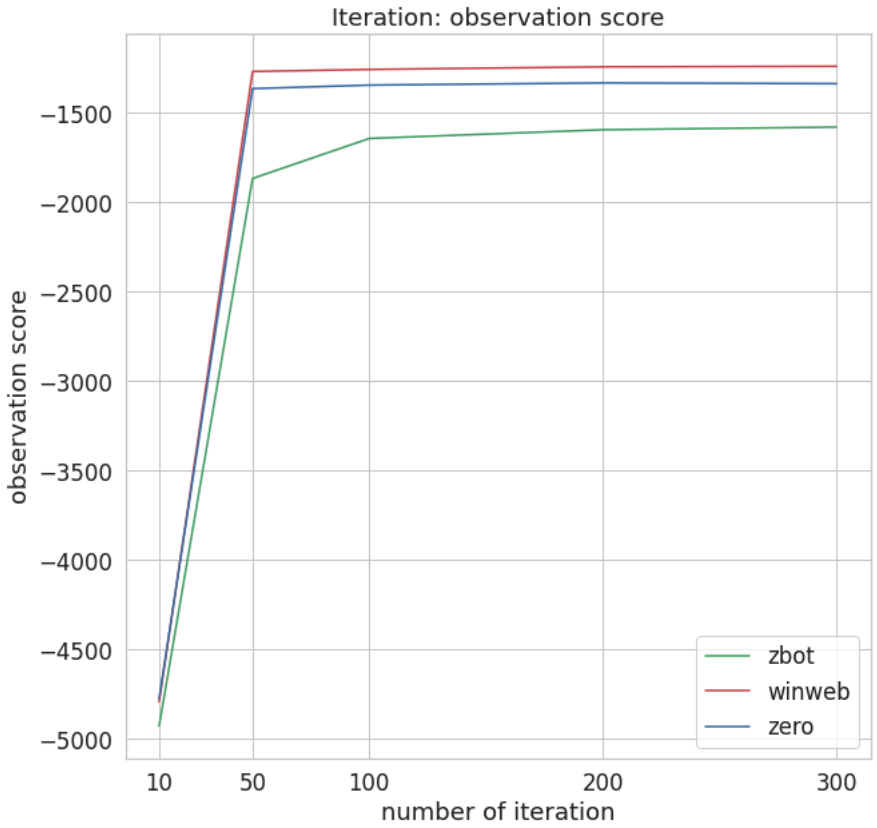

Figure 6: Iteration: HMM score

5.1.4 Result - HMM: Random Restart

Table13 is the resulting table. Figure7 is the plot of observation score. This is another anti-intuition result.

number of Random Restart	0	3	5	8	10
Zbot	-1752.074	-1876.282	-1890.358	-1659.969	-1856.070
Winwebsec	-1296.288	-1284.652	-1282.698	-1294.726	-1286.232
Zero Access	-1369.007	-1353.620	-1367.059	-1351.493	-1350.222

Table 13: Random Restart: HMM result

Random Restart doesn't boost the result that much. This can lead to two conclusions.

First, the discrete hill clime reach the peak already under this HMM setup. That is after 50 iterations on 50 hidden states, no matter where to start from, this score is the best this HMM can get. This explain why the curve is flat in Figure7.

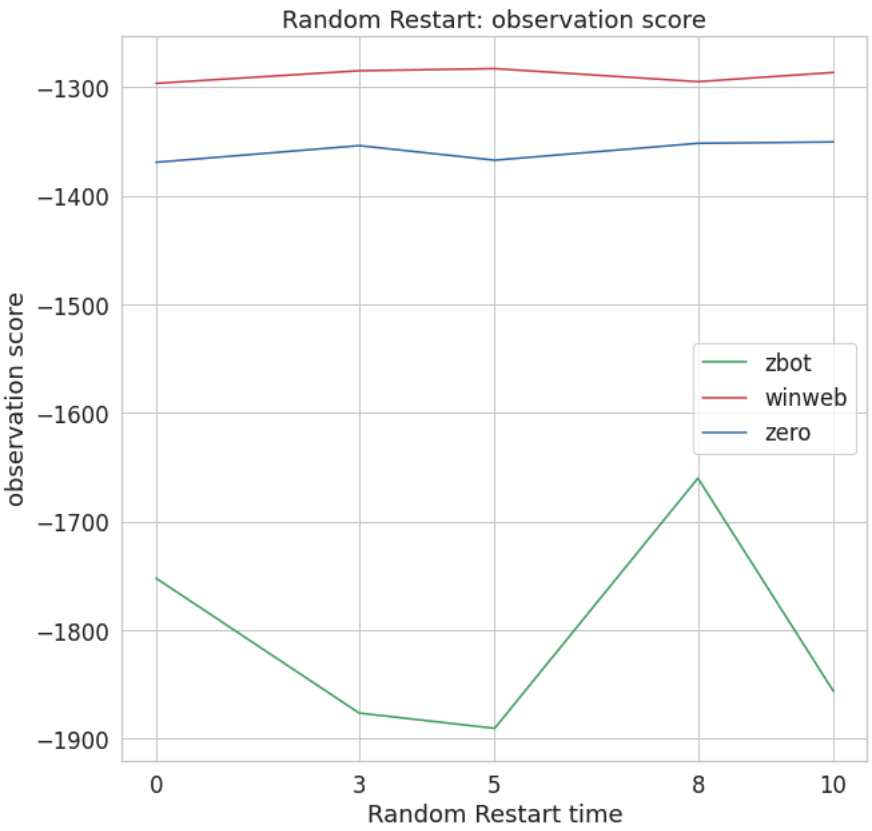

Figure 7: Random Restart: HMM score

Second, the random restart is not random enough. Right now the interval of the random restart is

$$average \pm 0.01$$

This interval maybe too small for it to be random. To prove this assumption, $average \pm 0.05$ and $average \pm 0.02$ are being tested with random restart times set to 3. From Table14 we can tell that with interval set to

initial interval	±0.01	±0.02	±0.05
Zbot	-1876.282	-1713.986	NaN
Winwebsec	-1284.652	-1269.187	NaN
Zero Access	-1353.620	-1337.588	NaN

Table 14: Random Restart interval: HMM result

$\pm 0.02 the score is slightly increased.$

As a result, the final HMM model is settled with the following configuration. The following SVM

number of state	FOC	iteration	number of restart	random interval
50	2500	50	3	±0.02

Table 15: HMM final setup

experiments will uses the HMM setup from Table15

5.2 SVM Results

5.2.1 Result - SVM: RBF Kernel

C penalty	SVM score	average cross validation	max cross validation
0.5	%95.4	%96.3	%98.9
1	%95.4	%96.3	%98.9
1.5	%95.4	%96.3	%98.9

Table 16: Result: RBF Kernel

As Table16 shows, the result shows no difference in these 3 picks. The $C = 0.5$ model is picked to check the ROC curve plotting and AUC score. As the Figure8 is showing, the classification is nicely done.

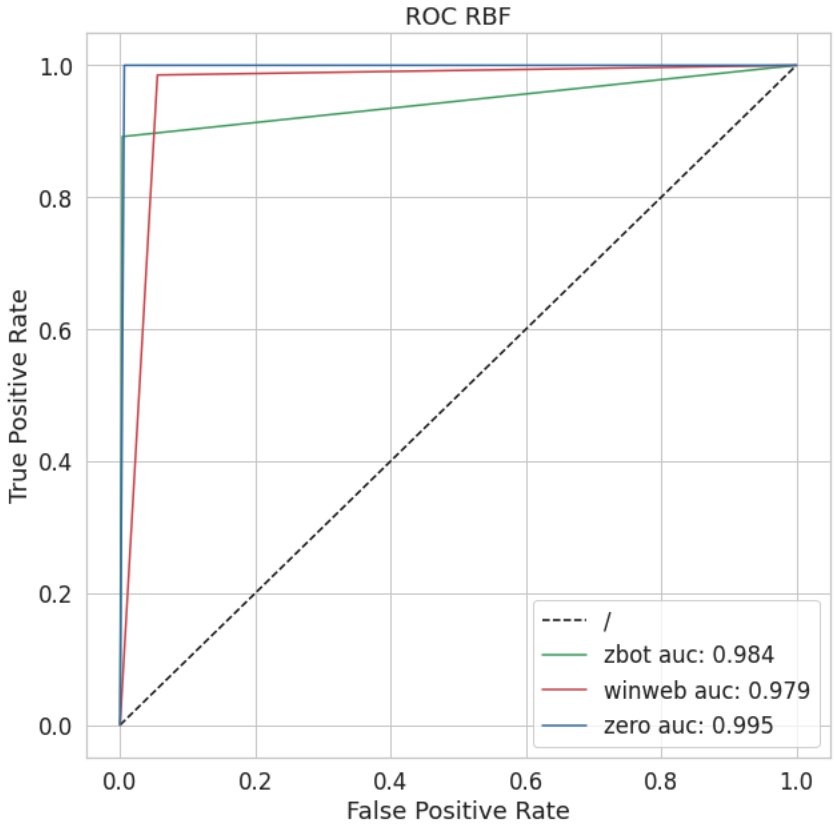

Figure 8: RBF: ROC & AUC

5.2.2 Result - SVM: Polynomial Kernel

As the Table17 is showing, the results from Polynomial Kernel is less impressive than the RBF Kernel. Figure9 also shows that Polynomial Kernel is less effective in classifying the score vectors.

C penalty	SVM score	average cross validation	max cross validation
0.5	%83.3	%78.7	%86.4
1	%87.5	%78.7	%86.4
1.5	%87.5	%85	%91.6

Table 17: Result: Polynomial Kernel

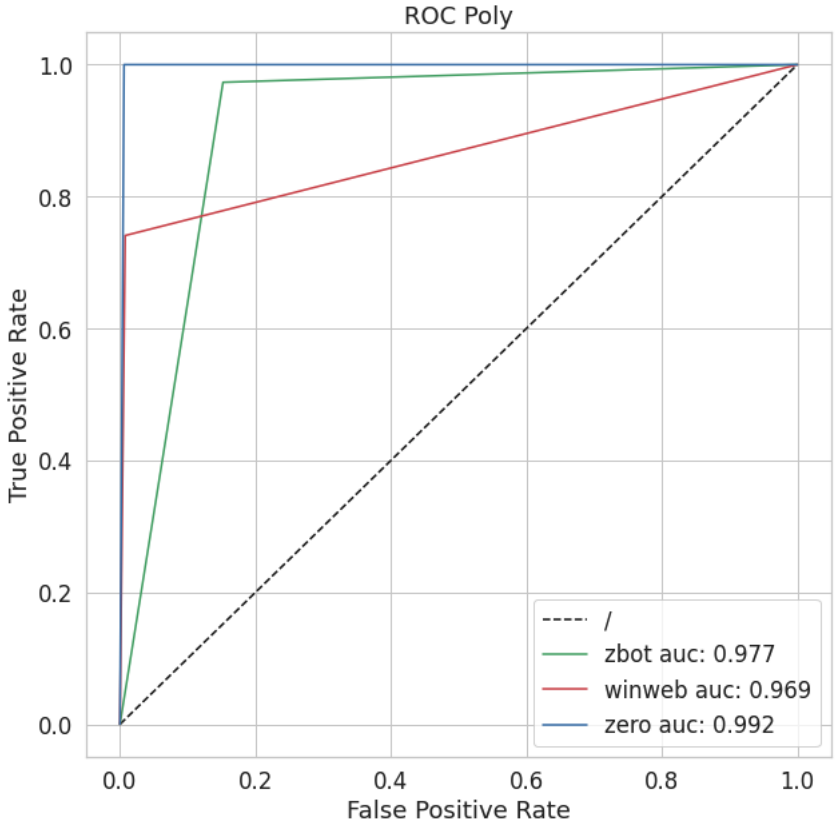

Figure 9: Polynomial: ROC & AUC

5.2.3 Result - SVM: Linear Kernel

As the Table18 shows, the results for Linear Kernel are worse then Polynomial. The Figure10 is the ROC curve for Linear Kernel.

6 Conclusions and Future Work

To Conclude, This article talks about a HMM-SVM pipeline model that is arming to classify three malware families. HMM and SVM are analyzed separately. For HMM, 4 hyper-parameters, number of states, FOC length, iteration and random restart times, are being tested in a control variable manner. As a result, number of states and random restart times with good pick of random interval are the two factors that influence the HMM models the most. For SVM, 3 kernels are being tested classifying the HMM score vectors. Among all 3, RBF kernel is the best fit for this dataset. The highest final score for this pipeline classification model is %98.9.

C penalty	SVM score	average cross validation	max cross validation
0.5	%79.1	%78.6	%82.2
1	%79.1	%78.6	%82.2
1.5	%79.1	%78.6	%82.2

Table 18: Result: Linear Kernel

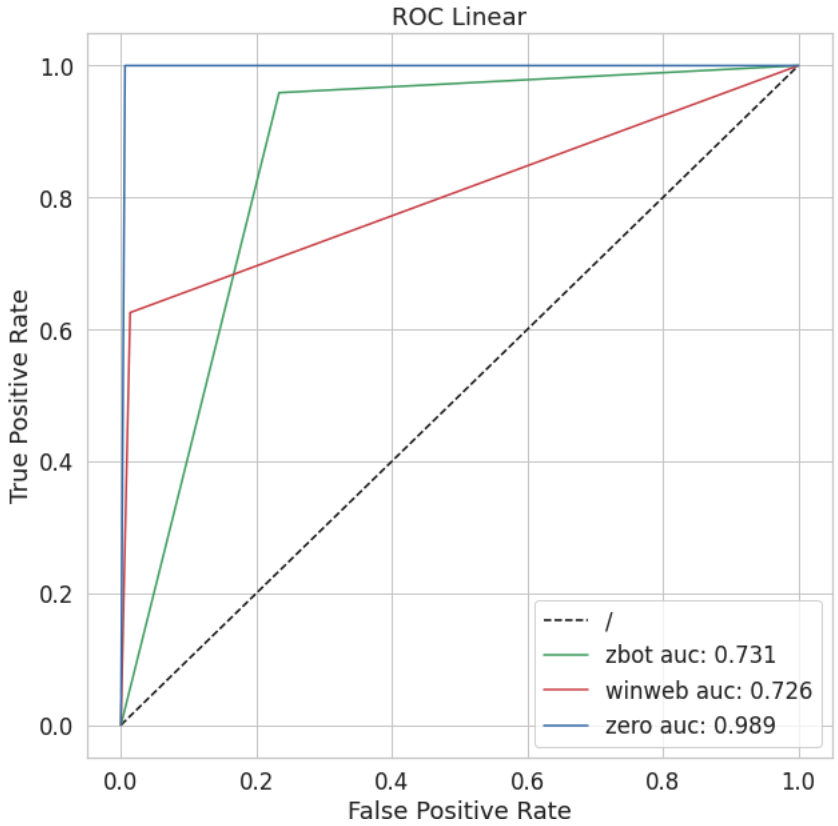

Figure 10: Linear: ROC & AUC

There are still many unexplored experiments. Especially in the second layer of the pipeline. K nearest neighbor, random forest and K-Mean can all be the alternative choice of SVM. In fact, SVM is not the best choice for multi-class classification. If more time is given, more ensemble learning techniques can be applied for model selection too. I'm always looking forward to further exploration.

7 Reference

References

[1] Everette Li. "Zero-day Detection: Classification Experiments on Malware with Small Sample Size". In: *CS 185C midterm* (2020).

[2] Aditya Raghavan, Fabio Di Troia, and Mark Stamp. "Hidden Markov models with random restarts versus boosting for malware detection". In: *Journal of Computer Virology and Hacking Techniques* (2018). DOI: 10.1007/s11416-018-0322-1.