

# **Comparison of Malware Detection Technique of HMM, PCA, and MLP**

## **Abstract**

In this project, we used different machine learning techniques to train on some Malware files to gain models. Then, we used the models to score on the Malware files and benign files to gain some scores. Using these scores, we then did some analysis and comparisons. Specifically, we used two features of Malware files. One is sequence of the Opcode, another is the frequency of the registers. In the HMM process, we used Opcode to train and score. In then PCA process, we trained and scored on the frequencies of the registers. Then, we calculate the True Positive Rate (TPR), False Positive Rate (FPR) to obtain ROC curve and PR curve. Basing on the curves, we found the thresholds and train the multilayer perceptron to classify the malware and benign files on the HMM scores and PCA scores. At the end, we concluded that PCA technique with frequencies of the registers had a effective way to detect Malware while HMM didn't show an ideal accuracy on the detection. In addition, with an accurate threshold, the MLP can yield a better classification.

## **1. Introduction**

Signature detection to malware is simplest and lowest-cost technology. When it detect on some simple malware, it yields a low FPR. However, the viruses we might experience now are no more simple. They already evolved to some metamorphic viruses. At this point, signature detection has no longer advantages. Meanwhile, some machine learning techniques will yield a better result than signature because those techniques can make computer become smarter. Basing on the data we offer, those techniques can train a model to identify the other data belonging to the same or close kind of data that we offered. Informally, the machines can learn the important information from the Malware features and use that information to help us classify the Malware and benign files.

## 2. Methods

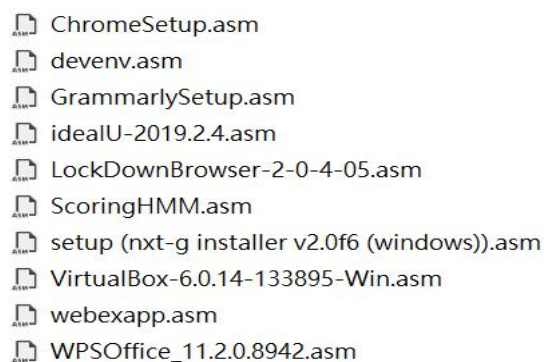
### 2.1 Cross Validation

We used 5 cross validation to train and score the data sets. First we asked Professor Fabio to get the Malware files. We choose 5 families and 5 files for each Family. We also downloaded 10 benign files as our test set. In each fold, we chose 20 Malware files from those 5 families to train a model, then used the model to score on the rest 5 Malware files and those 10 benign files. That is, in each fold, we obtained 5 Malware scores and 10 benign scores. After 5 folds, we got 25 Malware scores and 50 benign scores. Then, we calculate the average scores with the 50 benign scores since they were repeated.

For every Malware files, we released it in virtual box, and extracted the ASM file. Then using program to read the Opcode sequence and count the register frequency. For convenience, we rename all the Malware files as “family name+ number”.

cleaman1	securityshield1	winwebsec1	zbot1	zeroaccess1
cleaman2	securityshield2	winwebsec2	zbot2	zeroaccess2
cleaman3	securityshield3	winwebsec3	zbot3	zeroaccess3
cleaman4	securityshield4	winwebsec4	zbot4	zeroaccess4
cleaman5	securityshield5	winwebsec5	zbot5	zeroaccess5

**Figure 1 : name of Malware files**



ChromeSetup.asm  
devenv.asm  
GrammarlySetup.asm  
idealU-2019.2.4.asm  
LockDownBrowser-2-0-4-05.asm  
ScoringHMM.asm  
setup (nxt-g installer v2.0f6 (windows)).asm  
VirtualBox-6.0.14-133895-Win.asm  
webexapp.asm  
WPSOffice\_11.2.0.8942.asm

**Figure 2: name of benign files**

### 2.2 HMM with Opcode Sequence

In HMM, we used forward and backward algorithm to get the probability at time  $t$  while the observation is  $O_t$  and the state is  $q_t$ . We also used Baum-Welch algorithm to

re-estimate the model. In our project, we use 200 iterations for each re-estimate. And we used 2 hidden states, 54 observations and 100 random restart. Then, we use forward algorithm to score on the test set. The normalizing Log probability represented the score.

### 2.3 PCA with Register Frequency

In PCA, we used 16 registers frequency as our measurements/features, used 20 Malware files as our experiments. So, we constructed a 16\*20 matrix called B matrix. Then, we subtract the mean for each row to get A matrix. We used inner product to produce the covariance matrix and calculate the eigenvalue and eigenvector space called U. Then, we discarded the less significant eigenvector with small eigenvalue. We projected the A matrix onto U to gain scored matrix. Finally, we projected the test vector on to U and calculate the Euclidean Distance between the result and each column of the score matrix. The smallest distance was the preferred score.

### 2.4 Analysis with TPR, FPR, ROC and PR

In this process, our main goal was determining the threshold where we can classify Malware and benign with better accuracy and sensitivity. Therefore, we made the ROC and PR graph on both HMM and PCA scores.

### 2.5 MLP with HMM and PCA Scores

We then use threshold we just determined from last step to choose samples as the MLP training samples. Then we scored on all the samples to to classify them as 1 or 0. We use these classification to calculate the accuracy.

In this process, we used  $f(s,t) = \frac{1}{1 + e^{-(s+t)}}$  and  $g(s,t) = s + t$  to construct the layers.

We also tried single layer with  $f(s,t) = \frac{1}{1 + e^{-(s+t)}}$  to make a single layer perceptron.

## 3. Result

### 3.1 HMM result and Analysis

After we trained on the 1<sup>st</sup> fold, it yielded a  $\pi$ , A, and B matrix as figures 3. We did see some Opcodes have near 0 probability on the side of B matrix, such as js, jns, jl, rep, and so on. That means these Opcodes barely show up in one of those two states.

```
best model with prolog -93166 is model 18
The pi, A and B matrix of best model:
 $\pi = \begin{pmatrix} 1 & 0 \end{pmatrix}$ 
A =
(0.866655  0.133345 )
(0.0490685  0.950932 )
```

```
B =
jz      : ( 0.00521027  0.0127871 ) shr      : (1.42834e-105  0.0153761 )
jnz     : ( 0.00745044  0.0128817 ) sbb      : ( 0.00373602  0.0175713 )
jb      : ( 0.00113982  0.0026911 ) sub      : ( 0.0196318  0.0454428 )
jnb     : ( 0.00062026  0.00472038 ) xor      : ( 0.020391  0.0964526 )
js      : ( 0 0.000282779 ) pusha    : ( 0.000945583  4.08315e-05 )
jns     : ( 0.000164301  8.09242e-05 ) dec      : ( 0.00223115  0.00465775 )
ja      : ( 0.001456  0.00275148 ) and      : ( 0.00670232  0.0288866 )
jg      : ( 0.0035184  0.000613931 ) fidiv   : ( 9.60485e-05  2.32422e-136 )
jge     : ( 0.00311292  0.000197598 ) not     : ( 0.000385215  0.00611473 )
jbe     : ( 0.00576881  0.000704778 ) neg     : ( 0.000524908  0.00277601 )
jl      : ( 0.0031696  5.62043e-71 ) imul    : ( 8.46686e-05  0.00834618 )
jle     : ( 0.00363948  0.000569372 ) movsb   : ( 0.000480243  2.36399e-263 )
jnp     : ( 9.60485e-05  1.61353e-206 ) loop    : ( 7.23816e-155  7.06949e-05 )
jmp     : ( 0.0215756  0.0403798 ) std      : ( 0.000105314  3.19374e-05 )
rep     : ( 0.00758783  2.62595e-96 ) cmp      : ( 0.00266837  0.0431669 )
mov     : ( 0.146049  0.455149 ) movzx    : ( 1.044e-08  0.0176384 )
pop     : ( 0.068666  0.000392012 ) start   : ( 0.00557081  7.76034e-229 )
add     : ( 0.043785  0.0466634 ) popf     : ( 7.26614e-175  0.000106042 )
retn    : ( 0.0300552  0.000497791 ) retf    : ( 0.000347475  0.000649767 )
push    : ( 0.391494  3.59671e-23 ) sar     : ( 9.36913e-05  0.00173289 )
rcl     : ( 9.60485e-05  1.36043e-251 ) xchg    : ( 0.00144432  0.000493542 )
call    : ( 0.120407  0.010618 ) aam      : ( 0.000192097  3.85643e-43 )
popa    : ( 0.000864437  1.25497e-33 )
lea     : ( 0.0397963  0.0109105 )
div     : ( 0.000310892  0.00681368 )
cld     : ( 0.000381809  0.00010692 )
adc     : ( 0.00527914  0.0194777 )
shl     : ( 0.00420063  0.0153855 )
ror     : ( 1.88982e-209  0.00014139 )
inc     : ( 0.00193407  0.00766557 )
or      : ( 0.00783356  0.0553343 )
test    : ( 0.00870597  0.00262839 )
```

Figure 3: HMM model of 1<sup>st</sup> fold

After we trained and score for 5 folds, we obtained the scores for 25 Malware and benign Opcodes as figure 4 and figure 5 shown below.

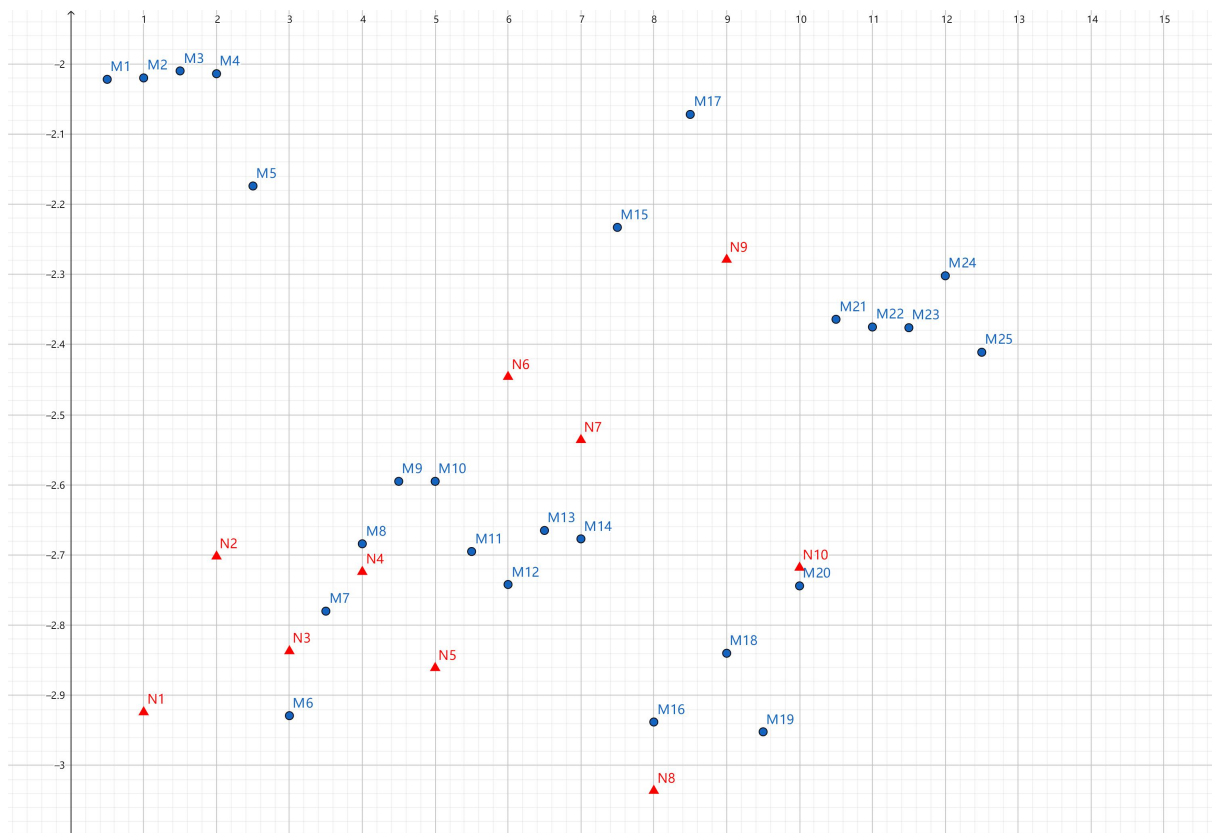
	HMM on Opcode		HMM on Opcode
Cleaman1	-2.022	ChromeSetup	-2.924
Cleaman2	-2.020	Devenv	-2.702
Cleaman3	-2.010	GrammarlySetup	-2.837
Cleaman4	-2.014	IdealU	-2.724
Cleaman5	-2.174	LockDownBrowser	-2.861
Securityshield1	-2.929	ScoringHMM	-2.446
Securityshield2	-2.780	Setup (nxt-g installer v2.0f6 (windows))	-2.536
Securityshield3	-2.684	VirtualBox-6.0.14-133895-Win	-3.036
Securityshield4	-2.740	Webexapp	-2.279
Securityshield5	-2.595	WPSOffice	-2.718
Winwebsec1	-2.695		
Winwebsec2	-2.742		
Winwebsec3	-2.665		
Winwebsec4	-2.677		
Winwebsec5	-2.233		
Zbot1	-2.938		
Zbot2	-2.072		
Zbot3	-2.840		
Zbot4	-2.952		
Zbot5	-2.744		
Zeroaccess1	-2.364		
Zeroaccess2	-2.375		
Zeroaccess3	-2.376		
Zeroaccess4	-2.302		
Zeroaccess5	-2.411		

**Figure 4: HMM scores of Malware Opcodes**

**Figure 5: HMM scores of benign Opcodes**

For the Malware from Cleaman and Zeroaccess family, the scores are generally lower than the scores of the benign files, but most of the scores from Securityshield and Zbot are pretty close to the scores of the benign. These two figures show us using HMM on Securityshield, Zbot and even Winwebsec is not so ideal. However, we had an assumption that the benign files we downloaded might show the function as a malware in some ways. That might explain why the scores don't much separation.

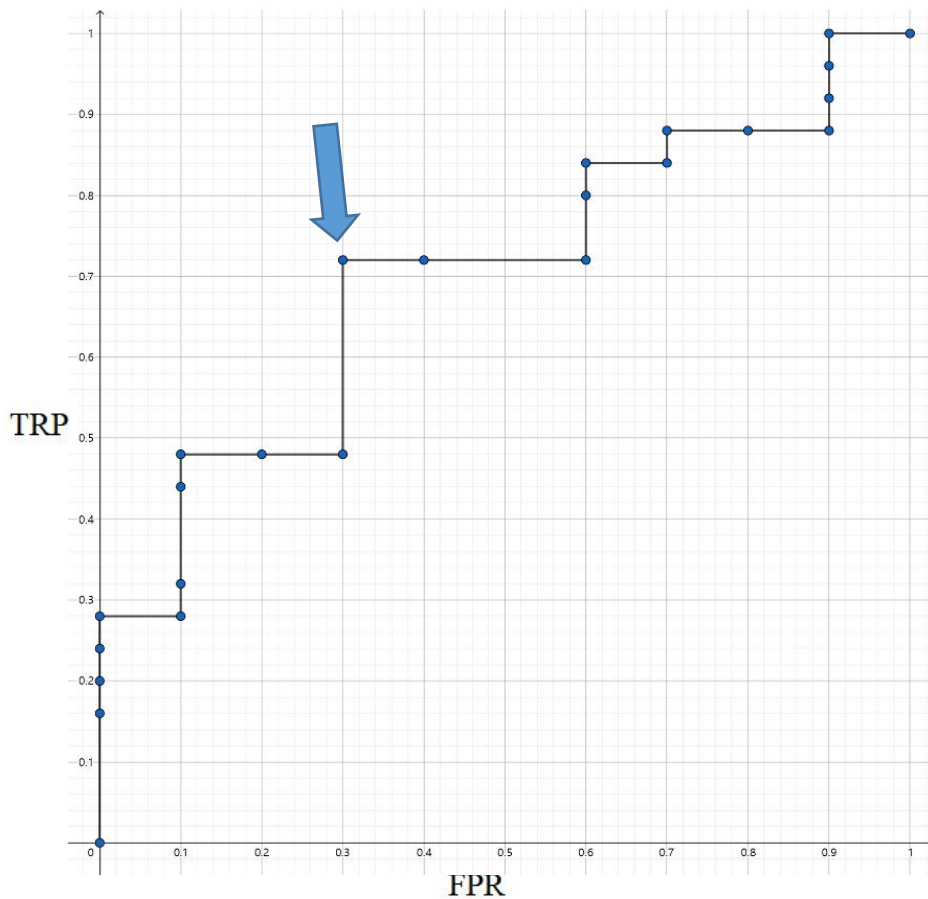
Now, we drew the scores on the graph with circles meaning malware scores and triangle meaning benign scores.



**Figure 6: HMM scores of Malware and Benign**

From this figure, we can see that , most of the Malware points are mixing with benign points, except Cleaman1~Cleaman5 (M1~M5) and Zeroaccess1~Zeroaccess5 (M21~M25).

Then, we drew the ROC curve and PR curve as figure 7 and 8 to see if there was a threshold that can help us to yield a better accuracy and sensitivity.



**Figure 7: HMM ROC**

From this ROC curve, we can see at the point where the arrow is pointing to, the TRP is around 0.7, and the FPR is around 0.3. And the FPR is decreasing from the previous point while the TRP remain the same. However, after this point, the TRP is crashing to lower than 0.5. Then, we find that this point happens at around score -2.7. The accuracy for this threshold is 0.71.

From the PR curve, we also see that while the threshold of score is around -2.7, it yields a Recall = 0.72 and Precision = 0.86. Therefore, we determined to draw the threshold around score = -2.7. That means, if the score is lower than -2.7, the file is more likely to be a benign, and if the score is higher than -2.7, it is more likely to be a Malware.



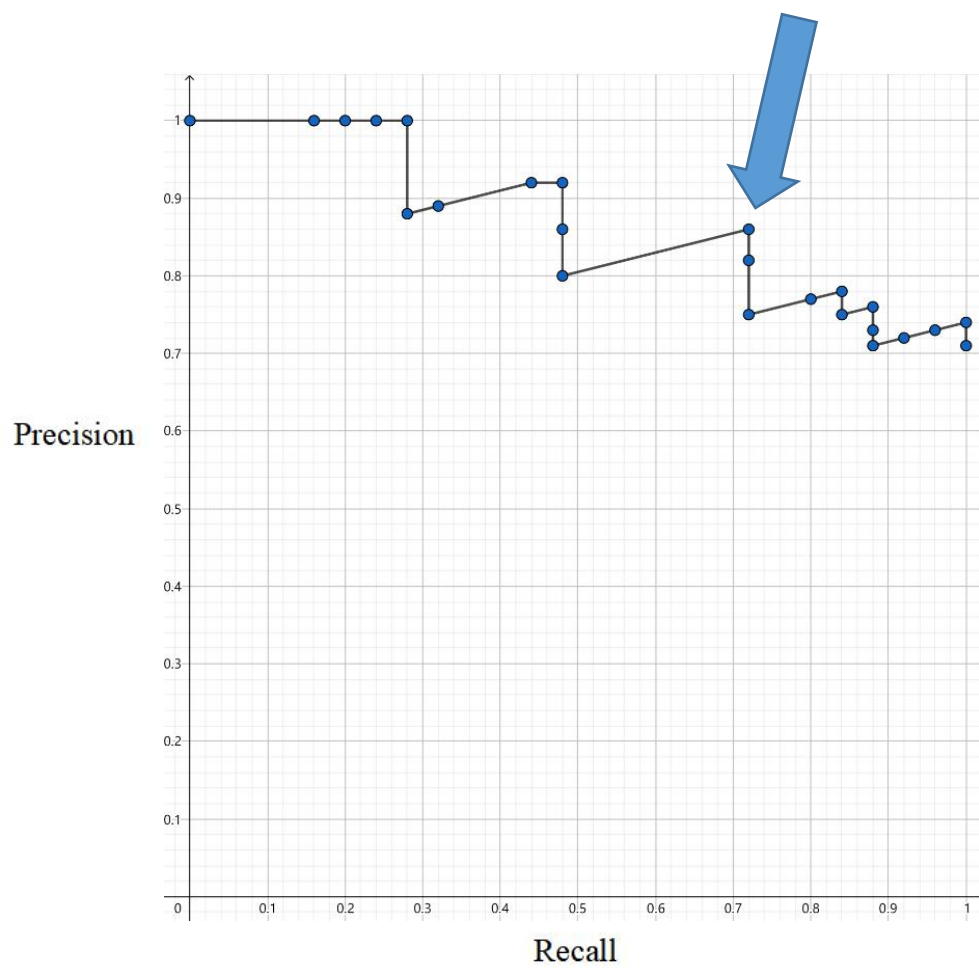
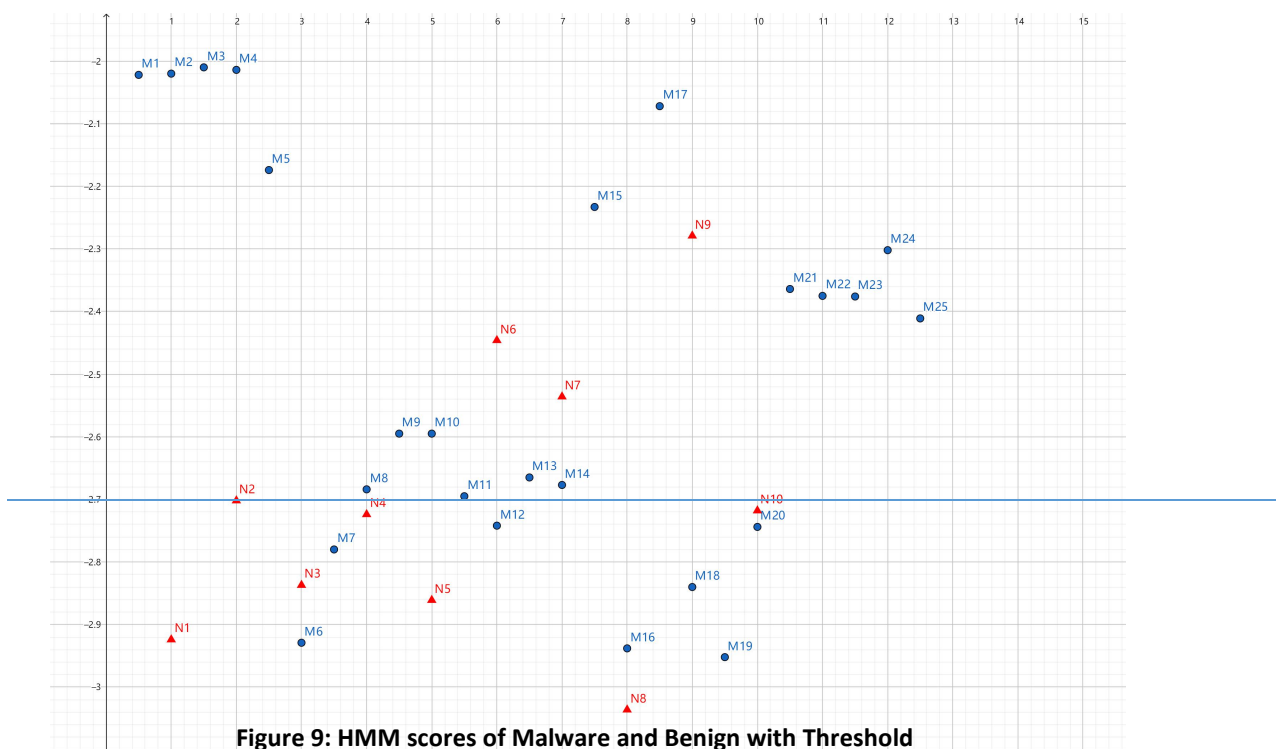


Figure 8: HMM PR





### 3.2 PCA result and Analysis

In the PCA process, we got the eigenvector and eigenvalue. And basing on the eigenvalue, we chose which eigenvector should be saved. We attached the result eigenvector, eigenvalue and significant eigenvector of the 1<sup>st</sup> fold.

```
eigenvector =
0.250056 0.0585006 -0.0366645 0.0052316 -0.0492506 0.0187915 -0.209483 0.0565226 -0.0278998 0.0244788 -0.149358 0.384486
-0.081216 0.210473 -0.808868 0.0783834 -0.081216 0.210473 -0.808868 0.0783834 -0.081216 0.210473 -0.808868 0.0783834
0.250068 0.0791808 -0.00286666 -0.0611939 0.0804471 -0.0790307 -0.269187 0.20525 0.10921 0.161287 0.119437 0.530966
0.46902 0.230952 0.417638 0.135152 -0.0595266 -0.111464 -0.19598 -0.217961 0.0310273 -0.26052 -0.11525 -0.34135
0.250062 0.0631301 -0.0460956 -0.0129955 -0.0595266 -0.111464 -0.19598 -0.217961 0.0310273 -0.26052 -0.11525 -0.34135
0.650192 -0.396466 -0.185569 0.127673 0.0555529 -0.0690425 -0.0705557 0.183396 0.218281 -0.249123 0.478564 -0.467069
0.250066 0.0623999 -0.0574775 0.00911629 0.0555529 -0.0690425 -0.0705557 0.183396 0.218281 -0.249123 0.478564 -0.467069
-0.116356 0.484516 -0.0503932 0.275683 0.00611205 0.0306386 -0.00677438 0.355635 0.214583 0.409152 0.339507 -0.0103537
0.250072 0.0576647 -0.0749969 -0.00641878 0.00611205 0.0306386 -0.00677438 0.355635 0.214583 0.409152 0.339507 -0.0103537
-0.211765 -0.636997 -0.0836506 0.120603 -0.081716 -0.0801006 -0.468626 0.140868 -0.545425 -0.329399 -0.114668 0.00502365
0.250048 0.0466138 -0.0192561 -0.148948 -0.081716 -0.0801006 -0.468626 0.140868 -0.545425 -0.329399 -0.114668 0.00502365
-0.385089 -0.164615 0.248993 0.0603619 -0.0419421 -0.158927 -0.252434 -0.43967 0.430994 0.334113 -0.418095 -0.147747
0.250066 0.0660073 -0.0393585 -0.0179762 -0.0419421 -0.158927 -0.252434 -0.43967 0.430994 0.334113 -0.418095 -0.147747
-0.308608 0.0974677 0.187061 0.115926 -0.196387 0.373073 0.301699 0.498886 0.129622 -0.134672 -0.542373 -0.107098
0.25007 0.0631743 -0.118979 0.184546 -0.196387 0.373073 0.301699 0.498886 0.129622 -0.134672 -0.542373 -0.107098
0.0426618 0.060954 0.125437 0.075846 -0.0570614 -0.0373285 -0.167497 0.123243 0.0961614 0.0483022 0.0672927 -0.148086
0.250061 0.0623704 -0.0448595 -0.00980157 -0.0570614 -0.0373285 -0.167497 0.123243 0.0961614 0.0483022 0.0672927 -0.148086
0.0526164 0.0908689 -0.0164364 -0.91337 -0.154858 -0.718532 0.55468 0.0446951 -0.0589644 -0.0819652 -0.0598849 0.11609
0.250073 0.0807983 -0.0144724 -0.226825 -0.154858 -0.718532 0.55468 0.0446951 -0.0589644 -0.0819652 -0.0598849 0.11609
-0.0425112 -0.0030398 0.00225868 -0.0160896 0.849274 0.0915362 0.184358 -0.0277047 -0.25772 0.159827 -0.149919 -0.131111
0.250122 0.0437166 -0.116433 -0.149012 0.849274 0.0915362 0.184358 -0.0277047 -0.25772 0.159827 -0.149919 -0.131111
0.0292907 0.0413078 -0.0292884 -0.017709 0.85727 0.211428 0.166113 -0.328847 0.326506 -0.560445 0.167104 0.382628
0.250056 0.0564648 -0.0504767 0.160289 0.85727 0.211428 0.166113 -0.328847 0.326506 -0.560445 0.167104 0.382628
-0.194116 -0.199512 -0.199512 -0.0996969 0.166113 -0.328847 0.326506 -0.560445 0.167104 0.382628 0.167104 0.382628
0.250006 0.0462781 -0.0674606 0.801504 -0.100173 -0.0825665 0.101107 -0.201733 -0.379694 0.220671 0.140377 -0.0161963
0.0222611 0.0619104 0.0548703 0.0213068 0.101107 -0.201733 -0.379694 0.220671 0.140377 -0.0161963 0.140377 -0.0161963
0.249231 -0.9481 0.186587 0.019874 0.0167709 -0.0316006 0.00111518 0.0301781 0.033477 -0.0074264 -0.0184174 0.000279488
0.00846106 -0.000911588 0.00131232 0.000328587 0.0167709 -0.0316006 0.00111518 0.0301781 0.033477 -0.0074264 -0.0184174 0.000279488
0.24984 0.219805 0.879311 -0.119559 -0.0826133 0.224035 0.124506 -0.093149 -0.0882924 0.0826877 0.0488358 -0.0535625
0.00876823 0.0218271 0.00102777 0.012128 -0.0826133 0.224035 0.124506 -0.093149 -0.0882924 0.0826877 0.0488358 -0.0535625
0.250104 -0.0609571 -0.375018 -0.427584 -0.37053 0.419098 0.206965 -0.329608 -0.231862 0.183016 0.20686 0.00309393
0.0563903 0.101266 0.0425793 0.0234724 -0.37053 0.419098 0.206965 -0.329608 -0.231862 0.183016 0.20686 0.00309393
```

Figure 10: 16\*16 Eigenvector Space

```
eigenvalue =
5.18498e-15
3.30004e-09
3.98154e-08
7.26992e-07
1.89145e-06
3.64989e-06
1.49184e-05
3.06707e-05
5.09706e-05
0.00016577
0.000209075
0.000687356
0.00183409
0.00296781
0.00804753
0.0241245
```

Figure 11: Eigenvalue

```
after discarding the less significant eigenvectors
eigenvector =
-0.081216 0.210473 -0.808868 0.0783834
0.46902 0.230952 0.417638 0.135152
0.650192 -0.396466 -0.185569 0.127673
-0.116356 0.484516 -0.0503932 0.275683
-0.211765 -0.636997 -0.0836506 0.120603
-0.385089 -0.164615 0.248993 0.0603619
-0.308608 0.0974677 0.187061 0.115926
0.0426618 0.060954 0.125437 0.075846
0.0526164 0.0908689 -0.0164364 -0.91337
-0.0425112 -0.0030398 0.00225868 -0.0160896
0.0292907 0.0413078 -0.0292884 -0.017709
-0.194116 -0.199512 0.0930288 -0.0996969
0.0222611 0.0619104 0.0548703 0.0213068
0.00846106 -0.000911588 0.00131232 0.000328587
0.00876823 0.0218271 0.00102777 0.012128
0.0563903 0.101266 0.0425793 0.0234724
```

Figure 12: Significant Eigenvector

Then we projected the matrix A onto those significant eigenvector and gained the score matrix.

```

score_matrix =
0.0178148 0.028984 0.0273814 0.0096428 -0.0790783 -0.0175749 -0.0606527 -0.0241206 -0.080948 -0.0170629 -0.0129195 0.00839778
0.0245069 0.031987 0.031987 0.118793 -0.000689038 -0.00389634 -0.00111798 -0.00143447
0.0204776 0.00586627 0.00873914 0.0476367 0.0821644 -0.0568491 -0.0372721 -0.0925581 0.0731963 -0.0183877 -0.0636449 0.0212987
0.0242899 0.0928159 0.0928159 -0.02071 -0.044072 -0.0396987 -0.0440242 -0.0520839
-0.059586 -0.080176 -0.0805009 0.0435717 -0.0707933 0.0925013 0.0968496 0.0831629 0.0210861 0.0803706 0.0861485 -0.106703
-0.169296 0.12845 0.12845 0.0808438 -0.0683477 -0.069473 -0.0682267 -0.0683328
-0.120456 -0.136272 -0.120691 -0.480078 0.0902737 -0.0458475 -0.0415405 -0.061339 0.0332095 -0.0299859 -0.0430881 -0.128053
0.0386957 0.187225 0.187225 0.060681 0.152848 0.147341 0.152511 0.15734

```

Figure 13: 4\*20 Score Matrix

After we did 5 Cross Validations, we got the PCA scores for the Malware and benign as

figure 14.

	PCA on Reg Frequency		PCA on Reg Frequency
Cleaman1	0.06181	ChromeSetup	0.1111
Cleaman2	0.02227	Devenv	0.1022
Cleaman3	0.01600	GrammarlySetup	0.1066
Cleaman4	0.01603	IdeaIU	0.0826
Cleaman5	0.2989	LockDownBrowser	0.1032
Securityshield1	0.06035	ScoringHMM	0.08353
Securityshield2	0.1114	Setup (nxt-g installer v2.0f6 (windows))	0.1347
Securityshield3	0.01329	VirtualBox-6.0.14-133895-Win	0.1077
Securityshield4	0.05168	Webexapp	0.1214
Securityshield5	0.01770	WPSOffice	0.1034
Winwebsec1	0.01455		
Winwebsec2	0.08123		
Winwebsec3	0.05356		
Winwebsec4	0.01491		
Winwebsec5	0.05698		
Zbot1	0.04066		
Zbot2	0.1539		
Zbot3	0.2046		
Zbot4	0.04062		
Zbot5	0.1411		
Zeroaccess1	0.0005145		
Zeroaccess2	0.0003901		
Zeroaccess3	0.007353		
Zeroaccess4	0.0006864		
Zeroaccess5	0.008226		

Figure 14: PCA Scores

From figure 14, we can easily see that PCA with the register frequency has a significant result on the Malware detection since most of the Malware scores are much smaller than the benign scores.

We also drew a graph for these scores:

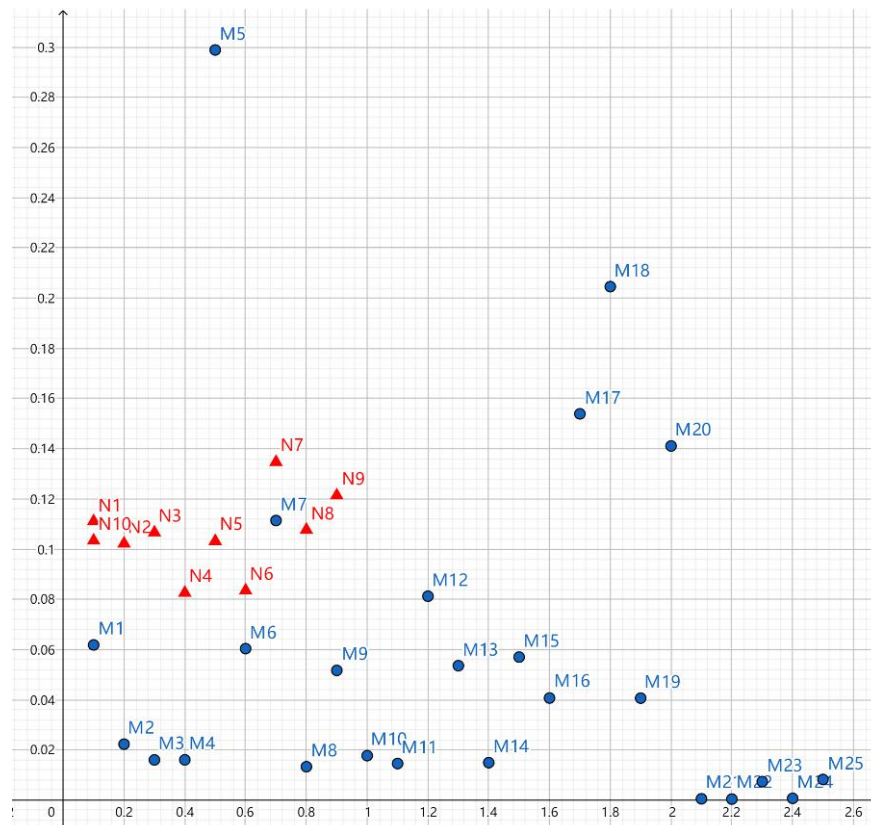


Figure 15: PCA Scores

The blue circles show us the Malware and the red triangles show us the benign. The lower score means the smaller distance between the training sample and scoring sample. That is, if the score is near 0, it means the score sample is much closer to the training sample. From figure 15, we see that most of the scoring Malware samples are close to the training Malware samples. And the benign samples are grouping together and far away from the training samples.

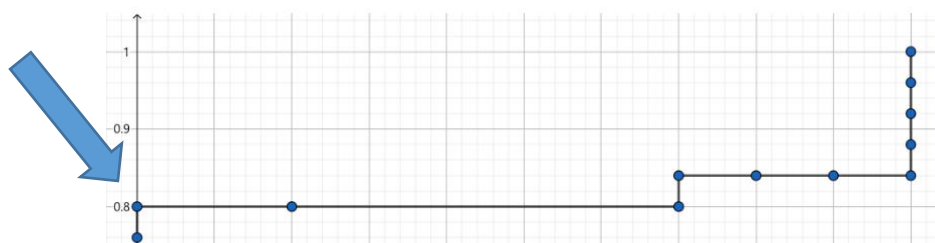
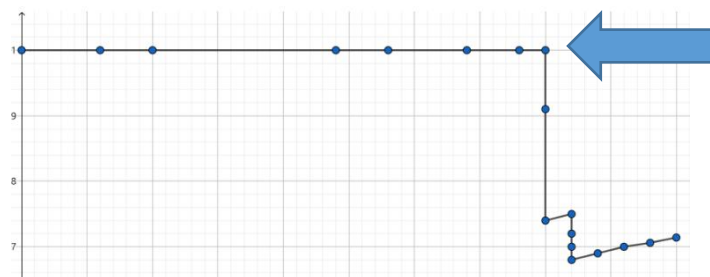


Figure 16: PCA Scores

As our arrow shown, the FPR =0 and TPR = 0.8. And after this point, the TPR is decreasing quickly. When the threshold is around 0.08, we can obtain this FPR and TPR. That means, if we set score =0.08 as a threshold, we can have a 0 FPR, but 0.8 TPR, and

$$Accuracy = \frac{20+10}{25+10} = \frac{30}{35} = 0.86$$

As the arrow point on the figure 17, when score = 0.08, the precision reach 1 and recall keeps at 0.8. Therefore, we can determine that, if the score is lower than 0.08, the scoring sample is most likely to be a Malware while the scoring sample is classified as a benign if the score is higher than 0.08.



Precision

Recall

Figure 17: PCA Scores

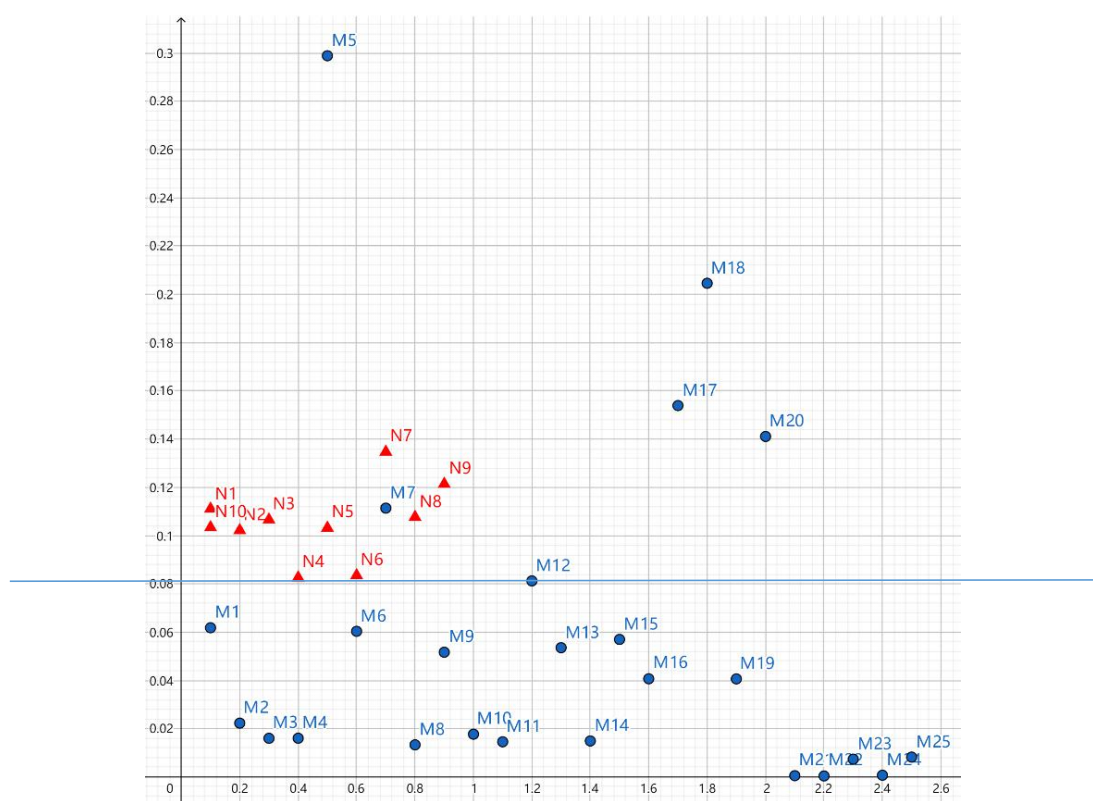


Figure 18: PCA Scores with Threshold

### 3.3 MLP on both HMM and PCA Scores



	HMM on Opcode	PCA on Reg Frequency
Cleaman1	-2.022	0.06181
Cleaman2	-2.020	0.02227
Cleaman3	-2.010	0.01600
Cleaman4	-2.014	0.01603
Cleaman5	-2.174	0.2989
Securityshield1	-2.929	0.06035
Securityshield2	-2.780	0.1114
Securityshield3	-2.684	0.01329
Securityshield4	-2.740	0.05168
Securityshield5	-2.595	0.01770
Winwebsec1	-2.695	0.01455
Winwebsec2	-2.742	0.08123
Winwebsec3	-2.665	0.05356
Winwebsec4	-2.677	0.01491
Winwebsec5	-2.233	0.05698
Zbot1	-2.938	0.04066
Zbot2	-2.072	0.1539
Zbot3	-2.840	0.2046
Zbot4	-2.952	0.04062
Zbot5	-2.744	0.1411
Zeroaccess1	-2.364	0.0005145
Zeroaccess2	-2.375	0.0003901
Zeroaccess3	-2.376	0.007353
Zeroaccess4	-2.302	0.0006864
Zeroaccess5	-2.411	0.008226

	HMM on Opcode	PCA on Reg Frequence
ChromeSetup	-2.924	0.1111
Devenv	-2.702	0.1022
GrammarlySetup	-2.837	0.1066
IdeaIU	-2.724	0.0826
LockDownBrowser	-2.861	0.1032
ScoringHMM	-2.446	0.08353
Setup (nxt-g installer v2.0f6 (windows))	-2.536	0.1347
VirtualBox-6.0.14-133895-Win	-3.036	0.1077
Webexapp	-2.279	0.1214
WPSOffice	-2.718	0.1034

Figure 19: HMM and PCA Scores

Before we chose the data set using the threshold we determine above, we chose the training data from the HMM and PCA scores of the Malware. For example, we chose only 3 Zbot scores:

Zbot2	-2.072	0.1539
Zbot3	-2.840	0.2046
Zbot5	-2.744	0.1411

And we choose 3 benign scores:

ScoringHMM	-2.446	0.08353
Webexapp	-2.279	0.1214
WPSOffice	-2.718	0.1034

Then we got the training scores and testing scores:



```

error E[w] = 0.00354133
w0 = 0.189236 w1 = 0.375979 w2 = -11.8426 w3 = 13.0362 w4 = -6.4184 w5 = 1.70512
Y[0] = 0.686799
Y[1] = 1.10247
Y[2] = 0.533424
Y[3] = -0.292904
Y[4] = 0.291132
Y[5] = 0.0310502

```

```

29
test sample Y[0] = -0.713436
test sample Y[1] = -1.55109
test sample Y[2] = -1.69381
test sample Y[3] = -1.69261
test sample Y[4] = 1.50902
test sample Y[5] = -0.688823
test sample Y[6] = 0.148102
test sample Y[7] = -1.66374
test sample Y[8] = -0.8643
test sample Y[9] = -1.57981
test sample Y[10] = -1.6349
test sample Y[11] = -0.32482
test sample Y[12] = -0.832897
test sample Y[13] = -1.62952
test sample Y[14] = -0.794766
test sample Y[15] = -1.06511
test sample Y[16] = -1.06469
test sample Y[17] = -1.99465
test sample Y[18] = -1.99576
test sample Y[19] = -1.83855
test sample Y[20] = -2.00039
test sample Y[21] = -1.8139
test sample Y[22] = 0.141985
test sample Y[23] = 0.0128997
test sample Y[24] = 0.0780503
test sample Y[25] = -0.301957
test sample Y[26] = 0.027519
test sample Y[27] = 0.461487
test sample Y[28] = 0.0917218

w0 = 0.189236 w1 = 0.375979 w2 = -11.8426 w3 = 13.0362 w4 = -6.4184 w5 = 1.70512

```

Figure 20: MLP Scores of Sandom Training Sample

If the score is higher than 0.5, we would round it to 1, and if lower than 0.5, we would round it to 0. And “1” means, it’s Malware when “0” means its benign. Then, we calculated the accuracy basing on the classification in the figure 20.

$$Accuracy = \frac{11}{35} = 0.31$$

The accuracy is very low.

But when we chose the training data basing on the threshold we determined before. For example, we choose

Securityshield3	-2.684	0.01329
Winwebsec2	-2.742	0.08123
Winwebsec5	-2.233	0.05698

IdeaIU	-2.724	0.0826
LockDownBrowser	-2.861	0.1032
VirtualBox-6.0.14-133895 -Win	-3.036	0.1077

We had the training scores and testing scores as below:

```
error E[w]= 0.0235859
w0 = 3.21505 w1 = 0.761547 w2 = -1.03171 w3 = -14.2614 w4 = -1.44008 w5 = 9.98143
Y[0] = 0.965749
Y[1] = 0.373596
Y[2] = 0.74699
Y[3] = 0.371492
Y[4] = 0.252588
Y[5] = 0.208309
How many test sets do you have?
29
test sample Y[0] = 0.812054
test sample Y[1] = 1.34715
test sample Y[2] = 1.46436
test sample Y[3] = 1.46004
test sample Y[4] = 0.0258008
test sample Y[5] = 0.433781
test sample Y[6] = 0.23929
test sample Y[7] = 0.559341
test sample Y[8] = 0.969937
test sample Y[9] = 0.943013
test sample Y[10] = 0.575509
test sample Y[11] = 0.950347
test sample Y[12] = 0.562812
test sample Y[13] = 0.222734
test sample Y[14] = 0.0615325
test sample Y[15] = 0.557481
test sample Y[16] = 0.162225
test sample Y[17] = 1.40591
test sample Y[18] = 1.39797
test sample Y[19] = 1.28191
test sample Y[20] = 1.46071
test sample Y[21] = 1.23891
test sample Y[22] = 0.21592
test sample Y[23] = 0.28807
test sample Y[24] = 0.245242
test sample Y[25] = 0.449108
test sample Y[26] = 0.20715
test sample Y[27] = 0.301293
test sample Y[28] = 0.279991
w0 = 3.21505 w1 = 0.761547 w2 = -1.03171 w3 = -14.2614 w4 = -1.44008 w5 = 9.98143
```

Figure 21: MLP Scores of Training Sample Basing on Threshold

$$\text{The Accuracy} = \frac{28}{35} = 0.8$$

And we also used single layer perceptron to train and classify on these samples, and we gained:

```
error E[w] = 0.0790826
w0 = -0.20184 w1 = -10.3599
Y[0] = 0.599662
Y[1] = 0.428472
Y[2] = 0.465157
Y[3] = 0.424112
Y[4] = 0.379498
Y[5] = 0.376842
```

Figure 22: Single Layer Perceptron Scores of Training Sample Basing on Threshold

$$\text{Accuracy} = \frac{25}{35} = 0.71$$

```
How many test sets do you have?
29
test sample Y[0] = 0.442203
test sample Y[1] = 0.544135
test sample Y[2] = 0.559699
test sample Y[3] = 0.559821
test sample Y[4] = 0.0655113
test sample Y[5] = 0.491493
test sample Y[6] = 0.355952
test sample Y[7] = 0.504411
test sample Y[8] = 0.584289
test sample Y[9] = 0.597058
test sample Y[10] = 0.495757
test sample Y[11] = 0.595286
test sample Y[12] = 0.542838
test sample Y[13] = 0.23574
test sample Y[14] = 0.175605
test sample Y[15] = 0.543642
test sample Y[16] = 0.287423
test sample Y[17] = 0.615815
test sample Y[18] = 0.616644
test sample Y[19] = 0.599503
test sample Y[20] = 0.612427
test sample Y[21] = 0.599028
test sample Y[22] = 0.363361
test sample Y[23] = 0.374394
test sample Y[24] = 0.37011
test sample Y[25] = 0.40814
test sample Y[26] = 0.29243
test sample Y[27] = 0.310518
test sample Y[28] = 0.372241
w0 = -0.20184 w1 = -10.3599
```

From these experiments, we knew that when we set up the MLP or single layer perceptron, it's very important to choose the training data that corresponding to the threshold.

And we also want to know which score from HMM or PCA has more impact on the MLP training. Therefore, we decide to make some change to the training samples. We again chose the Malware scores from Zbot family but with low HMM score ( we said lower HMM score is more likely to be classified as benign) and also low PCA score (lower PCA score is more likely to be classified as Malware ). And we also chose the benign that has higher HMM scores and higher PCA score.

Zbot1	-2.938	0.04066
Zbot4	-2.952	0.04062
Securityshield1	-2.929	0.06035
ScoringHMM	-2.446	0.08353
Webexapp	-2.279	0.1214
WPSOffice	-2.718	0.1034

Then we obtained the result as below:

```
error E[w]= 0.0355306
w0 = 0.274388 w1 = -0.508331 w2 = 11.6453 w3 = -5.78965 w4 = -2.76912 w5 = 2.61319
Y[0] = 0.878507
Y[1] = 0.884709
Y[2] = 0.664947
Y[3] = 0.18861
Y[4] = -0.304529
Y[5] = 0.0970843
```

```
29
test sample Y[0] = 0.230113
test sample Y[1] = 0.674859
test sample Y[2] = 0.73887
test sample Y[3] = 0.740356
test sample Y[4] = -1.7117
test sample Y[5] = 0.0391414
test sample Y[6] = 1.05513
test sample Y[7] = 0.676529
test sample Y[8] = 0.974397
test sample Y[9] = 1.04695
test sample Y[10] = 0.352282
test sample Y[11] = 1.03611
test sample Y[12] = 0.622884
test sample Y[13] = 0.386198
test sample Y[14] = -0.727373
test sample Y[15] = -0.830334
test sample Y[16] = -0.289137
test sample Y[17] = 1.05334
test sample Y[18] = 1.05907
test sample Y[19] = 0.989324
test sample Y[20] = 1.02616
test sample Y[21] = 0.994976
test sample Y[22] = 0.108832
test sample Y[23] = 0.102722
test sample Y[24] = 0.117476
test sample Y[25] = 0.328853
test sample Y[26] = 0.165488
test sample Y[27] = -0.320833
test sample Y[28] = 0.196527
```

Figure 22:MLP Score

$$Accuracy = \frac{27}{35} = 0.77$$

Therefore, now we know that if we choose the Malware samples (that have lower PCA score) and the benign samples (that have higher PCA score) to be the MLP training sample, we will probably get the model with higher accuracy. That is, PCA plays an important role when we train a MLP.

#### **4. Conclusion**

From this project, we found that PCA with register frequency played very good in Malware detection, it yielded an accuracy with 0.86. HMM with Opcode sequence also had an accuracy with 0.71, but the malware and benign samples were hardly separated. However, the choices of benign samples in this project maybe not so good. In this situation, PCA can solve this problem effectively. Furthermore, with the correct choice of training samples (especially choosing the samples with significant PCA scores), MLP can perform better than HMM. It yielded an 0.8 accuracy in this project.