Spring 5-25-2015

Clustering versus SVM for Malware Detection

Usha Narra
San Jose State University

Clustering versus SVM for Malware Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Usha Narra

May 2015

The Designated Project Committee Approves the Project Titled

Clustering versus SVM for Malware Detection

by

Usha Narra

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2015

Mark Stamp	Department of Computer Science
Robert Chun	Department of Computer Science
Fabio Di Troia	Università del Sannio

ABSTRACT

Clustering versus SVM for Malware Detection

by Usha Narra

Previous work has shown that we can effectively cluster certain classes of malware into their respective families. In this research, we extend this previous work to the problem of developing an automated malware detection system. We first compute clusters for a collection of malware families. Then we analyze the effectiveness of classifying new samples based on these existing clusters. We compare results obtained using k-means and Expectation Maximization (EM) clustering to those obtained using Support Vector Machines (SVM). Using clustering, we are able to detect some malware families with an accuracy comparable to that of SVMs. One advantage of the clustering approach is that there is no need to retrain for new malware families.

ACKNOWLEDGMENTS

I would like to express my gratitude to my project advisor, Dr. Mark Stamp for his continued support and guidance throughout the project. He has always been very patient in explaining things and very encouraging through out the project.

A very special thanks to my committee member, Fabio Di Troia for his valuable inputs and suggestions.

I would also like to thank my committee member, Dr.Robert Chun for his time and guidance.

Last but not least, I would like to thank my husband, Mr.Jayachandra Gullapalli for his support and guidance throughout my Masters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is a piece of software that runs unauthorized tasks without the consent of the user. The tasks performed by malware intend to do harm to the user by gaining access to their sensitive information stored on the system or by tricking the user to provide their personal and/or financial information. To protect the user, system and any critical information stored on it, malware has to be identified and eliminated from the system. Malware is a broad term that encompasses various types of computer viruses, worms, trojan horses, spyware, adware etc. A more detailed view on different kinds of malware is presented in Chapter 2.

Stuxnet is an advanced form of malware (worm) that is known worldwide for its attack on industrial control systems. The distinguishing feature that makes Stuxnet stand out from other worms is its access control over physical systems [45]. Stuxnet came to light in the year 2010, June. It was programmed to infect the industrial programmable logic controllers (PLC's), targeting Iran's nuclear facilities. Stuxnet exploits vulnerabilities in industrial control systems and spreads through removable USB drives. A component of Stuxnet (rootkit) hid all the background activity on the system.

Koobface is a type of malware that is platform independent and spreads through social networking sites. It wouldn't be an exaggeration to say that every person who has access to internet uses social networking sites these days. Koobface takes advantage of this fact and targets the users on social networking sites such as Skype, Facebook, Twitter and mailing services like GMail and Yahoo Mail! Koobface is not a single module of code, it contains several units each performing its respective task. All the units coordinate with each other to form the Koobface worm. The worm resides in the spam of these websites and the users who encounters it get infected. From the infected system, the Koobface collects the user's login details and also other user profile information from any social networking sites accessed from that system. One important unit that works to the advantage of Koobface is its rouge DNS (Domain Name System) changer unit which prevents the user form accessing anti-virus software or security websites [44].

It is clear from the above information that malware is a serious threat in this digital era. According to [43], malware or harmful programs are adapting to the increasing use of anti-virus software at a fast rate and are growing more powerful, posing danger to the community at large. Some malware stay dormant for a long time and are capable of spreading rapidly across the network when they encounter loopholes in the security of the system. Anti-virus software is the most widely used defense mechanism against malware, but are not capable of detecting advanced malware. Most anti-virus software use signature based detection which identifies a malware by its signature. The drawback of this technique is it cannot identify obfuscated or new class of malware that does not match the list of known signatures. By the time security experts identify the new malware and trace out its signature, the malware would have enough time to carryout its actions. There are several other techniques used to detect malware which are discussed in Chapter 2. Inspite of all the existing techniques, there is no classification model that can differentiate the new malware and fit it into the existing models. More research and experiments are needed in the area of detection of malware to develop a technique which can effectively combat evolving malware.

In this paper, we apply Hidden Markov Models (HMM) and clustering techniques to the problem of classifying new/unknown malware. The combination of HMM and clustering techniques is inspired from previous research in malware classification [2, 3]. The basic idea of this paper is to train the HMMs [47] for different malware families. These trained models are then used to score malware samples. Based on these scores, the malware samples are grouped into clusters using Expectation Maximization (EM) [9] and k-means clustering algorithms [1]. Analysis of the resulting clusters indicate that HMM-based clustering can be an effective tool for classifying new/previously unknown malware. We compare the results of both the clustering techniques to see which one does a better job in classifying different malware families. We also classify the same malware samples using Support Vector Machines (SVM) to see how well clustering performs as compared to SVM.

This paper is organized as follows: Chapter 2 defines various kinds of malware and describes some of the existing malware detection techniques and concealment strategies adopted by malware writers. Chapter 3 provides insight into previous work done on malware classification. Chapter 4 provides details about Clustering (especially EM clustering) and SVM, while Chapter 5 discusses our implementation and experimental setup for clustering and SVM in more detail. In Chapter 6 experimental results are presented and analyzed. Finally, Chapter 7 contains our conclusion and suggestions for future work.

CHAPTER 2

Malware Detection and Concealment Strategies

2.1 Malware

Malware is a computer program that performs malicious actions [46]. Unlike a bad piece of software which may not provide the intended functionality due to accidental bugs in the code, malware is written to cause harm.

As mentioned in Chapter 1, malware is used to refer computer infections such as worms, viruses, spyware, trojan horses, adware etc. A short description of the various types of malware is as follows:

2.1.1 Virus

Majority of the known malware today are viruses. A computer virus is a self-replicating program that links itself to other executables to spread the infection. This implies that the infected program is capable of spreading the infection to the other executable programs [21]. Virus may conceal itself anywhere in the system such as the executable files, data files, hidden folders or even in the boot sector. A virus that resides in the boot-sector of the hard drive is known as boot-sector virus and it becomes active at the start-up of the system [7]. Thus it makes itself immune to any anti-virus software that starts with the booting of the system or even before any security at the operating system level is enabled. As it is one of the first programs to run during the boot process, it can even block anti-virus software from executing on the system. Some of the historic examples of computer viruses are Brain, Lehigh, MBDF, Pathogen, Melissa [34].

2.1.2 Worm

Worms are similar to the viruses as they create copies of themselves except that worms do not require a host file. Thus they differ from viruses which require an infected host file for spreading [40]. According to [4], worms are independent programs and they do not depend on other host files. A worm usually spreads on a network compromising the systems on its way in the network. Some of the examples of famous worms are Morris, Ramen, Lion, Adore, Code Red, Nimda [30].

2.1.3 Trojan Horse

A Trojan horse is a malware which pretends to do some legitimate task, but secretly performs malicious tasks [4]. Trojan horse differs from virus as it does not spread itself [40]. They camouflage as harmless programs or files that the end users might mistake for authentic programs and use them [37]. But when the user runs the programs assuming them to be known software, the trojans perform undesired activities in the background. Some of the notable trojan horses are Netbus, Subseven, Back Orifice [30].

2.1.4 Spyware

Spyware is a form of malware when installed in a system steals the user's credentials from the system of which the user is unaware of [25]. Spyware generally conceals itself and collects data related to keystrokes, browsing patterns, credit card details, personal information etc. At times the activities of the spyware can get far beyond normal monitoring and carries out activities such as installing additional software, reducing connection speeds, redirecting web browser searches, changing computer settings etc [25]. Unlike viruses and worms, spyware do not usually make copies of themselves. Some of the examples of spyware are CoolWebSearch, HuntBar, WeatherStudio, Zango, Zlob [23].

2.2 Detection Techniques

Most anti-virus software in use today depend on static detection techniques. Static analysis extracts interesting features from file without executing the file and studies them whereas dynamic analysis extracts the actual behavior of the file by running it in a sandbox. In this section we look at some of the static malware detection techniques, for example, signature detection, change detection, anomaly detection and HMM based detection.

2.2.1 Signature Detection

Amongst static detection techniques, signature detection is the most popular one. Signature detection depends on discovering a series of bits (signature) which uniquely identifies a particular malware [31]. That is, the anti-virus scanner scans all the files in the system looking for some specific signatures. The anti-virus software saves all the signatures of known malware in a database and frequently updates this database to include any new signatures. When the scanner finds a match within the list of known malware signatures in its database, the file will be marked as malware.

Signature detection works on known malware whose signature is already in the database. The major disadvantage of this approach is it cannot detect new (previously unknown) or even obfuscated malware. Also, another important factor to consider in signature detection is the signature database. The efficiency of this technique entirely depends on the quality of the database. To have an effective detection rate, the database must be regularly updated [31].

2.2.2 Change Detection

When a file is infected by malware, the contents of the file are changed. Change detection monitors the files looking for any unapproved changes [4]. The basic idea behind this technique is to calculate the hash values of all the files on the system and save them in a file. From time to time we re-calculate the hashes of all files and compare them with the actual hash values saved in the file. Any mismatch in hash values indicates that the file has been modified, which indicates that the file might have be infected with malware. That is, change detection checks the integrity of the files on the system periodically and reports red flags.

One advantage with change detection is that any changes to a file can be detected without fail which guarantees that this technique could atleast detect if not classify new malware. But the real problem with change detection is that files on a system change regularly and this may prompt numerous false positives thereby reducing the efficiency of detection [31].

2.2.3 Anomaly Detection

Anomaly-based detection is based on the organization of file contents [13]. The structural details of a file can be considered as its attributes. These attributes are extracted and provided to a machine learning algorithm that is trained to classify ordinary from anomalous file structures. But there is no set definition for normal and unusual structure of programs. This poses a fundamental challenge to anomaly detection, defining what is normal and what is unusual. Depending on how we train the learning model and the definitions we give for normal and unusual behavior, there may be numerous false positives and false negatives.

In an anomaly-based detection technique mentioned in [16], the author developed a malware detector model which can classify the files on the system into different categories based on the structural organization of the files. Any new file that needs to be categorized is given to the detector model. If the file looks similar to the model, it is marked as benign otherwise as malware. When a file is marked as malware it is further reviewed to determine whether it is actually malicious or just a false positive.

Since anomaly detection alone might give numerous false positives, it is often combined with other techniques such as signature-based detection for further screening [31].

2.2.4 HMM Based Detection

Some of the previous research [2, 47] prove that HMM can be an effective tool for malware detection. Apart from malware detection, HMMs are also used for speech recognition [27], gene prediction, human activity recognition [26], etc. In the following sections we give a brief overview of HMM and how it is used for malware detection.

2.2.4.1 Introduction to HMM

In general, a Markov process is a statistical model in which the internal states and state transitions are transparent to the user. A HMM can be considered as a special case of Markov model as the states in HMM are not visible to the user (hidden), consequently the name 'Hidden' Markov Model. Table 1 provides the standard notation used for HMM.

The three matrices A, B and π together represent the HMM. These three matrices are row stochastic in nature. A matrix is said to be row stochastic if the sum of all the elements in a row is equal to 1. So, an HMM can be written in terms of these matrices as $\lambda = (A, B, \pi)$. Figure 1 illustrates a generic view of Hidden Markov Model.

Table 1: HMM Notation

Notation	Description
T	Length of observation sequence
N	Number of hidden states in the model
M	Number of distinct observation symbols
Q	Distinct states of the Markov process
V	Set of possible observations
A	State transition probability matrix, $N \times N$
B	Observation probability matrix, $N \times M$
π	Initial state distribution matrix, $1 \times N$
$\mathcal{O}$	Observation sequence

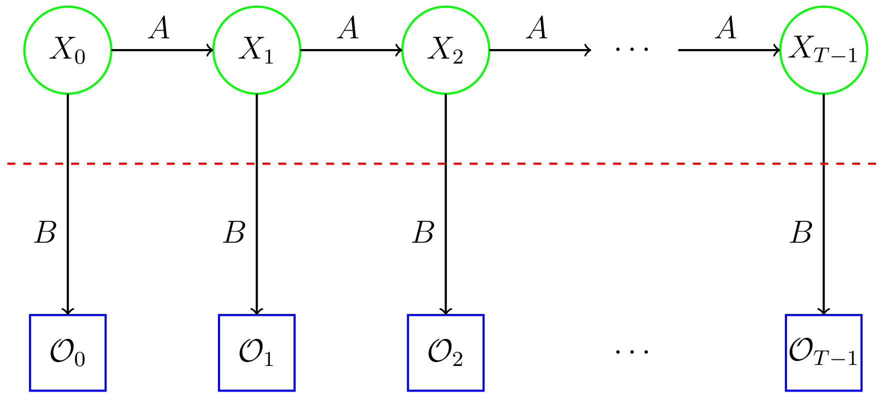

Figure 1: Generic View of HMM

We can solve three problems using HMM. The three problems and algorithms to solve these problems [32] are as follows:

Problem 1:

Given a model $\lambda = (A, B, \pi)$ and an observation sequence $\mathcal{O}$, we want to find the probability $P(\mathcal{O}|\lambda)$.

Problem 2:

Given a model $\lambda = (A, B, \pi)$ and an observation sequence $\mathcal{O}$, we want to find the most likely hidden state sequence of the model.

Problem 3:

Given a scenario where the observation sequence $\mathcal{O}$ and dimensions N and M are known, the model λ which increases the probability of $\mathcal{O}$ need to be found.

For the research presented in this paper, HMM models are trained to represent different compilers and malware families. This is similar to problem 3 mentioned above. A large collection of malware and benign samples are scored using these HMM models. Again, this is similar to problem 1 mentioned above. So, in this paper we are going to use the HMM algorithms that solve the Problems 1 and 3. Now, we discuss these HMM algorithms (forward algorithm, backward algorithm and Baum-Welch algorithm) to solve these problems in detail.

Solution to Problem 1:

Our aim in problem 1 is to find the probability of an observation sequence, $P(\mathcal{O}|\lambda)$ with respect to $\lambda = (A, B, \pi)$. To find the $P(\mathcal{O}|\lambda)$ we use the forward algorithm or α-pass. For $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, we define

$$\alpha_t(i) = P(\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_t, x_t = q_i|\lambda). \tag{1}$$

where $\alpha_t(i)$ corresponds to the probability of the partial observation sequence until time t, when the Markov process is in state q_i at time t.

The forward algorithm can be computed recursively as follows:

1. Let $\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$, for $i = 0, 1, \ldots, N-1$

2. For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, compute

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right] b_i(\mathcal{O}_t)$$

3. From Equation (1), the desired probability is given by

$$P(\mathcal{O}|\lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i)$$

Solution to Problem 2:

Our aim in problem 2 is to find the most likely state sequence. In order to do that, we use the backward algorithm or β-pass. This is similar to the forward algorithm, except as the name suggests it starts at the end and finds its way to the beginning. For $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, we define

$$\beta_t(i) = P(\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \ldots, \mathcal{O}_{T-1}|x_t = q_i, \lambda).$$

The backward algorithm can be computed recursively as follows:

1. Let $\beta_{T-1}(i) = 1$, for $i = 0, 1, \ldots, N-1$

2. For $t = T-2, T-3, \ldots, 0$ and $i = 0, 1, \ldots, N-1$, compute

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)$$

For $t = 0, 1, \ldots, T-2$ and $i = 0, 1, \ldots, N-1$ we define,

$$\gamma_t(i) = P(x_t = q_i|\mathcal{O}, \lambda),$$

Since $\alpha_t(i)$ measures the relevant probability until time t and $\beta_t(i)$ measures the probability after time t, we can write $\gamma_t(i)$ as follows

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O}|\lambda)} \tag{2}$$

Solution to Problem 3:

Our aim in problem 3 is to train a model that best fits the observations. One of the interesting aspects of HMM is its ability to re-estimate the model itself. We already know the size of the matrices A, B and π, we just need to find the elements of these matrices. In order to find these elements, we use the Baum-Welch algorithm. Initially we define "di-gammas" as,

$$\gamma_t(i,j) = P(x_t = q_i, x_{t+1} = q_j | \mathcal{O}, \lambda)$$

for $t = 0, 1, \ldots, T-2$ and $i = 0, 1, \ldots, N-1$.

Where $\gamma_t(i,j)$ is the probability of being in state q_i at time t and going to state q_j at time $t+1$. Writing the di-gammas in terms of α, β, A and B gives,

$$\gamma_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)}{P(\mathcal{O}|\lambda)} \tag{3}$$

Equations (2) and (3) are related as follows:

$$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i,j)$$

Re-estimating the model is an iterative process. The process starts by initializing the model, $\lambda = (A, B, \pi)$ with random values such that $\pi_i \approx 1/N$ and $a_{ij} \approx 1/N$ and $b_j(k) \approx 1/M$. Note that the values of A, B, π are random. This is important to note because if we initialize the matrices with exactly uniform values, the model might get stuck in a local maximum. The iterative process can be summarized as follows:

1. Initialize, $\lambda = (A, B, \pi)$.

2. Compute $\alpha_t(i), \beta_t(i), \gamma_t(i,j)$ and $\gamma_t(i)$.

3. Re-estimate the model $\lambda = (A, B, \pi)$ as follows:

For $i = 0, 1, \ldots, N-1$, let

$$\pi_i = \gamma_0(i)$$

For $i = 0, 1, \ldots, N-1$ and $j = 0, 1, \ldots, N-1$, compute

$$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i,j) \Bigg/ \sum_{t=0}^{T-2} \gamma_t(i)$$

For $j = 0, 1, \ldots, N-1$ and $k = 0, 1, \ldots, M-1$, compute

$$b_j(k) = \sum_{t \epsilon (0,1,\ldots,T-2), \mathcal{O}_t = k} \gamma_t(j) \Bigg/ \sum_{t=0}^{T-2} \gamma_t(j)$$

4. If $P(\mathcal{O}|\lambda)$ increases, go to 2.

2.2.4.2 HMM for Malware Detection

A trained HMM model can be used to classify malware from benign samples. There is a lot of previous work done on using HMM for malware detection. According to [47], we can 'train' an HMM on a set of files such that the trained model represents a particular family (either malware or benign on which it has been trained). Now, given a new observation sequence which is not a part of training data, we can score the sequence using the trained HMM. The model will output a high score if the sequence is similar to the training data and output a low score otherwise. In an ideal case, the model should output a score of 1 for a sequence on which it has been trained.

Previous research [2, 15] has shown that HMMs trained on opcode sequences and/or function call-graphs can effectively classify different families of malware.

2.3 Concealment Strategies

With more and more malware detection software coming into light, malware writers are using many concealment strategies to prevent their malware from being detected by regular malware detection techniques. Some of the concealment strategies are discussed below:

2.3.1 Encrypted Virus

As mentioned in Section 2.2.1, signature detection is the most commonly used malware detection technique. One basic approach used even by novice malware writers to evade signature detection is encryption [5]. The major advantage of encrypted virus is it does not need strong cryptographic encryption techniques. A simple encryption such as XOR operation on the malware file completely changes the signature of the malware. But the set back is an encrypted virus cannot execute directly because the instructions are encrypted and cannot be read by the system. In order to execute the malware, it first needs to be decrypted. So encrypted virus comes as a package containing decryption code and actual encrypted malicious code known as the virus body [28]. So when the malware is executed, the decryption code executes the virus body by decrypting it and provides control to the virus body. The virus now performs its malicious activity.

Encryption seems to be an easy technique to avoid signature detection. But this simple technique comes with a drawback. Instead of trying to find the signature of the virus body, signature detection system can try to find the signature of decryption code which if found could indicate the presence of malware.

2.3.2 Polymorphic Virus

Polymorphic virus is an improvement over encrypted virus. It is built on the same idea as encrypted virus but with additional features to overcome the drawbacks of encryption. The new feature in polymorphic virus is its ability to create a new decryption code with every infection [28]. As a result, the decryption code looks different across different infections of the same virus. So a single signature of the decryption code cannot detect different generations of the same virus. Even though polymorphic virus can evade signature detection, it can be caught when run in an emulated environment [5].

2.3.3 Metamorphic Virus

Metamorphic virus is built on top of polymorphic malware. Instead of just changing the decryption code with every infection, metamorphic virus changes entire virus body still keeping the virus functionality unchanged. Peter Szor quoted in [41], the shortest definition of metamorphic virus defined by Igor Muttik is “Metamorphics are body-polymorphics”. Every single conceivable strategy appropriate for polymorphic virus to create new decryptor can be utilized by a metamorphic virus to create a new virus body to make another copy of virus [28]. As a result, different generations of metamorphic virus look completely different yet have the same malicious intent.

CHAPTER 3

Related Work

Automatic malware classification is a challenging task and a lot of previous work has been done in this area. Some of the previous research techniques include behavioral analysis, compression based analysis etc. One of the recent research papers [2] which inspired the current paper uses clustering techniques to detect malware. In this chapter we discuss some of the previous work done which includes both static and dynamic detection techniques.

3.1 MEDiC

Malware Examiner using Disassembled Code (MEDiC) is a static malware detection technique. The underlying logic for MEDiC is an extension of signature-based malware detection. Instead of using the standard signature database, MEDiC creates its own database by extracting necessary features from assembly code. This technique is developed with the intent of detecting obfuscated and mutated malware which cannot be identified by regular signature detection [35]. As the name suggests, the first step in MEDiC is to disassemble the executable file using a disassembler. A disassembler converts machine code into assembly instructions. MEDiC then extracts appropriate subroutines from the assembly code. A subroutine in assembly code consists of a label and instructions associated with it. Since there will be many labels in the assembly code, each subroutine is treated as a checkpoint. Each label and instruction set combination is stored as key-value pairs where label is the key and instructions associated with that label is the value. These key-value pairs are analogous to the signature database. Based on the average length of the subroutine, a dictionary threshold is determined. This whole process can be considered as building the MEDiC model.

In order to test the MEDiC model, the test file is first disassembled to get the assembly code. Then, the assembly code is parsed to extract the sequence of key-value pairs. The length of each key-value pair is compared with the dictionary threshold value. If the length exceeds the threshold, the key-value pair is counted towards the number of matches and the number of subroutines count towards the number of checks. This process is called MEDiC signature matching. Based on the result of signature matching, the detection algorithm calculates the virus threshold. The virus threshold is calculated as,

$$\text{virus threshold} = \frac{\text{number of matches}}{\text{number of checks}}$$

The test file undergoes multiple levels of scanning before it is classified as malicious or benign. The main reason behind multiple scanning is to reduce the number of false positives. In the first level of scanning, the checkpoints extracted from the test file are matched against the signature database. If more checkpoints match the signature database than the virus threshold the file is marked as malicious. If the file passes this level, it enters second level of scanning where the key names are excluded in the matching process. This second level of scanning is trying to match similar instruction sets with different labels. In the third level the raw assembly code instructions are matched against the signature database. We can observe that with each level, the scanning becomes more generalized. Since MEDiC uses disassembled code, the authors in [35] claim that this model can detect malware on any operating system.

3.2 Behavior Based Malware Detection

Behavior-based malware detection as the name suggests extracts the dynamic behavior of the files and studies this behavior to detect malware. Dynamic behavior of a file can be extracted by executing the file in an emulated (sandbox) environment. The emulated environment allows the user to see step by step execution of the program. Many emulators provide a convenient user interface where the user can specify which characteristics of the file to identify and report. The research methodology in [11] combines the features extracted from dynamic analysis with machine learning (classification) techniques for effective and efficient malware detection.

As with all the malware detection techniques, this technique also has training and testing phases. In the training phase, data samples (both malware and benign) are collected and monitored for behavioral patterns. This is done by passing each file through an automatic dynamic analysis tool which executes the file in an emulated environment. As mentioned above, the emulator generates a behavioral report of the input file. These report files are preprocessed for feature selection. Feature selection is the process of selecting only the crucial and applicable characteristics for further processing. These selected features from each file are stored as vectors. Machine learning techniques are applied on these vector models for further learning and classification. The research in [11] compares five different machine learning classifiers namely k-Nearest Neighbor, Naïve Bayes, Support Vector Machine, J48 decision tree and Multilayer Perceptron (MLP) neural network of which J48 gives better performance results.

3.3 Compression Based Malware Detection

Several machine learning techniques are in use for the purpose of identifying unknown malware. Since machine learning techniques are supervised algorithms, they require labeled and organized input data. Despite the fact that machine learning techniques provide promising results, the limiting factor with these techniques is the requirement of labeled input. However the raw data from the executable files is not structured. So we cannot directly apply machine learning algorithms on raw executable files. The data from the files must be preprocessed and converted into desirable format for the algorithm. Preprocessing generally involves feature selection and extraction, labelling the extracted features, etc. Such preprocessing might introduce errors into the data and may not use all the available data for learning. The research methodology in [48] proposes a new malware detection technique that does not require any preprocessing, thereby working on raw executables.

Compression is the process of reducing the size of a file. This is achieved by removing redundancy in a file. The basic logic behind compression is to assign a unique symbol to each string and replace all the occurrences of the string with its corresponding symbol thereby reducing the size of the file considerably. The research in [48] uses an adaptive data compression model namely prediction by partial matching (PPM). The detection mechanism in this paper [48] builds two compression models using the PPM algorithm. One of the compression models represent malware class and the other represents benign class. Each file that needs to be classified, is encoded (compressed) using the two compression models. Whichever model yields the best compression output in terms of compression ratio determines the classification of the file. The preliminary results shown in [48] are promising.

3.4 HMM for Malware Classification

The current research is inspired from the work in [2]. As discussed in Chapter 2, HMM can be used for malware detection. The research in [2] combines HMM and k-means clustering to come up with a malware classification model to classify over 8000 malware samples. Initially several HMM models are trained on a variety of compilers and malware generators (GCC, MinGW, TurboC, Clang, TASM, NGVCK and MWOR). Each malware file is scored against all the trained HMM models. As a result, each of the malware sample is represented by a 7-tuple score. These 7-tuple scores are provided to the k-means clustering algorithm.

In k-means clustering, a dataset of n samples is partitioned into k clusters such that each data point belongs to one and only one cluster which has the nearest centroid. The k-means algorithm involves the following steps:

1. Specify the number of clusters k.
2. Choose initial centroids. The initial centroid estimate can be chosen either at random or uniformly across the dataset.
3. We then associate each data point to the nearest centroid (squared euclidean distance measure is used to compute the distance from each data point to all the centroids).
4. Re-compute the cluster centroids based on current clustering of data points.
5. If there is a significant change in centroids, re-associate the data points to the nearest centroid.
6. The process is repeated until there is no significant change in centroid positions.

Once the samples are clustered, Receiver Operating Characteristics (ROC) curves are plotted to find the accuracy of cluster formations. The clustering algorithm is performed for varied values of k (from 2 to 15) and the results in [2] show that by using k-means clustering on HMM scores, the malware samples can be classified into clusters of appropriate families with good accuracy. But the major drawback of k-means algorithm is that the algorithm does not consider the underlying distribution of the data and the cluster formations depend heavily on the initial choice of the centroids.

To overcome this drawback of k-means algorithm, we use EM clustering in this project. More details about EM algorithm are given in the next chapter.

CHAPTER 4

Clustering and SVM

4.1 Clustering

Clustering is the process of grouping objects together such that related objects fall in same group. Clustering is a common technique used in statistical data analysis, data mining, pattern recognition, etc. This section will briefly discuss various categories of clustering algorithms and then look at EM clustering in detail.

4.1.1 Clustering Techniques

- Intrinsic vs Extrinsic: Intrinsic clustering is an unsupervised learning method and hence is directly applied to the data. In unsupervised learning, we do not have class labels for the data [33]. On the other hand, extrinsic clustering is a supervised learning method. In supervised learning, prior to running the algorithm, data is preprocessed to assign class labels such that each data point belongs to one of the classification classes [33].

- Agglomerative vs Divisive: In agglomerative clustering each data point is considered a cluster in itself [33]. As the algorithm proceeds, the nearest data points merge together forming fewer large clusters. This process continues until all the clusters are merged into a single cluster. Since the algorithm starts with multiple small clusters and works its way up to a single large cluster, it is known as "bottom-up" approach. In contrast, divisive clustering starts with a single cluster consisting of all data points [42] and as the algorithm proceeds, the large cluster is split into multiple mini sized clusters. Hence, divisive clustering can be viewed as a "top-down" approach [33].

- Hierarchical vs Partitional: Hierarchical clustering is in a way related to divisive clustering except for the hierarchical relationship. That is in hierarchical clustering, when a large cluster is split into few small clusters, the large and the small clusters are bound by a parent-child relation respectively. The hierarchical relationship between the clusters can be pictorially represented in the form of a dendrogram [33]. On the other hand, clusters in partitional algorithm do not share any relationship. That is, the data is divided into mutually exclusive clusters such that each data point precisely belongs to one cluster [42].

4.1.2 Expectation Maximization (EM) Clustering

EM clustering is an unsupervised clustering technique. This technique uses gaussian mixture models to find the maximum likelihood estimates of the parameters in the data. Unlike k-means clustering, which uses distance measures to classify the data points into different clusters, EM clustering uses existing probability distributions of the data. That is, instead of assigning a data point exclusively to a single cluster, EM clustering calculates the probability with which a data point belongs to each cluster. The data points are then assigned to the cluster with highest probability. Thus, EM clustering can be viewed as soft clustering technique. As the name suggests, the EM algorithm has two steps, namely E-step and M-step. The algorithm iterates between these two steps to find the maximum likelihood of the parameters of the data [6, 33]. The algorithm continues until the parameters converge or the maximum number of iterations is reached. The steps of the EM algorithm are as follows:

- Initialize the parameters θ and τ, where

  θ_i denotes relevant parameters of distribution i

  τ_i denotes the ratio of times that we sample from the distribution i.

- **E-Step:** Using Bayes' formula, compute the probability of a data point belonging to a particular cluster

$$p_{j,i} = \tau_j f(x_i, \theta_j) \Big/ \sum_{j=1}^{K} \tau_j f(x_i, \theta_j)$$

  Where,

  $j = 1, 2, \ldots, K$ (number of clusters)

  $i = 1, 2, \ldots, n$ (number of data points)

  f is the probability density function

  $p_{j,i}$ is the probability of x_i under distribution j

  We know that the sum of probabilities of a data point belonging to all the clusters is 1. This is mathematically represented as,

$$\sum_{j=1}^{K} p_{j,i} = 1, \tag{4}$$

  for a particular i.

- **M-Step:** Re-estimate the parameters θ and τ

$$\tau_j = \sum_{i=1}^{n} p_{j,i} \Big/ \sum_{j=1}^{K} \sum_{i=1}^{n} p_{j,i}$$

  By using (4), the above equation can be simplified as

$$\tau_j = \frac{1}{n} \sum_{i=1}^{n} p_{j,i}$$

  The estimates for means and variances are given as follows:

$$\mu_j = \sum_{i=1}^{n} p_{j,i} x_i \Big/ \sum_{i=1}^{n} p_{j,i}$$

$$\sigma_j^2 = \sum_{i=1}^{n} p_{j,i} (x_i - \mu_j)^2 \Big/ \sum_{i=1}^{n} p_{j,i}$$

The parameters θ_j will be some function of μ_j and σ_j depending on the probability distribution function.

- Iterate between the E-step and M-step until the parameters θ and τ converge.

Once the model is converged, each data point is assigned to a cluster for which it has the highest probability. This results in each data point belonging to one cluster. Or, we could allow for soft assignment, where each data point belongs to multiple clusters with relative probability. More implementation specific details of EM clustering are given in chapter 5.

4.1.3 Cluster Validation

Different clustering algorithms can form different clusters with the same data. To compare different clustering algorithms or to quantify how well a particular algorithm clusters the data, we should be able to measure individual cluster quality. In this section we will discuss some measures of cluster quality. There are two general approaches to cluster validation, namely, external and internal. We discuss these two approaches in detail in the coming sections.

Let $x_1, x_2, \ldots, x_m$ denote the data points and $C_1, C_2, \ldots, C_K$ denote the clusters. Let m_j be the total count of objects in cluster C_j and m_{ij} the number of objects of type i in cluster C_j. The probability of having type i objects in cluster j is given by $p_{ij} = m_{ij}/m_j$. We follow the same notation to discuss the validation techniques below.

4.1.3.1 External Validation

External validation uses metadata about the data to determine the cluster quality. This metadata often corresponds to class labels attached to the data. There are two common measures in external validation, entropy and purity. Both these measures determine the quality of a cluster by counting the number of objects of a particular class assigned to a cluster.

Entropy is a standard measure of instability [14, 33, 42]. Higher entropy implies that the cluster is a mix of objects from multiple classes and no particular class objects dominate the cluster. Following the notation mentioned earlier, entropy of a cluster C_j is given by

$$E_j = -\sum_{i=1}^{n} p_{ij} log p_{ij}$$

The (weighted) intra-cluster entropy is given by

$$E = \frac{1}{m}\sum_{i=1}^{K} m_j E_j$$

We would want E to be smaller, since small E indicates maximum objects of the cluster belong to a single class.

Purity is the measure of uniformity within the cluster [14, 33, 42]. An ideal cluster should have all the data points of a particular type. Using the same notation as above, purity of a cluster C_j is defined as

$$U_j = max_i P_{ij}$$

The overall (weighted) purity is given by

$$U = \frac{1}{m}\sum_{i=1}^{K} m_j U_j$$

We would want U to be as close to 1 as possible because, smaller U indicates that the clusters elements share no relationship with each other.

4.1.3.2 Internal Validation

Internal validation determines quality based on cluster characteristics. Naturally, we want the clusters to be well separated from each other. Also, the data points with in a cluster should be as close as possible to each other. These properties are referred to as separation and cohesion respectively. These two properties can be consolidated to get a solitary quality measure called silhouette coefficient [14, 33, 42]. Using the same notation as above, silhouette coefficient of a data point x_i is given by

$$S(x_i) = \frac{b - a}{\max(a, b)}$$

Where,

a is the average distance of x_i to the data points in its cluster

b is the minimum (average distance of x_i to data points in another cluster)

Figure 2 illustrates an example of silhouette coefficient calculation. In general for any reasonable cluster formation, $b > a$, therefore

$$S(x_i) = 1 - \frac{b}{a}$$

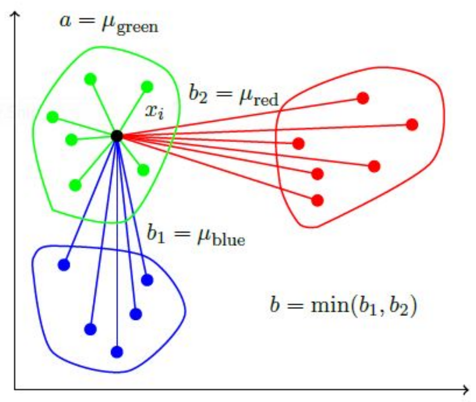

Figure 2: Silhouette Coefficient Example

We would want $S(x_i) \approx 1$ i.e., we want x_i to be cohesive with respect to the data points in its own cluster and well separated from the data points in other clusters.

In this project we combine purity and silhouette coefficient measures to come up with a score which we use to plot the ROC curves. Implementation details of this score are provided in Chapter 5.

4.2 Support Vector Machines (SVM)

Support Vector Machines are amongst the most popular supervised learning algorithms used for classification and regression analysis. Supervised learning method deals with labeled data i.e., we must preprocess the data and assign labels to each data point. SVMs are generally applied to binary classification problems. Given a labeled training data, the algorithm tries to find an optimal hyperplane such that all the data points on one side of hyperplane belong to one class and the data points on other side belong to another class. In other words the hyperplane acts a threshold separating the two classes of data [36]. Figure 3 illustrates an optimal hyperplane separating the red squares from blue circles [24].

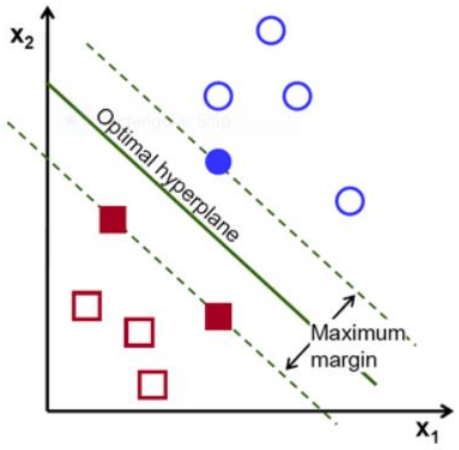

Figure 3: Optimal Hyperplane Example

The hyperplane should be chosen in such a way that it maximizes the distance between closest data points between the two classes. In other words the hyperplane should maximize the margin as shown in Figure 3. But for most of the real world applications, training data is not linearly separable. There are two ways to deal with this situation. Instead of maximum margin, specify a “soft” margin or use the “kernel trick”. By using soft margin, we allow the SVM to make some classification errors. Depending on the non linearity of the data and the soft margin, the number of classification errors may vary significantly. Kernel trick provides an effective solution to both the problems, non linearity and classification errors.

Kernel trick is the process of transforming non linearly separable data into linearly separable data. This is done by shifting the data into a higher dimensional space [33]. Generally, when dealing with machine learning algorithms, it is advised to reduce the dimensionality of the data so as to avoid the ‘curse of dimensionality’. However the kernel trick does the magic in SVM to work efficiently with high dimensional data. The kernel function spreads out the data in high dimensional space making it linearly separable by a high dimensional (one dimension less than the data) hyperplane. Figure 4 illustrates the kernel trick of transforming 2D data into 3D space [19]. Implementation details of SVM are provided in Chapter 5.

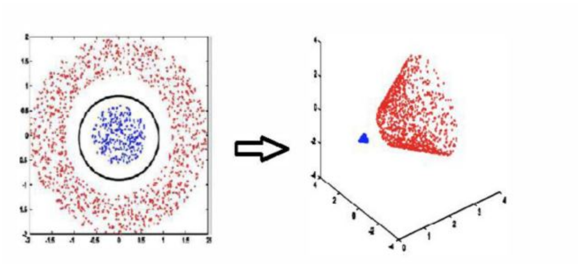

Figure 4: Transformation of Data to Higher Dimensional Space

CHAPTER 5

Implementation and Experimental Setup

This section provides the implementation and experimental setup details of the project. The first section gives a brief overview of the dataset. The second section gives details about training the HMMs and scoring the dataset. The third section explains the tools and packages used for clustering and the scoring technique used to plot the ROC curves. The fourth section explains the tools used for SVM. Finally, experimental setup details are discussed.

5.1 Dataset

The malware dataset used for this project is obtained from the Malicia project website [22]. The dataset consists of more than 11,000 malware binaries. All the malware files are either an executable (.exe) or a dynamic-link library (.dll). Along with the malware samples, the dataset also consisted of MySQL database with metadata. For each malware sample, the metadata has information of the family of malware and details about when and where the malware is collected. This metadata information helped us in calculating the cluster purity. Of the 11,000 malware samples, about 7800 samples were distributed among three dominant malware families namely Winwebsec, Zbot and ZeroAccess.

Winwebsec is a malware program that runs only on windows operating system. This malware convinces windows users into buying fake anti-virus software. As an initial step, the malware gives false alerts to the user that the system has been infected even though the system does not contain any other malware. As a commercial step, the malware offers to remove theses infections for a fee [18].

Zbot, also known as Zeus is a family of trojans [38]. Zbot is also targeted to run on windows operating system. This malware steals personal and financial information from the compromised system. Zbot files are usually installed via spam emails and hacked websites [17].

ZeroAccess is another family of trojans that uses advance rootkit to hide itself. ZeroAccess is also targeted at windows users and its major functionality is to download other harmful malware [39].

In this project we only used the malware samples from these three families. The benign samples used for this project are 32-bit Cygwin utility files. Table 2 shows the number of samples used from each family.

All of the malware and benign files are disassembled using the IDA disassembler [12]. Opcode sequences are then extracted from the disassembled files and given as input to the HMM models for scoring.

Table 2: Dataset Distribution

Family	Number of files
Winwebsec	4361
Zbot	2136
ZeroAccess	1306
Benign	213

5.2 Training the HMMs

In this project we perform multiple experiments . All the experiments follow the same steps except that the HMM models are generated with different data. In all the experiments, the HMMs are trained on sequence of opcodes. In order to get the opcodes, the malware and benign binaries are first disassembled using the IDA disassembler [12]. Opcodes are then extracted from the corresponding assembly files and used to train the HMMs. Now we will see in detail how to train the HMMs for our experiments.

5.2.1 Training with compilers and malware generators

Initially, the HMMs are generated for four different compilers (GCC, MinGW, TurboC, Clang), hand-written assembly (TASM), virus generation kit(NGVCK) and metamorphic malware (MWOR). For each of these models, the training is done for 800 iterations with number of hidden states being 2. The number of assembly files used for training each model is given in Table 3.

Table 3: Number of files used for training HMMs

Type	Number of files
MWOR	100
GCC	75
MinGW	72
Clang	72
TurboC	64
TASM	56
NGVCK	50

Each of the malware and benign files mentioned in Table 2 are scored against these seven HMM models. In order to score a file, the file is disassembled and opcode sequence is extracted. The opcode sequence is given to the trained HMM model. The model assigns high score to the files similar to the training data set and low score to other files. The score is nothing but log likelihood of the opcode sequence and is highly dependent on the length of the sequence. Since different files have different opcode sequence lengths, we divide the log likelihood of the sequence by the number of opcodes (length of the sequence). This normalizes the score for various opcode sequence lengths and we obtain log likelihood per opcode. And finally, each file is now represented with a 7-tuple score.

5.2.2 Training with subset of malware samples

For this experiment three HMM models are generated, one for each of the dominant malware families. The malware files from the remaining two families and the benign files are scored against these HMM models. For example, the Winwebsec HMM model is generated as follows:

- Train HMM on a subset of Winwebsec files.
- Score the remaining Winwebsec files (that are not part of training subset), Zbot , ZeroAccess and benign files against the model created in above step.

Similarly generate the HMM models for Zbot and ZeroAccess families. The models should assign high score to the files that are from the same family as training dataset and low score to other families.

Similar to the scores in Section 5.2.1, the final scores obtained in this section are also normalized to obtain the log likelihood per opcode. Each file is now represented with a 3-tuple/3-D score.

5.3 Clustering Implementation

In this project we used the Matlab Statistics Toolbox [20] to implement the clustering algorithms. The Statistics toolbox provides easy to use built-in functions for both *k*-means and EM clustering algorithms.

5.3.1 Clustering scores from compiler trained HMM models

The 7-tuple scores obtained from the HMM are used for clustering. A subset of the 7-tuple HMM scores is used as training dataset to generate a cluster model. The remaining scores are given to this cluster model to see how well the model fits these scores into different clusters. The number of samples used for training and testing are given in Table 4. Based on the resulting cluster formations we calculate a probability score for each sample that is clustered and use this probability score as a performance measure to compare k-means and EM algorithms. The probability score is calculated as follows:

1. Generate a clustering model using the train dataset.
2. Using the trained model, cluster the test data.
3. Calculate cluster purity for the clusters created in Step 2.
4. For each test sample, depending on the cluster to which it is assigned and the corresponding cluster purity, compute the probability with which the sample belongs to each of the malware/benign (Winwebsec, Zbot, ZeroAccess, benign) families. That is each test sample will have four probabilities associated with it.
5. From the metadata in the malware dataset, we know the family to which each test sample actually belongs to. Now for each test sample, take the probability that is associated with its own family.
6. Take the average of probability scores obtained in Step 5.

We repeat the experiment by varying the number of clusters, K from 2 to 15. Perform the same experiment for both EM and k-means algorithms.

Table 4: Distribution of Samples Used for Clustering

	Number of files	
Family	Training	Testing
Winwebsec	3000	1361
Zbot	1500	636
Zeroaccess	1000	306
Benign	0	213

5.3.2 Clustering scores from malware trained HMM models

In this experiment, the scores from HMM models trained on malware samples are used. We train the clustering algorithm on two malware families and generate a cluster model. We then try to classify the third malware family and benign files using the trained model. An example clustering model is trained and tested as follows:

- Prepare a 2-D train dataset such that column 1 corresponds to Winwebsec and Zbot scores from Winwebsec trained HMM model and column 2 corresponds to Winwebsec and Zbot scores from Zbot trained HMM model.
- Prepare a 2-D test dataset such that column 1 corresponds to benign and ZeroAccess scores from Winwebsec trained HMM model and column 2 corresponds to benign and ZeroAccess scores from Zbot trained HMM model.
- Generate a clustering model using the train dataset.
- Fit the test dataset with the model generated in the above step.

Next, we consider a score based on these clusters to plot the Receiver Operating Characteristic (ROC) [10] curves. the ROC curve is obtained by plotting true positive rate against false positive rate, varying the threshold through the range of data values. The region under the ROC curve (AUC) provides a convenient measure of the quality of a binary classifier [8]. An AUC of 1.0 shows perfect classification (i.e., we can draw a threshold that will separate malware files from benign files with out any misclassification), while an AUC of 0.5 indicates that the that the classifier is no more fruitful than flipping a coin.

The basis for our score is the silhouette coefficient, as discussed in Section 4.1.3.2. Given a set of clusters, where x is an element of one, say, cluster C_j, to compute the score of x, we first calculate $S(x)$, the silhouette coefficient of x. Next, as discussed in Section 4.1.3, we calculate $p_{ij} = m_{ij}/m_j$, where m_{ij} is the number of elements of family i in cluster C_j and m_j is the total number of elements in cluster C_j. Finally, the scoring function is defined as

$$\text{score}_i(x) = p_{ij}S(x)$$

where $\text{score}_i(x)$ is the score of x with respect to family i.

Since our test dataset has two different samples (ZeroAccess and benign), each test sample file will have two scores.

$$\text{score}_0(x) = \text{benign score of file } x$$

and

$$\text{score}_1(x) = \text{ZeroAccess score of file } x$$

We use $\text{score}_i(x)$ to plot the ROC curve.

Once the clustering algorithm is run on the 2-D data, we extend this experiment to include more dimensions to the train and test datasets. More details about this extended experiments are given in the Chapter 6.

5.4 SVM Implementation

In this project we used the RapidMiner [29] to run the SVM algorithm. RapidMiner is a software platform that provides an integrated environment for machine learning, predictive analysis, data mining etc. RapidMiner provides an easy to use web interface with several drag and drop tools to construct an SVM model. Since SVM is a supervised learning algorithm, we should pre-process the data to include the labels. We used the data labels 1 and -1 to represent the two classes of samples (malware and benign). In SVM, we train an SVM model on different malware families and verify how well the trained model can classify new samples from the same malware families. We use the same HMM scores that we used for clustering to train and test the SVM models.

5.5 Experimental Setup

In this project, since we are dealing with malware samples, most of the experiments are run on a virtual machine. We used the Hyper-V virtual machine manager to configure and manage the virtual machines. Following are the specifications of host and virtual machine instances.

Host:

- Model: Lenovo Ideapad Yoga
- Processor:Intel Core i5-3337U, CPU @ 1.80GHz
- RAM: 8.00 GB
- System Type: 64-bit
- Operating System: Windows 8.1

Guest 1:

- Software: Hyper-V Manager
- Start-up Memory:1024 MB
- System Type: 64-bit
- Operating System: Windows 8.1

Guest 2:

- Software: Hyper-V Manager
- Start-up Memory:2048 MB
- System Type: 64-bit
- Operating System: Ubuntu 14.10

As mentioned in Section 5.2, the HMMs are trained on opcode sequences. Extraction of opcode sequences from the malware and benign samples is done in Guest 2 system. Training the HMM models, clustering algorithms and SVM are performed in Guest 1 system.

CHAPTER 6

Results

This chapter provides a detailed description of the results. The first section discusses the results of EM and k-means clustering algorithms. The second section discusses the results of SVM algorithm. Finally we compare the results of clustering and SVM to see which technique gives better results.

6.1 Clustering

In this section we present and discuss the results of EM and k-means clustering algorithms. Since clustering is an unsupervised learning technique, no pre-processing is required on the data. Hence the HMM scores are directly given as input to EM and k-means algorithms.

6.1.1 Clustering results of compiler trained HMM scores

For the first experiment we consider the 7-tuple HMM scores for clustering and compute the probability score. As mentioned in Chapter 5, the probability score is computed using the metadata obtained along with the malware dataset. The clustering algorithm is run multiple times for the same K and the maximum probability score of all the runs is considered for that particular K. Figure 5 shows the probability score comparison of EM and k-means algorithms. The x-axis represents the number of clusters, K and y-axis represents the probability score. The red line corresponds to the probability scores of EM algorithm for $K = 2$ to $K = 15$. Similarly, the magenta line corresponds to the probability scores of k-means algorithm. The probability scores with which Figure 5 is plotted are given in Table 5.

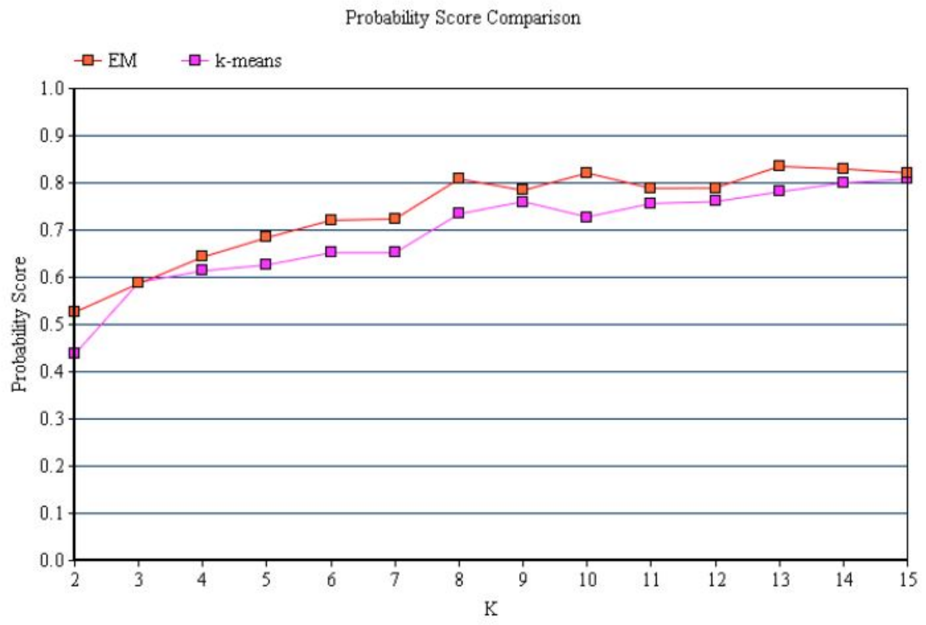

Figure 5: Probability Score Comparison

It can be seen from Figure 5 that the probability score of EM algorithm is slightly better than k-means. That is we can say that EM clustering algorithm classifies malware samples more accurately than k-means algorithm. Also, we can observe that with increase in K (number of clusters), there is a slight increasing trend in the probability score. This is because, as the number of clusters is more, data is grouped into more number of small but comparatively pure clusters.

6.1.2 Clustering results of malware trained HMM scores

For the initial experiment with malware trained HMM scores, we start with 2-D HMM scores (scores from two of the HMM models trained in Section 5.2.2). We perform both EM and k-means clustering algorithms on the same data with $K = 2$, compute the scores as metioned in Section 5.3.2 and plot the ROC curves. Figure 6 shows the ROC curve for EM clustering algorithm for 2-D data.

Table 5: Probability Scores of EM and k-means

K	EM	k-means
2	0.5261	0.4372
3	0.5868	0.5894
4	0.6430	0.6138
5	0.6839	0.6261
6	0.7208	0.6520
7	0.7236	0.6876
8	0.8084	0.7343
9	0.7843	0.7602
10	0.8209	0.7279
11	0.7886	0.7563
12	0.7893	0.7603
13	0.8351	0.7811
14	0.8296	0.8000
15	0.8213	0.8080

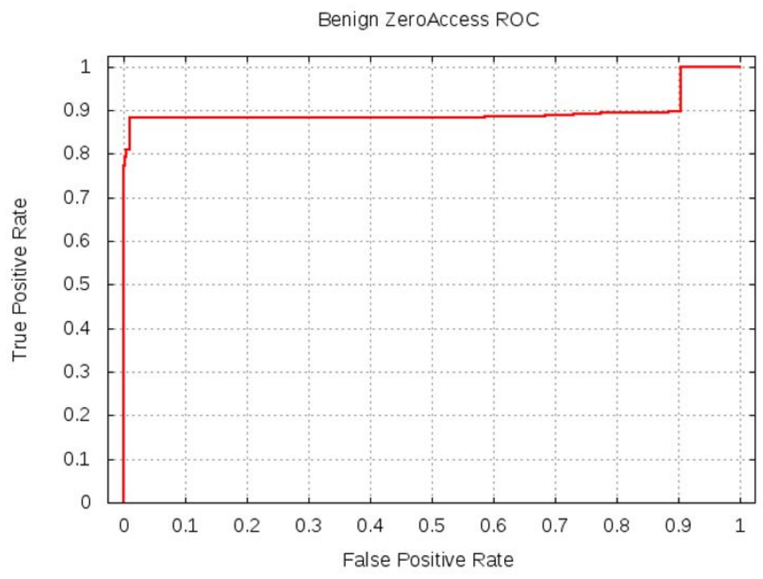

Figure 6: ROC curve for EM clustering algorithm (2-D data)

The corresponding AUC for the curve is 0.896 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 89.6% . The cluster formations of the ZeroAccess and benign samples using this model are given in Figure 7, where x-axis represents the HMM scores of Winwebsec model and y-axis represents the HMM scores of Zbot model.

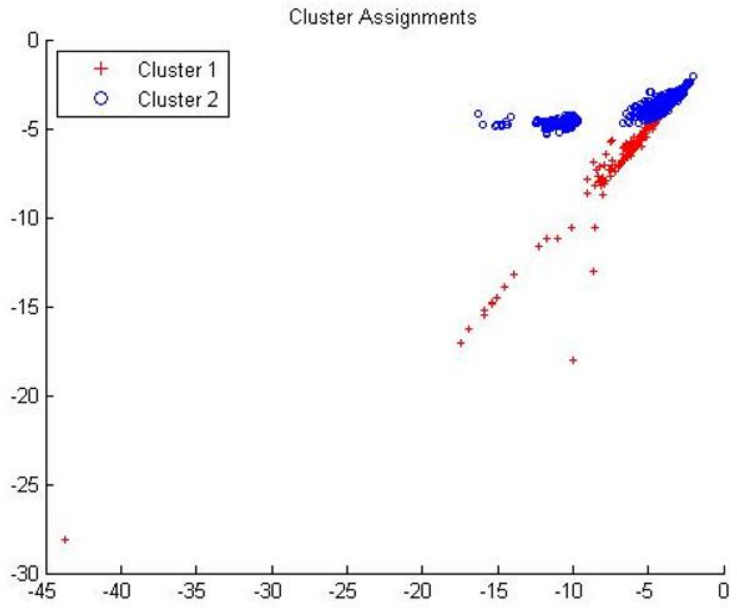

Figure 7: Cluster formation using EM clustering algorithm (2-D data)

Figure 8 shows the ROC curve for k-means clustering algorithm for 2-D data. The corresponding AUC for the curve is 0.81 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 81% . The clusters formations for the same data using k-means clustering model are given in Figure 9.

From Figures 7 and 9, we can observe that the same data samples have been put into different clusters by different clustering algorithms. Based on these different cluster formations, the two clustering models have different levels of accuracy in classifying ZeroAccess and benign samples.

For the next experiment we consider 3-D HMM scores (scores from all the three HMM models trained in Section 5.2.2). Similar to the 2-D data, we perform both EM

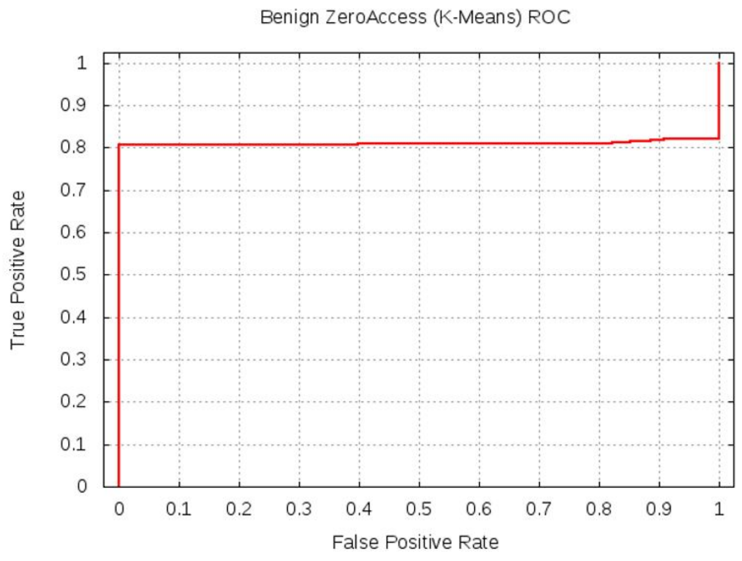

Figure 8: ROC curve for k-means clustering algorithm (2-D data)

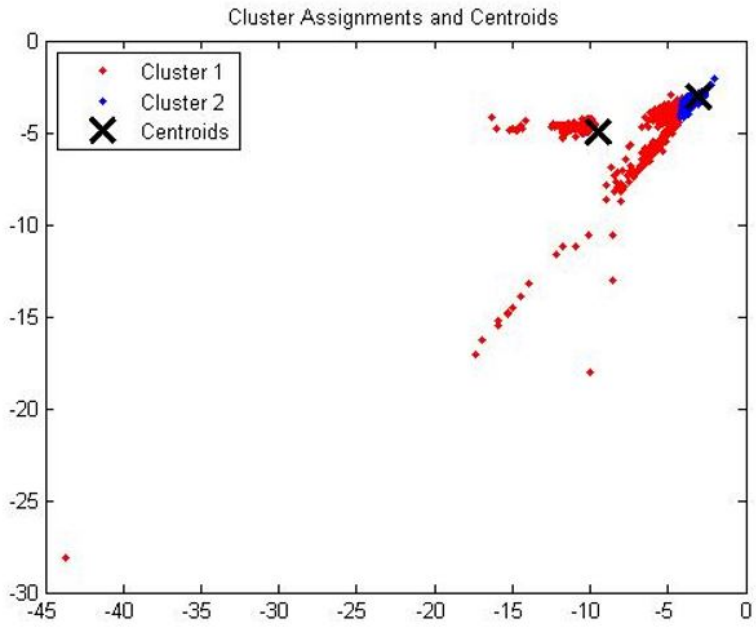

Figure 9: Cluster formation using k-means clustering algorithm (2-D data)

and k-means algorithms on 3-D data for $K = 2$, compute the scores and plot ROC curves.

Figure 10 shows the ROC curve for EM clustering algorithm for 3-D data. The

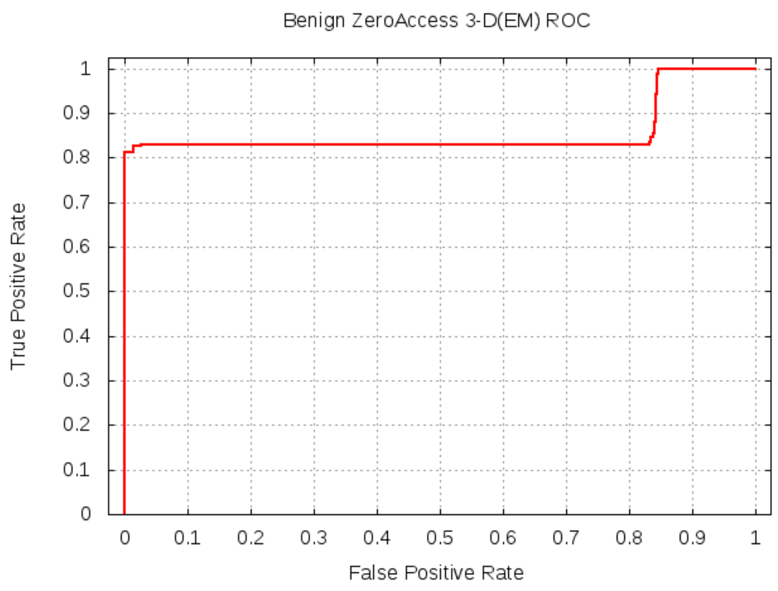

Figure 10: ROC curve for EM clustering algorithm (3-D data)

corresponding AUC for the curve is 0.856 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 85.6% .

Figure 11 shows the ROC curve for k-means clustering algorithm for 3-D data. The corresponding AUC for the curve is 0.852 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 85.2% .

Next we consider 4-D HMM scores. The 4-D data is obtained by adding the scores of NGVCK trained HMM model to the existing 3-D dataset. We perform similar experiments on the 4-D data and plot the ROC curves.

Figure 12 shows the ROC curve for EM clustering algorithm for 4-D data. The corresponding AUC for the curve is 0.94 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 94% .

Figure 13 shows the ROC curve for k-means clustering algorithm for 4-D data.

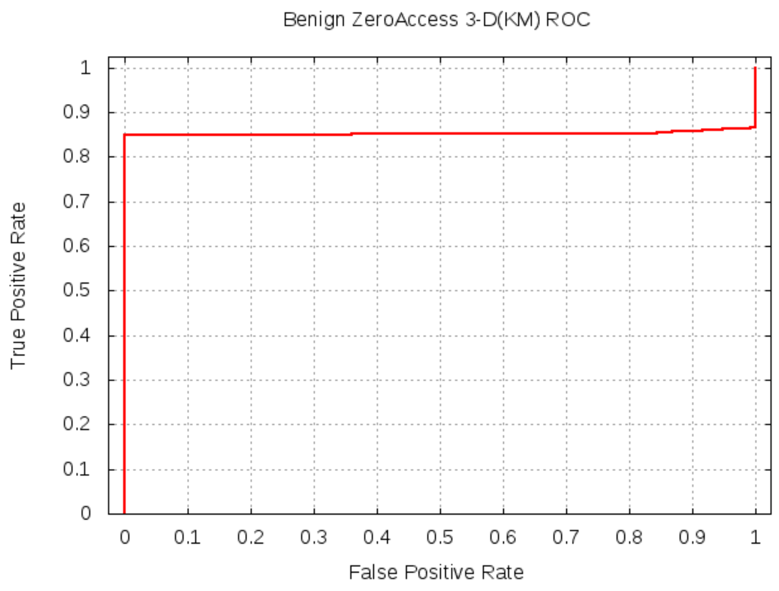

Figure 11: ROC curve for k-means clustering algorithm (3-D data)

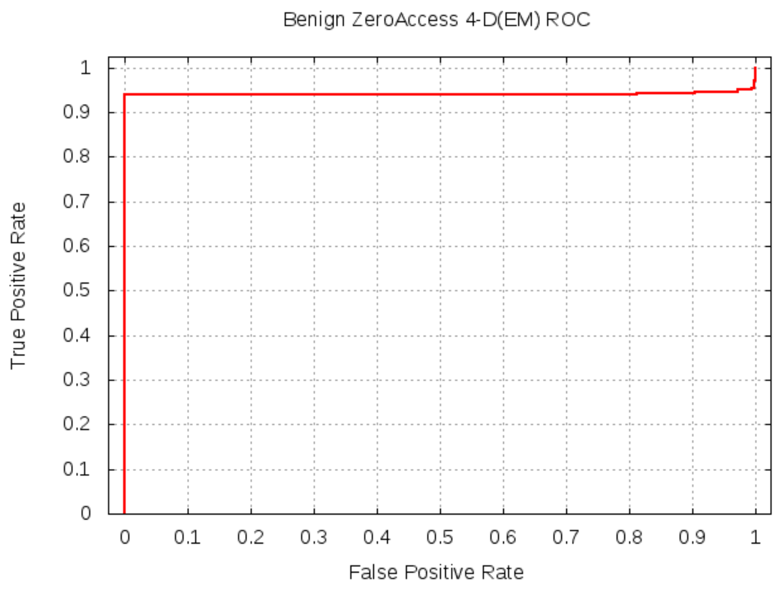

Figure 12: ROC curve for EM clustering algorithm (4-D data)

The corresponding AUC for the curve is 0.977 which implies that this clustering model can differentiate between ZeroAccess and benign files with a probability of 97.7% .

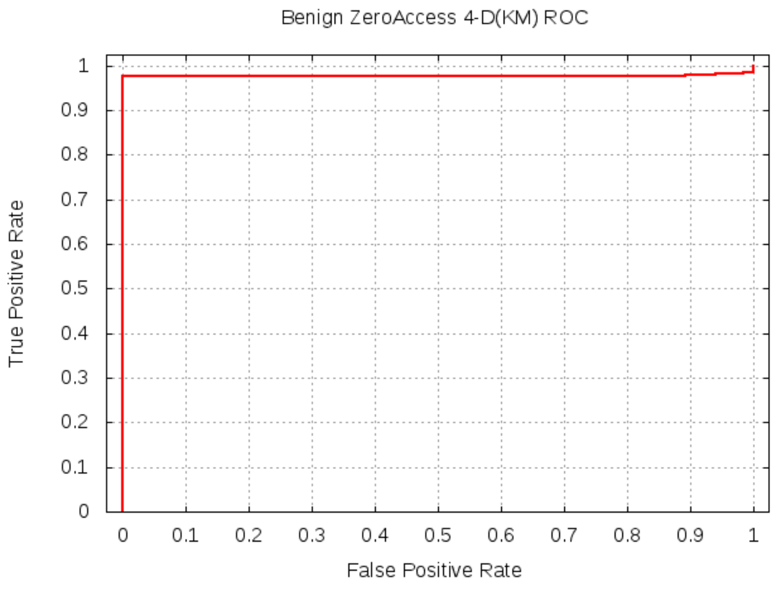

Figure 13: ROC curve for k-means clustering algorithm (4-D data)

6.1.3 Comparison of clustering algorithms based on accuracy

In this section we compare the performance of EM and k-means clustering algorithms based on accuracy measure. Accuracy gives the percentage of datapoints that are correctly clustered into their specific category. Since our test dataset contains two families (ZeroAccess and Benign), accuracy gives the percentage of datapoints that are correctly classified as malware or benign. The parameters used to calculate accuracy are,

- TP (True Positive), is the number of malware samples correctly classified as malware.
- TN (True Negative), is the number of benign samples correctly classified as benign.
- FP (False Positive), is the number of benign samples incorrectly classified as malware.

- FN (False Negative), is the number of malware samples incorrectly classified as benign.

Accuracy is calculated using the formula,

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{P + N}$$

Where,

$$P = \text{TP} + \text{FN} \text{ and } N = \text{TN} + \text{FP}$$

In simple terms accuracy is given as,

$$\text{accuracy} = \frac{\text{Number of correctly classified datapoints}}{\text{Total number of datapoints}}$$

In an ideal case, where all the datapoints are correctly classified, we get an accuracy of 1.0. In general we want accuracy to be as close to 1.0 as possible. In this experiment we calculate the accuracy of both EM and k-means algorithms by varying the value of K from 2 to 15. Also, we perform this experiment on dataset ranging from 2-D to 7-D.

Figure 14, shows the accuracy measure of EM algorithm in which X-axis represents the data dimensions varying from 2-D to 7-D, Y-axis represents the number of clusters, K varying from 2 to 15 and Z-axis represents the accuracy value. From Figure 14, we observe that the accuracy value in all the cases is in the range of 0.8 to 1.0.

Similarly, Figure 15, shows the accuracy measure of k-means algorithm. For k-Means algorithm also we see that the accuracy values are in the range of 0.8 to 1.0.

For better comparison of the accuracy values of EM and k-means algorithms, we overlap the Figures 14 and 15 to obtain 16. The stem plot in Figure 16, shows

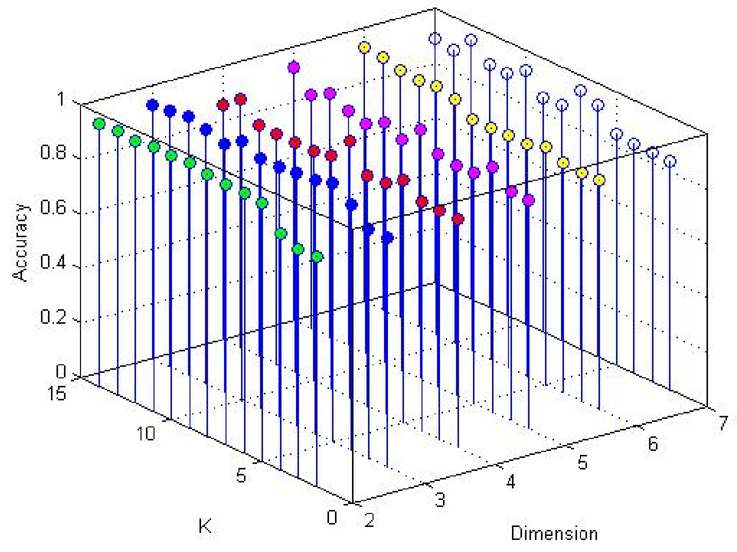

Figure 14: Accuracy measure for EM algorithm

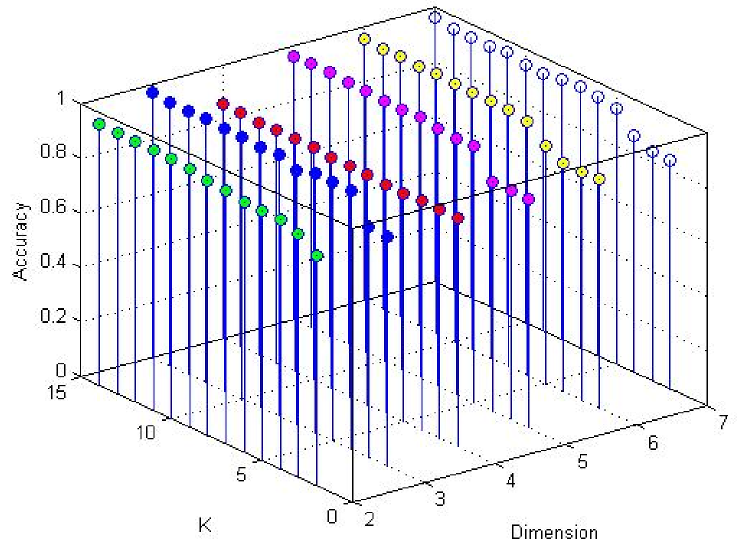

Figure 15: Accuracy measure for k-means algorithm

accuracy values of both EM and k-means algorithms in the same plot where the 'o' corresponds to the accuracy value of EM algorithm and '*' corresponds to the accuracy value of k-means algorithm. From Figure 16, we observe that with increase in K, the accuracy value of k-means algorithm is slightly greater than the EM algorithm. The main reason behind this behavior is, since Em clustering is based on the distributions in the data, eventhough we increase the value of K, after a certain point the algorithm created only few data clusters and more empty clusters. On the otherhand, k-means algorithm is based on distance measure, as a result with increase in K the algorithm created more small and pure clusters thereby giving better accuracy.

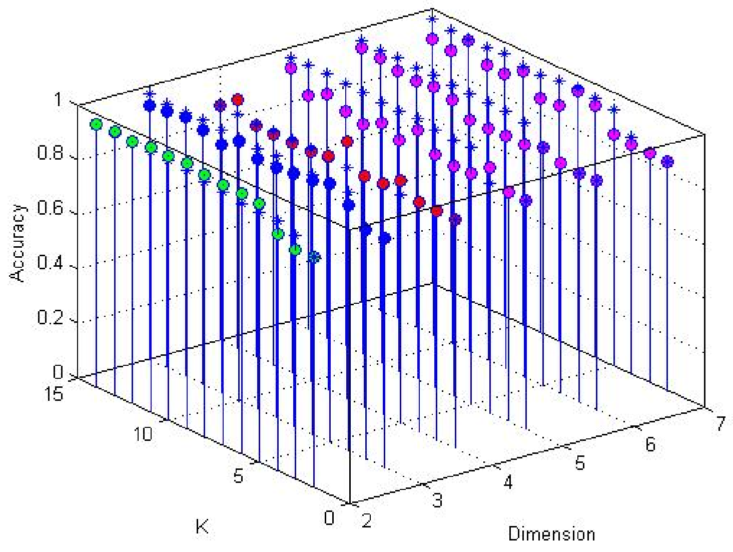

Figure 16: Accuracy comparison of EM and k-means algorithms

6.2 SVM

To perform the SVM experiments we need to first pre-process the data by assigning data labels such that malware scores are labeled as 1 and benign scores are labeled as -1 or vice-versa.

6.2.1 SVM results of compiler trained HMM scores

Similar to the clustering experiments we initially take the 7-tuple data and perform SVM. We use the same train and test datasets that we used for clustering. The only difference is that training dataset for clustering algorithm do not contain benign samples whereas benign samples are included in both train and test datasets of SVM. Figure 17 shows the ROC curve of SVM algorithm for 7-tuple using radial kernel function.

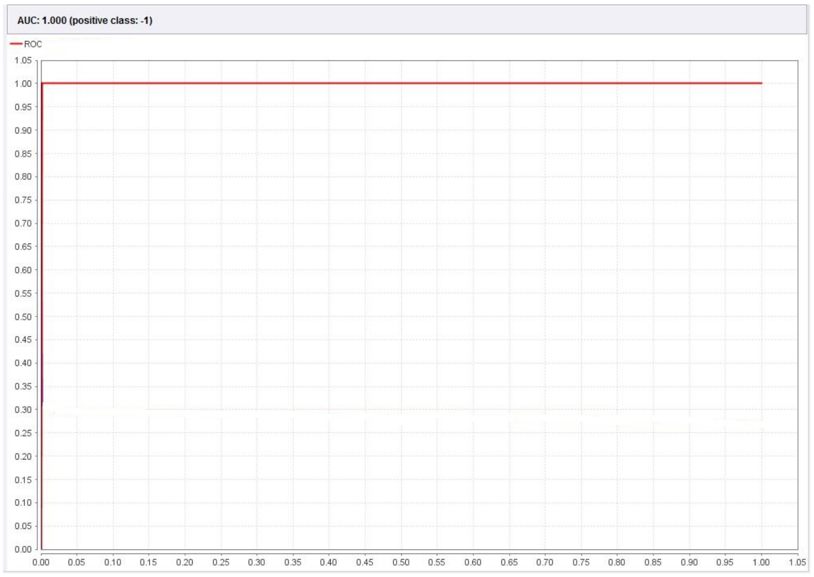

Figure 17: ROC curve for SVM algorithm (7-tuple data)

The corresponding AUC for the curve is 1.0 which implies that this SVM model can differentiate between malware and benign files with 100% accuracy. Also, we performed this experiment by varying the kernel functions. The kernel function and the corresponding AUC are given in Table 6.

From the AUC values in Table 6, we can observe that most kernel functions give 100% accurate SVM binary classifier.

Table 6: AUC for different kernel functions (7-tuple data)

Kernel Function	AUC
Dot	1
Anova	1
Neural	0.82
Polynomial	1
Epachnenikov	1

6.2.2 SVM results of malware trained HMM scores

For this experiment, initially we generate an SVM model to test the 2-D data. In this experiment we train an SVM model on a subset of Zbot and Winwebsec samples and we test the model on remaining Zbot and Winwebsec samples. We pre-process the data to label Zbot samples as 1 and Winwebsec samples as -1. Figure 18 shows the ROC curve of SVM algorithm for 2-D data using radial kernel function.

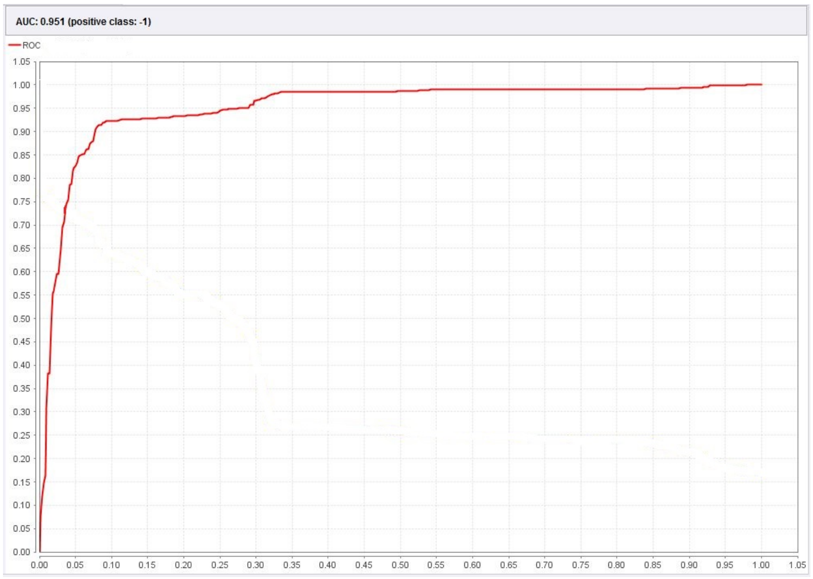

Figure 18: ROC curve for SVM algorithm (2-D data)

The corresponding AUC for the curve is 0.95 which implies that this SVM model can differentiate between the two malware families with an accuracy of 95% . Also, we performed this experiment by varying the kernel functions. The kernel function and the corresponding AUC are given in Table 7.

Table 7: AUC for different kernel functions (2-D data)

Kernel Function	AUC
Dot	0.83
Anova	0.95
Neural	0.65
Polynomial	0.72
Epachnenikov	0.83

Next, we test the SVM model on 3-D data by training an SVM binary classifier on a subset Zbot and Winwebsec scores from the three HMM models trained in Section 5.2.2 and testing the classifier on the remaining subset of Zbot and Winwebsec scores. Figure 19 shows the ROC curve of SVM algorithm for 3-D data using radial kernel function.

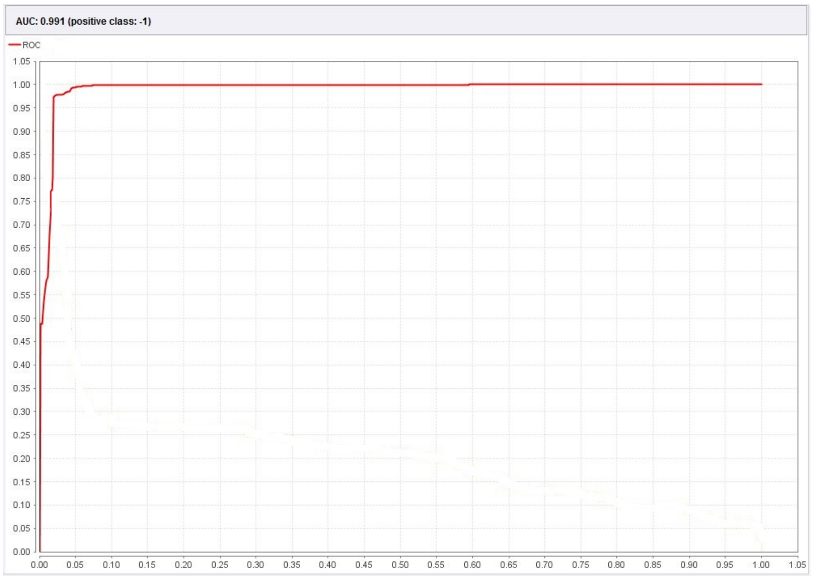

Figure 19: ROC curve for SVM algorithm (3-D data)

The corresponding AUC for the curve is 0.991 which implies that this SVM model can differentiate between the two malware families with an accuracy of 99.1% . Also, we performed this experiment by varying the kernel functions.

Various kernel functions and their corresponding AUC are given in Table 8.

Table 8: AUC for different kernel functions (3-D data)

Kernel Function	AUC
Dot	0.98
Anova	0.99
Neural	0.80
Polynomial	0.85
Epachnenikov	0.98

Next, we test the SVM model on 4-D data (we add the NGVCK trained HMM model scores to the 3-D dataset in the above experiment to get the 4-D data). Figure 20 shows the ROC curve of SVM algorithm for 4-D data using radial kernel function.

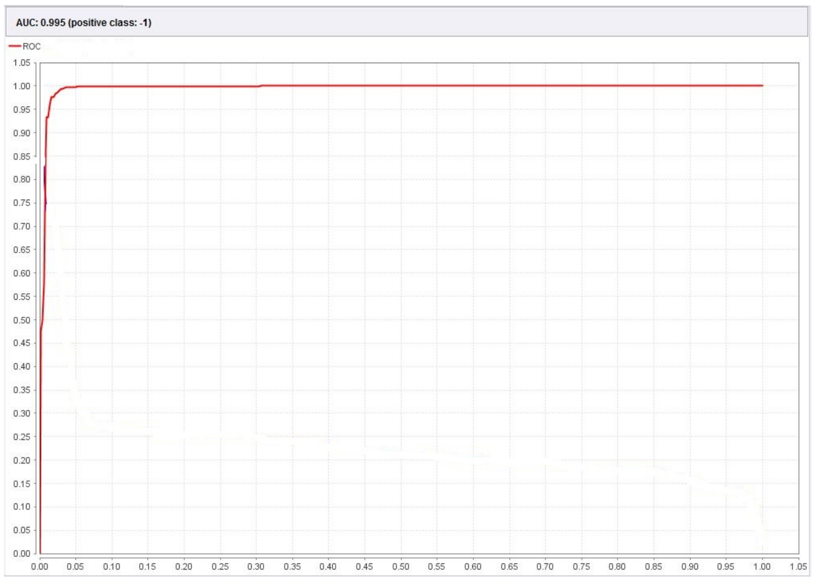

Figure 20: ROC curve for SVM algorithm (4-D data)

The corresponding AUC for the curve is 0.995 which implies that this SVM model differentiates between the two malware families with an accuracy of 99.5% . Also, we performed this experiment by varying the kernel functions. The kernel function and the corresponding AUC are given in Table 9.

Table 9: AUC for different kernel functions (4-D data)

Kernel Function	AUC
Dot	0.98
Anova	0.99
Neural	0.83
Polynomial	0.83
Epachnenikov	0.99

The results of the clustering and SVM algorithms for malware classification can be summarized as shown in Table 10. From Table 10, we can see that as the number of dimensions in the data increase, there is a gradual increase in the AUC values for the three techniques that we used in this project. From this we can say that cluster and SVM models trained on high dimensional data yield better classification results.

Table 10: Comparison of AUC

		AUC of	
Data	EM	k-means	SVM
2-D	0.896	0.81	0.95
3-D	0.856	0.852	0.991
4-D	0.94	0.977	0.995

From the above results of clustering and SVM, we can observe that the clustering algorithms performed comparable to SVM in classifying different malware (and benign) families. We can also say that when we are dealing with the problem of classifying malware samples (on which the SVM model has been trained), SVM gives better accuracy rate. On the other hand when we are dealing with unknown samples, clustering seems to give better results by placing the unknown samples in a different cluster. Moreover the advantage with clustering is that once we train a model, we can use that model to classify new malware without having to re-train the model.

CHAPTER 7

Conclusion and Future Work

In this paper, we compared the accuracy of two clustering algorithms namely EM and k-means for classifying over 8000 malware samples. We carried out different experiments for this purpose. In the first experiment, malware and benign samples are scored against a variety of compilers. That is we train different HMM models on a variety of compilers and the malware samples are scored against these trained HMM models. In the second experiment, we score the malware samples against each other. That is we train a HMM model on a particular malware family and score the remaining malware and benign samples against this trained HMM model. In both the cases, the HMM scores are used as input to the clustering and SVM algorithms. From the experiment results we observe that EM clustering algorithm performs better than the k-means algorithm for the data we considered.

In this project, we used opcode sequences as input for training the HMM models and scoring the malware samples. Other variations of input such as call-graph sequences instead of opcode sequences can be used for future experiments.

The initial parameters in the clustering algorithms, centroids in case of k-means and mean and co-variance in case of EM algorithms are chosen at random. Instead of random initialization, centroids of k-means algorithm can be selected from within the data domain. Similarly mean and co-variance of EM algorithm can be selected based on the actual data distribution.

The dataset we used for this project has only three dominant malware families. The clustering model is trained on two families and tested on the third malware family and benign samples. For the future work we can consider adding more malware families to the dataset and we expect to get better results with more malware families.

LIST OF REFERENCES

[1] K. Alsabti, S. Ranka and V. Singh (1998), An Efficient K-Means Clustering Algorithm, https://www.cs.utexas.edu/~kuipers/readings/Alsabti-hpdm-98.pdf

[2] C. Annachhatre, T. H. Austin, M. Stamp (2014), Hidden Markov Models for Malware Classification, *Journal of Computer Virology and Hacking Techniques*, http://link.springer.com/article/10.1007/s11416-014-0215-x

[3] T. Austin, E. Filiol, S. Josse, and M. Stamp (2013), Exploring Hidden Markov Models for Virus Analysis: A Semantic Approach, *46th Hawaii International Conference on System Sciences (HICSS 2013)*, 5039–5048

[4] J. Aycock (2006), *Computer Viruses and Malware*, Springer

[5] P. Beaucamps (2007), Advanced Polymorphic Techniques,*International Journal of Computer Science*, 2(3):194–205, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.4560&rep=rep1&type=pdf

[6] J. A. Bilmes (1998), A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, http://crow.ee.washington.edu/people/bulyko/papers/em.pdf

[7] Boot-Sector Viruses (n.d.), http://antivirus.about.com/cs/tutorials/a/bsvirus.htm

[8] A. P. Bradley (1997), The use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms, *Journal of Pattern Recognition*, 30(7):1145–1159

[9] C. B. Do and S. Batzoglou (2008), What is the Expectation Maximization Algorithm?, *Nature Biotechnology*, 26(8):897–899

[10] T. Fawcett (2005), An Introduction to ROC Analysis, http://people.inf.elte.hu/kiss/13dwhdm/roc.pdf

[11] I. Firdausi, C. Lim, A. Erwin, A. S. Nugroho (2010), Analysis of Machine Learning Techniques Used in Behaviour-Based Malware Detection, *Second International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)*, 201–203

[12] IDA Disassembler (n.d.), https://www.hex-rays.com/products/ida/index.shtml

[13] N. Idika, A. P. Mathur (2007), A Survey of Malware Detection Techniques, `http://cyberunited.com/wp-content/uploads/2013/03/A-Survey-of-Malware-Detection-Techniques.pdf`

[14] R. Jin (2008), Cluster Validation, `http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf`

[15] J. Lee (2013), Compression-Based Analysis of Metamorphic Malware, *Master's Projects*, Paper 329, `http://scholarworks.sjsu.edu/etd_projects/329/`

[16] W. Li, K. Wang, S. Stolfo, and B. Herzog (2005), Fileprints: Identifying File Types by n-gram Analysis, *6th IEEE Information Assurance Workshop*, 64–71

[17] Malware Protection Center (2010), Win32/Zbot, `http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fZbot`

[18] Malware Protection Center (n.d.), Win32/Winwebsec, `http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec`

[19] V. N. Mandhala, V. Sujatha and B. R. Devi (2014), Scene Classification using Support Vector Machines, *International Conference on Advanced Communication Control and Computing Technologies (ICACCCT 2014)*, 1807–1810

[20] Matlab Statistics Toolbox (n.d.), `http://www.mathworks.com/help/stats/index.html`

[21] P. Mullins (n.d.), Malware, `http://cs.sru.edu/~mullins/cpsc100book/module05_SoftwareAndAdmin/module05-04_softwareAndAdmin.html`

[22] A. Nappa, M. Z Raque and J. Caballero (2013), Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting of viruses, *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, Berlin, DE

[23] M. Onatli (2008), Examples Of Spyware And What They Are, `http://ezinearticles.com/?Examples-Of-Spyware-And-What-They-Are&id=1054106`

[24] OpenCV Tutorials (n.d.), *Introduction to Support Vector Machines*, `http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`

[25] PC Tools (n.d.), What is Spyware and What does it it do?, `http://www.pctools.com/security-news/what-is-spyware/`

[26] L. Piyathilaka, S. Kodagoda (2013), Gaussian Mixture Based HMM for Human Daily Activity Recognition Using 3D Skeleton Features, *8th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 567–572

[27] L. R. Rabiner (1989), A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, 77(2):257–286

[28] B. B. Rad, M. Masrom, S. Ibrahim (2011), Evolution of Computer Virus Concealment and Anti-Virus Techniques:A Short Survey, *International Journal of Computer Science Issues*, 8(1), `http://arxiv.org/ftp/arxiv/papers/1104/1104.1070.pdf`

[29] RapidMiner Documentation (n.d.), `https://rapidminer.com/documentation/`

[30] Security Innovation (2003), Malware: Trojans, Virii and Worms, `http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&uact=8&ved=0CFYQFjAG&url=http%3A%2F%2Fmy.fit.edu%2F~tgillett%2Fswe5900%2Fweek8%2FMalware.ppt&ei=LkowVdPkEMHyoAT4woHQBA&usg=AFQjCNH3B3LVm9JpHln5L2GCjKri5y-OHw`

[31] M. Stamp (2011), *Information Security: Principles and Practice*, second edition, Wiley

[32] M. Stamp (2012), A Revealing Introduction to Hidden Markov Models, `http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf`

[33] M. Stamp (2014), *Machine Learning for Malware Analysis*, first edition, unpublished manuscript, Wiley

[34] R. B. Standler (2002), Examples of Malicious Computer Programs, `http://www.rbs2.com/cvirus.htm`

[35] A. Sulaiman, K. Ramamoorthy, S. Mukkamala, and A. H. Sung (2005), Malware Examiner using Disassembled Code (MEDiC), *Proceedings of the 6th Annual IEEE SMC Workshop on Information Assurance and Security*, 428–429

[36] Support Vector Machines (n.d.), `http://support-vector-machines.org/`

[37] Symantec (2004), Trojan Horse, `http://www.symantec.com/security_response/writeup.jsp?docid=2004-021914-2822-99`

[38] Symantec (2010), Trojan.Zbot, `http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99`

[39] Symantec Security Response (2011), Trojan.Zeroaccess, `http://www.symantec.com/security_response/writeup.jsp?docid=2011-071314-0410-99`

[40] Symantec (2009), What is the difference between viruses, worms, and Trojans?, `http://www.symantec.com/business/support/index?page=content&id=TECH98539`

[41] P. Szor and P. Ferrie (2001), "Hunting for Metamorphic", *11th Virus Bulletin International Conference*, 123–143

[42] P. N. Tan, M. Steinbach and V. Kumar (2006), *Introduction to Data Mining*, Addison-Wesley, Chapter 8, Cluster Analysis: Basic Concepts and Algorithms

[43] A. Vasudevan (2008), MalTRAK: Tracking and Eliminating Unknown Malware, *Computer Security Applications Conference* 311-321

[44] N. Villeneuve with a foreword by R. Deibert & R. Rohozinski (2010), KOOBFACE:Inside a Crimeware Network, `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the-real-face-of-koobface.pdf`

[45] Virus Bulletin (n.d.), Last-minute paper: An indepth look into Stuxnet, `http://www.virusbtn.com/conference/vb2010/abstracts/LastMinute7.xml`

[46] What is Malware (2014), The Monthly Security Awareness Newsletter for Computer Users, `http://www.securingthehuman.org/newsletters/ouch/issues/OUCH-201402_en.pdf`

[47] W. Wong and M. Stamp (2006), Hunting for metamorphic engines, *Journal in Computer Virology*, 2(3), 211–229

[48] Y. Zhou, W. M. Inge (2008), Malware Detection using Adaptive Data Compression,*In Proceedings of the 1st ACM Workshop on AISec*, 53–60