

Automated Malware Detection Based on HMM and Genetic K-means

Saja Alqurashi, Omar Batarfi
King Abdiulaziz Uninversity, Saudi Arabia

Abstract

With the increased use of the Internet and application-based networks, malware detection is a serious challenge. The signature-based detection technique has been widely used as the main method of detecting malware, but with obfuscation techniques, it has failed to detect modern malware. Recent research has proven that a Hidden Markov model (HMM) is useful for malware detection using features that reflect the malware behavior. The motivation in this work is to enhance the working strategy of malware detection. In this study, the related problem of malware clustering based on HMM is considered.

In meeting this goal, this study has proposed a system of testing the malware behavior based on HMM scores, which have been extracted from the learning model on application programming interface (API) call sequences and Operational code (opcode) sequence datasets as malware behavior. API call sequences that extract dynamically and opcode sequences that extract statically are used; they are compared to see which behavior is better for malware detection. Then genetic operators are used in enhancing normal K-means working with the HMM. The proposed genetic K-means is used as a classification algorithm to cluster new behaviors based on the scoring from the HMM. Next, the enhancement results are compared to the normal K-means classification based on the HMM, evaluating the proposed optimization technique.

Therefore, this study is considered to be an optimization enhancement and an evaluation study among normal K-means with the HMM in malware detection, proposing genetic K-means with HMM in malware detection. The results obtained from the experiments demonstrate that the objectives are successfully completed with an average of high detection rate about 87.68%.

1. Introduction

A malicious code refers to a software (or a part of it) that has a harmful purpose to breach the confidentiality, integrity and functionality of a system [1]. McGraw defined a malicious code as:

“any code added, changed or removed from a software system to intentionally cause harm or subvert the system’s intended function” [2]. It includes all families of viruses, worms, Trojans, backdoors, spyware and adware.

Nowadays, the malicious codes are written using special techniques that enable them to escape the detection tools. Some malicious codes can modify their code as well as appearance on each infection. This technique is called obfuscation.

The most widely used techniques to detect malicious codes are signature based detection techniques [3]. To identify malicious codes, these techniques use software signatures, which need to be computed for every malicious code. However, the signature-based detection techniques are unable to cope with new codes whose signatures are not readily available. Before the signature of the new code is computed, there is a sufficient time for the malicious code to infect the systems [3]. In addition, signature based techniques are also ineffective for the codes using obfuscation techniques [4].

There are also other types of detection techniques that do not depend on software signatures. These include heuristic based and behavior-based detection techniques. Behavior based detection techniques analyze behavior of a software to identify if it is malicious. The heuristic based detection techniques employ machine learning and data mining methods to analyze malicious code [3]. For more details on different types of detection techniques.

To avoid any infection from malicious code, it is important to promptly detect it, analyze its behavior and effects on the system. This process is called malicious code analysis. Complete removal of the malicious code from the infected machine requires not only the deletion of malicious software, but also the removal of the associated processes, services and registry entries. To accomplish this task, a thorough understanding of the behavior of malicious code is required. Several detection techniques that analyze behavior of malicious code have been proposed [4].

Anti-virus companies are struggling to develop the tools that can detect most malware. On the other hand, the malicious code developers are making every effort

to escape from these tools. This fight between the security service providers and the malware developers seems a never ending game [5].

The main problem in malicious code detection is to cope with rapidly increasing number of malicious codes that use obfuscation techniques, because such codes can escape from the traditional detection techniques, like anomaly and signature-based detection techniques. To address this problem, we propose classification of malicious code based on their behavior as a malware detection technique.

2. Litterer Review

2.1. Malware Detection Based on Machine Learning Algorithm

Nowadays, modern malware can bypass signature-based detection tools by modifying their appearance using obfuscation techniques. Several researchers have addressed this problem [10], [15], [16]. The most robust solutions exploited machine learning algorithms, like Bayesian Network [6], Neural Network [7], and Hidden Markov Model [8]. These techniques depend on learning the behavior of the malicious code. The following section describes some of these solutions.

2.1.1. Malware Detection Based on Neural Network. Authors in [7] examined the binary code of a software using Neural Network algorithm to find if it is malicious. Their proposed approach takes a binary file as an input and generates an analysis report depicting its degree of maliciousness. It comprises three main components: file DE compilation, behavior recognition, and neural network. The authors found Radux more efficient than Norton [7], Rising and ClamAV to detect different versions of metamorphic malware. In addition, the False Positive Rate (FPR) of Radux is better than several commercial antimalware software.

2.1.2. Malware Detection Based on Bayesian Network. The authors in [6] used Bayesian network to detect metamorphic malware. First, they extracted opcodes and 1-grams from assembly codes of metamorphic malware as their features. Next, a Bayesian network was developed using these features. The nodes of the Bayesian network were connected exploiting hill climbing algorithm [6]. The developed Bayesian network could easily distinguish between viruses and normal files. The authors also showed that the features of lesser importance inversely affect the accuracy of detection and increase the time of network construction.

In all the tests, the accuracy of the HMM was better than the neural network and Bayesian network models. This was mainly because of the two algorithms used in HMM to find the best state.

However, the time to build HMM was comparable to that of the neural network and Bayesian network.

2.2. Malware Detection Based on Hidden Markov Model (HMM)

The authors in [9] used a novel HMM-based method to detect pirated software. Software piracy is referred to as unauthenticated use of a software. It also includes illegal copying of a copyrighted software or its installation on more computers than permitted under the terms of its license agreement. The proposed system comprises two main phases: the training phase and the detection phase. At the beginning, the authors generated morphed copies of software using a morphed generator called "morph.java". The authors generated 60000 tampered files for 10 different base files. Then, they trained HMM using these morphed files. After that, the opcode sequences were extracted from a piece of suspected software, and compared against the trained HMM. The higher the score of comparison, the more is the similarity of the suspected software to the original software and vice versa. On average, more than 50% of files with up to 50% of tampering were detected successfully. This means that software tempered up to 50% can be successfully detected by the proposed technique.

Author in [10] developed a "code emulator" to run the malware in an emulated environment. The code emulator comprises five major components: CPU emulation, memory, hardware, operating system, and emulation controller and analyzer. The code emulator can remove the instructions that were inserted using code obfuscation techniques. Once the metamorphic virus is morphed by the code emulator, HMM can successfully distinguish between viruses and normal files.

The authors in [11] also used an HMM-based classification method to determine malware of metamorphic family. At first step, they used two statistical parameters "normalized term frequency" and "term frequency-inverse document frequency" to eliminate unnecessary files. After that, they trained HMM with n-grams of opcode sequences extracted from different viruses' codes. For each group of viruses, a separate HMM was trained. They calculated the probabilities of observing n-gram sequences of a given code and compared them with the trained sequences.

The work in [12] proposed Profile Hidden Markov Models (PHMMs). They determined the similarity between the given sequences and the trained sequences using the "forwarded algorithm". Each HMM was trained with several n-grams sequences of similar types. Thus, the trained HMMs represented the average behavior of trained n-gram sequences. The authors evaluated the proposed solution for Zbot, win websec and Zero access. The main score of the whole virus family was set as the scoring threshold of

each model. If the virus score is below the threshold, the suspect program is supposed to belong to the virus family. Their experimental results show the effectiveness of PHMMs for some types of metamorphic malware, such as VCL32 and MPC, where the detection rate was 100%. However, PHMMs did not suit well for other types like NGVCK. It means that PHMM can effectively detect the virus family that does not move the blocks of code “far away” from their original positions. Whereas, the standard HMM works well regardless of code shifting. Their experiments showed an accuracy of 90%. This work shows that HMM is a good choice for detecting malware with high accuracy.

The authors in [13] proposed a Hierarchical Hidden Markov Model (HHMM) to eliminate the difficulty of defining state machines model for polymorphic malware. The authors proposed HHMM for effective induction of malware family. HHMM performs better than the traditional HMM. This is mainly because of the following two reasons. One, the topology of polymorphic malware is difficult to predict. Two, Baum–Welch algorithm may not estimate the model parameters effectively due to recursive nature of signature sequences of polymorphic malware and large scales of malware families. The authors also noticed that the polymorphic malware family has hierarchical nature of signature sequences, which facilitates effective modeling of its different components. In addition, using the hierarchical structure, HHMM can also extract related observations from different samples of malware. The authors present a comprehensive framework to understand the rationale behind several decisions made during detection process of polymorphic malware.

Annachhatre et al. [8] presented a new method for malware detection using malware classification. Authors proposed a new technique to detect malware based on classifying malware. This work used clustering based on HMM scores. In this work [10] authors [8] used opcode of malware after extracting it statically. Therefore, train HMMs for each family in dataset. Furthermore, based on HMM score observation sequences against a trained HMM to determine the probability of observing sequence by compute the log likelihood for each instant as score. Based on a scored data use K-means classifying malware, the goal of clustering is to extract relevant structure, which may or may not actually exist, from a given dataset without the aid of a training set to determine the threshold. From this work, it indicates that HMMs are effective tools for automatically classifying malware. This research also used K-means clustering; this procedure classifies the dataset in K clusters, where k is specified in advance. This finds the global optimal cluster in K-means is NP hard problem.

2.3. Static and Dynamic Malware Analysis to Extract Opcode Sequences and API Call Sequences)

The following section describe some researches use static or dynamic analysis for malware, some of them use extract API and other extract opcode.

2.3.1. API Call Sequences. Bayer et al. [5] proposed a novel algorithm to accurately detect variants of well-known viruses without creating their signatures. Using their recommended quantitative measures, they developed a “matrix of similarity scores” depicting the likelihood that a suspected code is a virus. In particular, the authors employed the following two methods for their similarity analysis, Malware Examiner using Disassembled Code (MEDiC) and Static Analyzer for Vicious Executables (SAVE). SAVE exploited API calls and MEDiC exploited assembly calls for malware analysis. In contrast to API calls, assembly calls can be used for in-depth comparisons of executables. However, the methods based on API calls are more efficient than the methods based on assembly calls. Their experiments demonstrated the effectiveness of both MEDiC and SAVE to detect polymorphic malware and variants of known viruses. It is important to note that our efforts on using API calls for malware behavior analysis are in accordance with this work by Bayer et al.

Above all, Uppal et al. [14] presented a novel malware identification approach based on API call sequences. In their work, the authors used odds ratio as feature selection method. Then they applied support vector machine (SVM) algorithm to classify the suspect program as benign and malware. This approach captures API calls by tracking the execution of the suspect program. After API sequence extraction, the authors selected distinct API sequences in two steps: first, they generated the call grams; second, they calculated the odds ratio of each gram and generated the feature vector. After feature selection, various machine-learning algorithms, such as Naïve Bayes, SVM, Decision Tree, and Random Forest, were applied to construct the proposed model for classification, the SVM showed the best accuracy of all algorithms. This work also concludes that API call sequences can be effectively used in behavior-based malware detection.

In another effort in the same direction, Alazab et al. [15] proposed a framework to extract API call sequences as features reflecting malicious behavior. To extract these features from executables, the authors disassembled them using reverse engineering tools. They used the extracted features and their frequencies to assess the similarity between the suspect software and the known cases in the knowledge base. Then, they employed machine learning and data mining algorithms to profile and classify malware behavior. Their experiments showed a high true positive rate of

94% and a low false positive rate of 5.1% [15], indicating that the proposed framework promises to detect malware.

2.3.2. Opcode Sequences. Moskovitch et al. [16] proposed the use of n-grams of opcode as features to detect malware. The opcodes were generated by disassembling the executable files. Using opcode n-gram features, they trained several classifiers including decision trees, SVM, Naïve Bayes, KNN, and similarity based classifiers. They experimented the proposed method on 30,000 files and the results showed an accuracy greater than 99%. They found that the best algorithms were decision tree (J48), KNN and SVM. Overall J48 performed the best. This work concluded that opcodes can be successfully used as a feature to detect malware.

Santos et al. [17] described a novel algorithm to detect variants of known malware. They evaluated the similarity measure for each malware and its set variant using frequency opcode sequences. Further, they also evaluated the similarity of the suspect software with the whole dataset of benign files.

The authors in [18] aimed to overcome the limitations of static malware analysis by proposing a two-phase framework. The first phase analyzes the dynamic behavior of the malware and the second phase classifies malware behavior based on the outcome of the dynamic analysis. The dynamic behavior analysis was performed in two steps: runtime analysis and resource monitoring. The malware behavior of each sample was then classified using artificial intelligence (AI) techniques. The proposed framework showed good results for malware detection.

3. Research Methodology

3.1. Hidden Markov Model (HMM)

Hidden Markov model is a machine learning technique, which acts as a state machine. HMM has states and known probabilities of the state transitions is called a Markov model [13]. A Markov model has the states are visible to the observer. But a HMM has states that are not directly observable [10].

Each state is associated with a probability distribution for observing a set of observation symbols. The transition between the states has fixed probabilities.

Generally, HMM is used for statistical pattern analysis. Also used in speech recognition [10], malicious code detection [10], [11], software piracy detection [9] and biological sequence analysis [12].

To train an HMM, we have used the observation sequences to represent a set of data [13]. Then match an observation sequence against a trained HMM to determine the probability of seeing such a sequence.

If the probability is high, the observation sequence is similar to the training sequences.

As mentioned in [19], the notations used in the hidden Markov models as following (Figure 1):

- T = length of the observation sequence
- N = number of states in the model
- M = number of distinct observation symbols
- Q = distinct states of the Markov Model
- V = set of possible observations
- A = state transition probability matrix
- B = observation probability matrix
- π = initial state distribution
- O = observation sequence

A hidden Markov model is defined by the matrices A, B and π . An HMM is denoted as $\lambda = (A, B, \pi)$

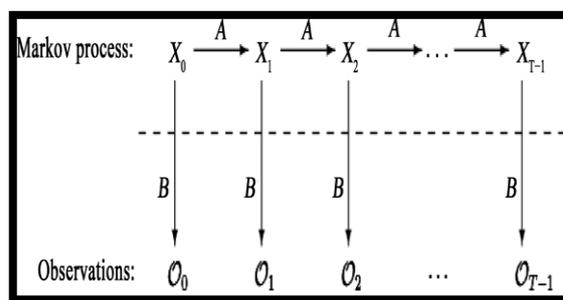


Figure 1. Hidden Markov Model Notations [19]

The following three problems can be solved efficiently using HMM algorithms[19]:

Problem 1: Given a model $\lambda = (A, B, \pi)$ and an observation sequence O, we need to find $P(O|\lambda)$. That is, an observation sequence that can be scored to see how well it fits a given model

Problem 2: Given a model $\lambda = (A, B, \pi)$ and an observation sequence O, we can determine an optimal state sequence for the Markov model. That is, the most likely hidden state sequence can be uncovered

Problem 3: Given O, N, M, we can find a model λ that maximizes probability of O. This is the training of a model in order to best fit an observation sequence.

These three problems can be efficiently solved by the following three algorithms:

- The Forward algorithm.
- The Backward algorithm.
- The Baum-Welch re-estimation algorithm.

The forward algorithm is for calculating the probability of being in a state q_i at time t given an observation sequence O [19]. The forward algorithm, or α pass,

determines $P(O|\lambda)$. The algorithm can be stated as follows.

$$\alpha_t(i) = P(O_0, O_1, \dots, O_t, x_t = q_i | \lambda)$$

The probability of the partial observation sequence up to time t is $\alpha_t(i)$. Using the forward algorithm, $P(O|\lambda)$ can be computed as follows:

$$\text{Let } \alpha_0(i) = \pi_i b_i(O_0), \text{ for } i = 0, 1, \dots, N-1$$

For $t = 1, 2, \dots, T-1$ and $i = 0; 1, \dots, N-1$ compute

$$\alpha_t(i) = \sum_{j=1}^{n-1} \alpha_{t-1}(j) a_{ij} b_i(O_t)$$

Then $P(O|\lambda) = \sum_{j=0}^{n-1} \alpha_{T-1}(j)$

The backward algorithm helps to find a most likely optimal state sequence. This algorithm can be stated as follows [19]:

For $t = 0, 1, \dots, T-1$ and $i = 0, 1, \dots, N-1$ define $\beta(i) = P(O_{t+1}, O_{t+2}, \dots, O_{T-1}, x_t = q_i, \lambda)$

Then $\beta(i)$ can be calculated in following steps:

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

For $t = T-2; T-3, \dots, 0$ and $i = 0; 1, \dots, N-1$, compute:

$$\beta_t(i) = \sum_{j=1}^{n-1} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

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

$$\gamma_t(i) = P(x_t = q_i | O, X)$$

The relevant probability up to time t is given by:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{P(O|X)}$$

The most likely state at any time t is the state for which $\gamma_t(i)$ is maximum.

The Baum-Welch algorithm helps in iteratively re-estimating the parameters A, B, π [19]. It provides efficient way to best fit the observations. The number of states N and number of unique observation symbols M are constant. However, other parameters like A, B and π are changeable with row stochastic condition [19]. This process of re-estimating the model is explained as follows [19]:

Initialize $\lambda = (A, B, \pi)$ with an appropriate guess or random values. For example

$$\pi = 1/N, A_{ij} = 1/N, B_{ij} = 1/M.$$

Compute $\alpha(i), \beta(i), \gamma_t(i)$ and $\gamma_t(i, j)$ where $\gamma_t(i, j)$ is a di-gamma. The digammas can be defined as:

$$\gamma_t(i) = \alpha_t a_{ij} b_j \left(\frac{O_{t+1} \beta_{t+1}(j)}{P(O|\lambda)} \right)$$

$\gamma_t(i)$ and $\gamma_t(i, j)$ are related by:

$$\gamma_t(i) = (x + \alpha)^n = \sum_{j=0}^{n-1} \gamma_t(i, j)$$

Re-estimate model parameters as: For $i = 0, 1, \dots, N-1$ let

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

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

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

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

$$b_j(k) = \sum_{t \in \{0, 1, \dots, T-2\}} \gamma_t(j) / \sum_{t=0}^{T-2} \gamma_t(i)$$

If $P(O|\lambda)$ increases go to step 3.

3.2. K means algorithm

The K-means algorithm is considered the simplest unsupervised learning algorithm. It is related to a solution to the clustering problem. The following section list steps of K-means process [20]:

- The number of clusters for the sample is determined; this is known as (K).
- The K centroids are defined at initialization, one for each cluster. The centroids should be placed as far as possible from each other.
- For each new object in the dataset, we must determine the distance from the K centroid by the simplest distance equation, known as Euclidean distance. Objects to be assigned to one cluster should be assigned to that with the closet centroid.
- The centroids for each cluster are recalculated depending on the object assigned to the cluster.
- The two previous steps are repeated until the distance between the initial and recalculated centroids is negligible.

3.2.1. Problem with K-means. The core difficulty of the K-means algorithm is the advance prediction of the number of clusters that render the algorithm sensitive to the initial cluster centres. In K-means, we first compute the distances between the object and cluster centroid, then compute the average of all distances between objects in the same class as cluster centres. Accordingly, the result of the cluster is affected by this cluster centroid [21].

To solve this problem, we use a genetic algorithm to search for the initial cluster centres; this enhances the K-means algorithm to reduce the impact of the cluster centroid [22].

4. Research Method

In our research, we trained an HMM, using the observation sequences (API call sequences and Opcodes sequences) to represent a set of data. We could match an observation sequence against a trained HMM to determine the probability of observing such a sequence (score). If the probability (score) was high, the observation sequence would be similar to the training sequences; otherwise, it would be different.

The datasets are initially dealt with by preparing them for the HMM process. Once the HMM reaches the dataset records, it will start to build learning objects for each record in the dataset. Each object will be related to a chain that holds within itself an average probability from the contained objects (each object contains information about its learned data). The likelihood probability(score) for each object comes from the learning chain equations in the HMM, which try to connect objects with each other. Meanwhile, learning chains are connected with each other on the top level, while on the level below, a learning object from one chain might be connected to its counterparts in other chain.

After the HMM process, the code extracts a learning model, based on its accuracy, for the next classification process, which will be structured from two components. The first side is the normal K-means, which will be used to test its classification accuracy regarding the malware's data behaviour from both datasets. Then, the proposed genetic K-means will also be used to test its classification accuracy, based on test cases from both datasets

The use of the genetic operation will be guaranteed to find the fittest classification generation in the normal K-mean algorithm; meanwhile, this fitness function will meet several challenges to define the set of classes. Here, the normal K-means will play its art in defining the required set.

K-means algorithm works on scored data comes from learning model see Appendix B. In our project, we chose the numbers of the cluster is five.

Each malware sample is represented as table in appendix B, where each field represents its score generated by the HMM. The range of each dimension of the dataset is divided into 5 parts. Then determine the cluster centroid. Once the centroids are initialized, we enter a loop where each malware is assigned to the cluster corresponding to the centroid closest to it, and then the centroids are recomputed by taking the mean of each dimension of the malware assigned to it. The distance between the malware and the first centroid is calculated using the Euclidean distance equation.

Genetic K-means algorithm first apply k-means process on first population have chosen from dataset also here we chose 5 as the number of the clusters. After first clustering result we starts the Genetic operations, select two objects (sequence) as chromosome then crossover between these chromosome and mutation the chromosome to generate the new generation. To evaluate the new generation meet the optimal solution we use fitness function.

In the aim to evaluate the working behavior of genetic generations, we observe the fitness value since each generation have a value of similarity with the optimization goal.

Although, the fitness value for each generation represented as a probability value between 0 and 1.

Hence, how much we close to 0 is the amount of dissimilarity while near to 1 is the similarity value.

5. Dataset

The structure of the adopted dataset is aimed at reflecting the behavior of the malware according to two types; the first is based on process Opcode sequences [8] while the second is based on API call sequences [22] as shown in figure 4.8 . The authors of [22] analyzed about 23,000 instances of malware dynamically and extracted the API call sequences, as well as obtaining the Opcodes sequences from [8]. Meanwhile, the authors of [8] analyzed 8,000 malware programs statically to extract Opcode sequences (more information about each dataset can be found in the adopted references in this research) [8], [22]. In our project, we have to compare between API call sequences in learning stage with opcode sequences; for that, we chose the common malwares in both datasets, which are 6,900 malwares. In [18] authors compared between five classifiers for malware classification. In their experiment used the fixed size for datasets, they used 220 malwares for each classifier. Also based on data quality criteria in experimental research our dataset met the following criteria; timeliness, accurately and up-to-date, value-added, completeness, appropriate amount of data and relevancy [49]. Consequently, in our project, we equalize datasets size between the experiment based on API call sequences and the experiment based on Opcode sequences, then used them in each experiment. The experiment based on API call sequences has 6900 malwares and the experiment based on opcode sequences has 6900 malwares.

6. Results

The silhouette values for the classification using Genetic -K-means and normal K-means are listed and presented in figure 2,3,4 and 5. The silhouette values [23] is an object consisting of sample data, clustering data, and silhouette criterion values used to evaluate the optimal number of data clusters.

Figure 2 and figure 3 show the values for both normal K-means and genetic K-means for Opcode data; the number of clusters is small in genetic K-means while it is bigger in normal K-means. It shows a relation between the density of the clusters and the silhouette value for each cluster. The density of each cluster refers to the number of objects that has been classified in the cluster.

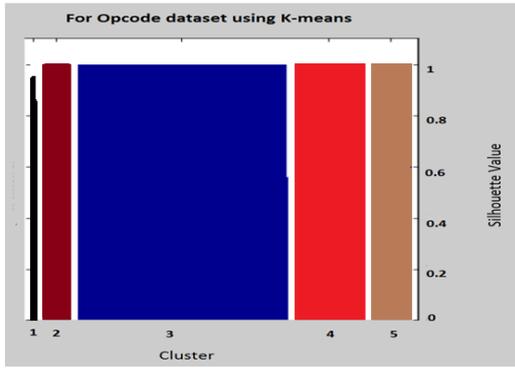


Figure 2. Clustering Opcode dataset using K-means

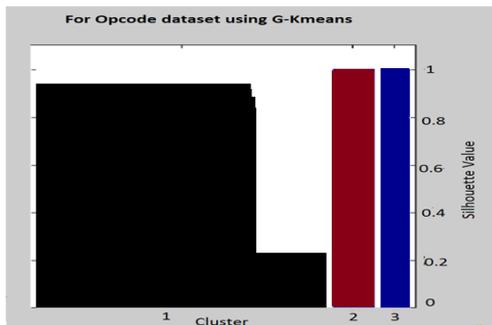


Figure 3. Clustering Opcode dataset using Genetic K-means

The results in Figure 2 show the average of silhouette value for all clusters are high while Figure 3 shows the best value for cluster 2 and 3, and they show the worst value for cluster 1 (a huge part of this cluster's objects is near the 0 value) since each cluster has multiple silhouette values that are related to the density of objects in it; for this we calculate the average of the silhouette values for the cluster. In Figure 3 the use of genetic operation shows a smaller number of clusters and better silhouette values for those clusters.

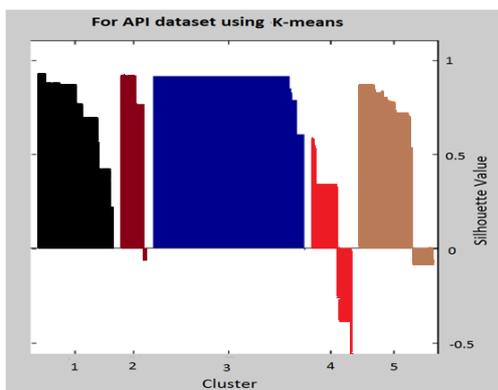


Figure 4. Clustering API dataset using K-means

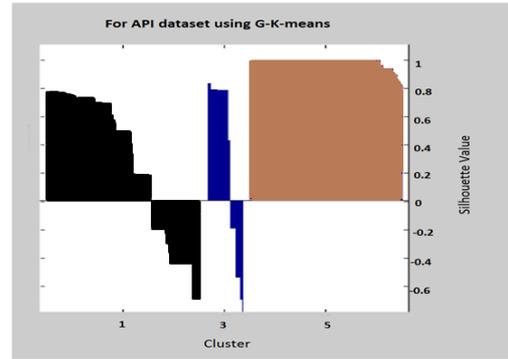


Figure 5. Clustering API dataset using genetic K-means

Also Figure 4 and figure 5 show that the use of genetic K-means is much better than the use of normal K-means in the classification of API data malware. The results in Figure 4 show the best value for cluster 3, and they show the worst value for cluster 4 (a huge part of this cluster's objects is under the 0 value) since each cluster has multiple silhouette

values that are related to the density of objects in it. In Figure 5 the use of genetic operation shows a smaller number of clusters and better silhouette values for those clusters, while the worst cluster silhouette value was for cluster 3.

The aim behind the use of the genetic operation is to adopt using the best generation found in order to be suitable to solve the parameter problem. In this research, the use of genetic operations is meant to provide optimized classification results to normal K-means. Figure 6 shows the optimized behavior during the generations for API data. How close the generation is from +1 shows how the optimized result is strong and fits with the problem until it reaches a level where it will not be enhanced any more (related to the generation stopping condition).

In our methodology, the optimization results founded in the results table in the appendix that represent the optimization results for genetic operations. In the following figures we show the genetic operation workflow.

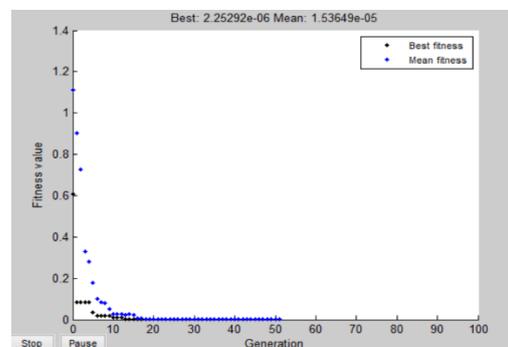


Figure 6. Genetic generations for API call sequences data

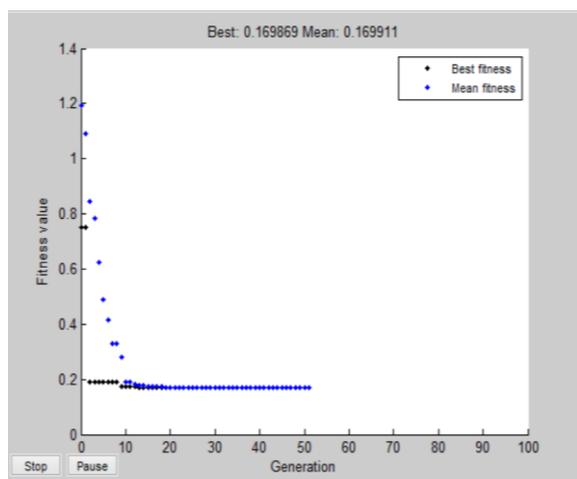


Figure 7. Genetic generations for Opcode call sequences data

The previous figures 6 and 7 represent the genetic operation workflow provided by the genetic method in Matlab so it must be shown in order to evaluate the generation progress over the optimization progress. The accuracy of the generations will be shown for each generation in table 1 and 2 in the appendix; the accuracy is evaluated by the best sum. The best sum is considered to be the best average value for the generation accuracy of the meeting fitness function that has been used in the genetic operations. We took this value as the best generation value that is similar to the wanted output that we engaged with the fitness function.

The tables in the appendix shows a best sum that equals 0.0110281 of classification distances for genetic K-means over the Opcode dataset. Also, it shows 0.413687 for the API dataset. This means that the API call sequences dataset is better than the Opcode sequences dataset in genetic operation.

7. Evaluation Process

To cover and evaluate the working accuracy of the proposed methodology, an evaluation section should be introduced with the aim to prove the enhancement that has been extracted by the methodology.

The second section compares the accuracy results for the classification of normal K-means and genetic K-means with the aim to determine the best algorithm in the detection process. A study provided by [8] has proven that their proposed detection system, which is based on normal K-means with HMM in the detection of malware behaviour, is the best optimization technique over other methodologies. Hence, we introduced the evaluation process between our work and their work. The results extracted from our methodology have proven that the classification accuracy of genetic K-means is better than that of the normal K-means related to the optimization process,

which was done by using the genetic operation over the classification model provided by K-means. The accuracy metrics that are used include silhouette values.

We compute the average silhouette value for each cluster as an accuracy value. Then, to evaluate our proposed cluster algorithm based on the API calls sequences that is better than Opcode dataset in learning model, we compute the average silhouette value for all of the clusters as an accuracy metric for our proposed algorithm. The previous methodology has been applied on the following table.

In table 1 the extracted accuracy results from both of the methodology parts are introduced.

Table 1. K-means vs Genetic K-means Classification Accuracy Results

#test	Normal K-means	Proposed Genetic K-means	Notes
1	75%	82%	G-K is better due to genetic operates
2	73%	71%	Normal K is better due to less operator amount
3	68%	88%	G-K is better due to genetic operates
4	78%	99.8%	G-K is better due to genetic operates
5	77%	99%	G-K is better due to genetic operates
6	71%	84%	G-K is better due to genetic operates
7	82%	92%	G-K is better due to genetic operates
8	76%	98%	G-K is better due to genetic operates
9	72%	71%	Normal K is better due to less operator amount
10	77%	92%	G-K is better due to genetic operates

Table 1 above shows that for a number of tests were conducted by applying the proposed methodology the using of genetic K-means, which is better in malware classification than normal K-means on average in terms of test case accuracy. Where

Genetic K-means averaged 87.68% for classification, normal K-means averaged 74.9%. Though, in some of the test cases, such as test number two, it was shown that using the normal K-means classification is better than using genetic K-means, as it is related to the genetic operators, some cases fall into infinite optimization faults due to the lack of determination of the best fitness function in genetic K-means.

As a second level of evaluation, we evaluate the extracted results from [2] with our proposed enhancement results. The evaluation in Table 2 below shows that using genetic operators with K-means in the malwares detection process is better than using the normal K-means working process.

Table 2. K-means [8] vs. Proposed Genetic K-means

K-means [8]	Proposed means	G-K-means notes
95%	99.8%	We select the best test from our proposed G-K-means

8. Conclusion and Future Work

This work seeks to optimize detecting malware in computers and network systems. This research proposes a genetic K-means algorithm based on the HMM. The proposed enhancement for malware classification uses genetic operators to improve normal K-means. In this work, API call sequences that are extracted dynamically and opcode sequences that are extracted statically are used. The evaluation based on the HMM proves that API call sequences are better for malware detection. The proposed clustering algorithm (genetic K-means algorithm) is applied in classifying the malware based on the scoring of the HMM.

This new optimization technique proposed in this study is evaluated using genetic operators in enhancing normal K-means. Working with the HMM learning model, the accuracy of genetic K-means is compared with the normal K-means classification. The enhanced detection accuracy provided by genetic K-means based on the HMM proves that it is better to provide a near optimal level of optimization. The extracted enhanced accuracy for genetic K-means based on the HMM provides 99.8% in its base detection states over malware, while the HMM with the normal K-means detection provides 82% in its detection based HMM. Therefore, genetic K-means provide a more harmonious scenario in working with HMM chains.

Possible future work could include:

- Try to Find patterns. Global alignment allows a full study in similar chains resulting from the short run. However, it must be seen to apply to identifying local alignment repeated subsequences related to malware behavior within the larger chain.
- Using other optimization algorithms. Other algorithms, such as ant colony optimization, could enhance the working process of genetic K-means.

9. Acknowledgements

Saja Alqurashi, Lecturer in King Abulaziz University MSc in Information Security. Interest in Information Security, IoT security, Malware detection, malware analysis and machine learning.

Omar Batarfi, Associate professor in King Abulaziz University. Interest in information security.

10. References

- [1] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of API calls," Proc. - 2nd Cybercrime Trust. Comput. Work. CTC 2010, no. November 2009, pp. 52–59, 2010.
- [2] Gary McGraw and G. Morrisett, "Attacking Malicious Code: A Reprt to the Infosec Research Council," 2000.
- [3] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," IKT 2013 - 2013 5th Conf. Inf. Knowl. Technol., pp. 113–120, May 2013.
- [4] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," J. Comput. Virol., vol. 2, no. 1, pp. 67–77, 2006.
- [5] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," J. Comput. Virol., vol. 7, no. 2, pp. 107–119, 2011.
- [6] S. Neda and MajidVafaeiJahan, "Metamorphic Virus Detection Based on Bayesian Network," in First International Congress on Technology, Communication and Knowledge (ICTCK 2014), 2012, pp. 4–9.
- [7] Y. Zhang, J. Pang, F. Yue, and J. Cui, "Fuzzy neural network for malware detect," Proc. - 2010 Int. Conf. Intell. Syst. Des. Eng. Appl. ISDEA 2010, vol. 1, pp. 780–783, 2011.
- [8] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden Markov models for malware classification," J. Comput. Virol. Hacking Tech., vol. 11, no. 2, pp. 59–73, May 2015.
- [9] S. Kazi, "Hidden Markov Models for Software Piracy Detction," SAN JOSE Sate University, 2012.

[10] S. Priyadarshi, "Metamorphic Detection via Emulation Metamorphic Detection via Emulation," San Jose State University, 2011.

[11] S. P. Thunga and R. K. Neelisetti, "Identifying Metamorphic Virus Using n-grams And Hidden Markov Model," in In Advances in Computing, Communications and Informatics (ICACCI), 2016, pp. 2016–2022.

[12] S. Attaluri, S. McGhee, and M. Stamp, "Profile hidden Markov models and metamorphic virus detection," J. Comput. Virol., vol. 5, pp. 151–169, 2009.

[13] B. Hariri, S. Shirmohammadi, and M. R. Pakravan, "A Hierarchical HMM Model for Online Gaming Traffic Patterns," in Instrumentation and Measurement Technology Conference Proceedings IMTC 2008. IEEE, 2008.

[14] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware Detection and Classification Based on Extraction of API Sequences," in Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014, pp. 2337–2342.

[15] M. Alazab, "Profiling and classifying the behavior of malicious codes," J. Syst. Softw., vol. 100, pp. 91–102, 2015.

[16] R. Moskovitch et al., "Unknown malcode detection using OPCODE representation," in In Intelligence and Security Informatics, 2008, pp. 204–215.

[17] I. Santos et al., "Idea: Opcode-sequence-based malware detection," in In International Symposium on Engineering Secure Software and Systems, 2010, vol. 5965 LNCS, pp. 35–43.

[18] A. Jantan, "An approach for malware behavior identification and classification," in 2011 3rd International Conference on Computer Research and Development, 2011, vol. 1, pp. 191–194.

[19] M. Stamp, "A revealing introduction to hidden Markov models," in Department of Computer Science San Jose State ..., 2004, pp. 1–20.

[20] J. B. MacQueen, "Kmeans Some Methods for classification and Analysis of Multivariate Observations," in proceeding of 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, 1967, vol. 1, no. 233, pp. 281–297.

[21] W. Min, "Improved K-means clustering based on genetic algorithm," in 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), 2010, pp. 36–39.

[22] Y. Ki, E. Kim, and H. K. Kim, "A Novel Approach to Detect Malware Based on API Call Sequence Analysis," Int. J. Distrib. Sens. Networks, 2015.

[23] Wuest, T., Tinscher, R., Porzel, R., Thoben, K. D. (2015). Experimental research data quality in materials science. International Journal of Advanced Information Technology (IJAIT) Vol. 4, No. 6, December 2014.

11. Appendix: Genetic K-means generations accuracy results over API and Opcode dataset

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	40	0.5306	0.8551	0
2	60	0.2467	0.7464	0
3	80	0.2467	0.5731	1
4	100	0.03055	0.4447	0
5	120	0.03055	0.4218	1
6	140	0.03055	0.3139	2
7	160	0.03055	0.2762	3
8	180	0.01632	0.2187	0
9	200	0.01632	0.142	1
10	220	0.01632	0.1042	2
11	240	0.01632	0.08334	3
12	260	0.012	0.04304	0
13	280	0.012	0.02716	1
14	300	0.012	0.01743	2
15	320	0.012	0.01836	3
16	340	0.004184	0.01457	0
17	360	0.004184	0.01294	1
18	380	0.002231	0.007812	0
19	400	0.002231	0.006853	1
20	420	0.002231	0.006289	2
21	440	0.002231	0.004983	3
22	460	0.001956	0.004676	0
23	480	0.0002412	0.003334	0
24	500	0.0002412	0.001964	1
25	520	0.0002412	0.001362	2
26	540	0.0002412	0.0009862	3
27	560	0.0002412	0.0006497	4
28	580	0.0002412	0.0005421	5
29	600	0.0002412	0.0002853	6
30	620	0.0002412	0.0002914	7
31	640	0.0001191	0.0002582	0
32	660	0.0001191	0.0002487	1
33	680	0.0001191	0.000221	2
34	700	0.0001191	0.0001731	3
35	720	0.0001191	0.0001419	4
36	740	2.954e-06	0.0001228	
37	760	2.954e-06	0.0001172	1
38	780	2.954e-06	7.796e-05	2
39	800	2.954e-06	6.242e-05	3
40	820	2.954e-06	6.409e-05	4
41	840	2.954e-06	3.78e-05	5
42	860	2.954e-06	3.399e-05	6
43	880	2.143e-06	1.778e-05	0
44	900	2.143e-06	2.124e-05	1
45	920	2.143e-06	1.608e-05	2
46	940	2.143e-06	3.375e-05	3
47	960	2.143e-06	2.909e-05	4
48	980	2.143e-06	1.456e-05	5
49	1000	2.143e-06	1.494e-05	6
50	1020	2.143e-06	2.427e-05	7
51	1040	2.143e-06	3.491e-05	8
Best total sum of distances = 0.0110281				

Table 2 for Opcode Dataset

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	40	0.5306	0.8551	0
2	60	0.2467	0.7464	0
3	80	0.2467	0.5731	1

4	100	0.03055	0.4447	0
5	120	0.03055	0.4218	1
6	140	0.03055	0.3139	2
7	160	0.03055	0.2762	3
8	180	0.01632	0.2187	0
9	200	0.01632	0.142	1
10	220	0.01632	0.1042	2
11	240	0.01632	0.08334	3
12	260	0.012	0.04304	0
13	280	0.012	0.02716	1
14	300	0.012	0.01743	2
15	320	0.012	0.01836	3
16	340	0.004184	0.01457	0
17	360	0.004184	0.01294	1
18	380	0.002231	0.007812	0
19	400	0.002231	0.006853	1
20	420	0.002231	0.006289	2
21	440	0.002231	0.004983	3
22	460	0.001956	0.004676	0
23	480	0.0002412	0.003334	0
24	500	0.0002412	0.001964	1
25	520	0.0002412	0.001362	2
26	540	0.0002412	0.0009862	3
27	560	0.0002412	0.0006497	4
28	580	0.0002412	0.0005421	5
29	600	0.0002412	0.0002853	6
30	620	0.0002412	0.0002914	7
31	640	0.0001191	0.0002582	0
32	660	0.0001191	0.0002487	1
33	680	0.0001191	0.000221	2
34	700	0.0001191	0.0001731	3
35	720	0.0001191	0.0001419	4
36	740	2.954e-06	0.0001228	
37	760	2.954e-06	0.0001172	1
38	780	2.954e-06	7.796e-05	2
39	800	2.954e-06	6.242e-05	3
40	820	2.954e-06	6.409e-05	4
41	840	2.954e-06	3.78e-05	5
42	860	2.954e-06	3.399e-05	6
43	880	2.143e-06	1.778e-05	0
44	900	2.143e-06	2.124e-05	1

45	920	2.143e-06	1.608e-05	2
46	940	2.143e-06	3.375e-05	3
47	960	2.143e-06	2.909e-05	4
48	980	2.143e-06	1.456e-05	5
49	1000	2.143e-06	1.494e-05	6
50	1020	2.143e-06	2.427e-05	7
51	1040	2.143e-06	3.491e-05	8
Best total sum of distances = 0.0110281				