Word Embedding Techniques for Malware Classification

Aniket Chandak
San Jose State University

Word Embedding Techniques for Malware Classification

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Aniket Chandak

May 2020

The Designated Project Committee Approves the Project Titled

Word Embedding Techniques for Malware Classification

by

Aniket Chandak

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2020

Dr. Mark Stamp	Department of Computer Science
Dr. Wendy Lee	Department of Computer Science
Dr. Fabio Di Troia	Department of Computer Science

ABSTRACT

Word Embedding Techniques for Malware Classification

by Aniket Chandak

Word embeddings are often used in natural language processing as a means to quantify relationships between words. More generally, these same word embedding techniques can be used to quantify relationships between features. In this paper, we conduct a series of experiments that are designed to determine the effectiveness of word embedding in the context of malware classification. First, we conduct experiments where hidden Markov models (HMM) are directly applied to opcode sequences. These results serve to establish a baseline for comparison with our subsequent word embedding experiments. We then experiment with word embedding vectors derived from HMMs—a technique that we refer to as HMM2Vec. In another set of experiments, we generate vector embeddings based on principal component analysis, which we refer to as PCA2Vec. And, for a third set of word embedding experiments, we consider the well-known neural network based technique, Word2Vec. In each of these word embedding experiments, we derive feature embeddings based on opcode sequences for malware samples from a variety of different families. We show that in most cases, we obtain improved classification accuracy using feature embeddings, as compared to our baseline HMM experiments. These results provide strong evidence that word embedding techniques can play a useful role in feature engineering within the field of malware analysis.

ACKNOWLEDGMENTS

First, I would like to express my gratitude to my advisor Dr. Mark Stamp who always motivated and guided me through the research. His immense knowledge and valuable suggestions has helped me follow the right direction and keeping the research on track. This project would not have been possible without his advise. I would also like to thank my committee members, Dr. Wendy Lee and Dr. Fabio Di Troia, for their time and feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

In this paper, we classify malware samples by applying machine learning techniques to engineered features. The feature engineering techniques themselves involve machine learning, that is, we apply machine learning to generate features. The motivation here is that machine learning can possibly derive more useful features from the available data, and hence classification based on such features may perform better than using the raw features. In this research, we study the effectiveness of such machine learning based feature engineering in the context of malware classification.

Malware is malicious software designed with the intention to harm the computer system or obtain unauthorized access to a computer system [1]. Malware exists on virtually all computer systems, including laptops, smartphones, and software services such as email and internet banking. In recent years, the increase in the number and importance of such computing systems has created many new opportunities for malware-based attacks. This has led to the development of a variety of advanced malware types.

Research aimed at improved detection and classification of malware is a critical topic in information security. Malware samples that belong to the same family should show some similar characteristics and behavior. Classifying malware into its family is a fundamental problem in malware analysis.

Existing approaches to detect malware include behavior-based and signature-based detection. These approaches have various advantages and disadvantages [2]. For example, signature-based detection is relatively fast and efficient, but it fails to detect malware that has not been seen before, and many obfuscation techniques can defeat signature scanning. On the other hand, behavior-based detection can detect new malware, but such approaches are often costly and have unacceptable false positive rates. Recent advances in learning techniques and the availability of large datasets has resulted in significant improvement over traditional malware detection approaches.

In this research, we consider the use of word embedding techniques based on opcode features for malware classification. We consider embedding techniques based on hidden Markov models (HMM) [3], principal component analysis (PCA) [4], and the neural network based technique, Word2Vec [5]. We refer to the techniques based on HMM and PCA as HMM2Vec and PCA2Vec, respectively. For all of the word embedding techniques considered, we generate the embeddings based on opcode sequences, and we experiment with a substantial malware dataset that includes samples from seven different malware families. For each embedding technique, we also consider an additional feature engineering step, where PCA is used to reduce the dimensionality of the feature embeddings prior to the classification step. We refer to this method as k-PCA where k represents the dimension of the reduced feature vectors. In all cases, for classification, we experiment with k-nearest neighbors (kNN), multilayer perceptrons (MLP), random forest (RF), and support vector machines (SVM).

The remainder of this paper is organized as follows. In Chapter 2, we provide background on malware detection and the various machine learning techniques used in this research. In Chapter 3 we discuss our dataset and experimental results. Finally, Chapter 4 concludes this report, and we consider some directions for future work.

CHAPTER 2

Background

Malware is one of the most alarming and crucial threats existing in the era of the internet. As per the Internet Threat Security Report 2019 [6], there is an increase of 25% compared to the previous year in the number of attack groups using malware to disrupt the business of the organization. As per California Data Breach Report 2016 [7], malware has contributed to 54% of breaches and 90% of total records breaches which accounts for 44 million records breach in the year 2012–2016. As per the Kaspersky Security Statistics 2017 [8], 29.4% of personal computers suffer a minimum of one malware attack over the year. Overall statistics imply that malware are increasing rapidly in terms of momentum, quantity, and variation.

Various factors like large volume, obfuscation, detection speed, detection of the virtual environment make it difficult to analyze and detect malware [9]. Every day, thousands of new malware are generated and it is difficult to analyze the enormous volume of data. Obfuscation is a technique used by malware creators to make it difficult to read the content of malware code, e.g., by adding a dead code [9]. Advancement in malware creation has reduced the speed of malware detection. Malware can cause damage if detection is delayed. Malware detection plays an important role in the information security domain because of these challenges.

Many companies and software products are dependent on conventional methods to detect malware. These methods can be classified into two types, signature-based detection, and behavioral-based detection. Antivirus companies use a signature-based method to detect malware and protect legitimate users from attack [2]. The signature-based method is based on the identification of a unique pattern in a file [10]. It refers to the database of existing known malware for pattern matching to classify a file as malware [2]. Signature extraction is a complex activity because it involves manual intervention. In this method, detection is faster but can be easily bypassed using obfuscation. Obfuscation is adding a dead code into the file. This method fails to detect new malware which are not part of the database.

Another method, behavioral-based detection is based on identifying the actions performed instead of a pattern [2]. When the file performs any action, which does not fall under normal behavior, this method triggers an alarm for the file. This method can detect obfuscated malware, however, the speed of detection is slower because of the complexity to detect actions. Limitations of conventional methods can be defeated with machine learning based solutions. In the following section, we describe previous work done in the malware detection and classification using machine learning.

2.1 Previous Work

Effectiveness of the machine learning algorithm not only depends on the task at hand but also the characteristics of features. In malware classification, a file can be represented with features associated with it such as opcode, bytecode, API calls, and permissions, etc. Features can be used to perform analysis using a compatible machine learning algorithm. Opcode has been used to detect the malware based on a count of opcode in the file [11]. An opcode is a machine-level instruction of a program. Support vector machine trained on 20 most frequently used opcodes as a feature with 100 malware and benign sample to achieve a 96.67% success rate.

In [12], the authors experimented with 67 malware and 20 non-malicious files and extracted opcodes from these files. Statistics in the experiments conclude that opcodes can be used as a feature to differentiate between malware and non-malware [12]. Another research in [13] achieved good results by using API calls as a feature. API can be defined as functionality or action performed by a program. They experimented with 3536 malicious programs and extracted API features using a sandbox deriving accuracy as 0.87 in malware detection [13]. Extracting API calls from any file is a difficult task as compared to extracting opcode. As the time efficiency is an important factor in designing malware detector, the opcode is a better choice as a feature compared to API calls.

Another research in [14] used opcode sequence as a feature for malware detection. Experiments in the research consider n-gram opcode sequence with values of $n = 2, 3, 4$. They introduced the Markov blanket for feature selection from the large n-gram feature set. Mutual information value is used for selecting the feature and reduced feature size by 99%Ȧfter feature selection, classifiers trained on hidden Markov model for five malware families contributing to 1200 malware files and 194 benign files. They achieved 99% precision and 98% sensitivity.

There have been researches in feature engineering using machine learning techniques. Word2Vec is used for feature extraction for malware [15]. In this research, malware is treated as a language for semantic analysis using language modeling techniques. They used the Word2Vec model to extract contextual information from the opcode sequence of malware. Learned contextual information is used as a feature for classification using kNN. Distance between vectors is calculated using the word movers distance algorithm in kNN classification. This method achieved 95% accuracy for 9 malware families. The research concludes that word embedding techniques such as Word2Vec are effective in understanding the contextual information in malware and word embeddings act as an effective feature for classification. In this research, the author does not consider classification methods other than kNN, also this research is focused on language modeling using Word2Vec and does not explore other language modeling techniques.

Another research in [16] used Word2Vec to generate the feature vector. The opcode sequence is used for training the Word2Vec for feature extraction. A deep neural network is trained using features for malware classification. The proposed method considers large number of opcodes in the range 50 to 200 and vector embedding of length in the range 250 to 750. Binary classification experiments are performed on 1200 benign and 1200 malicious samples. Proposed experiments can be further extended to cases with less number of opcodes and shorter embedding length. The research shows that malware can be treated as a language for semantic analysis.

In [17], the proposed method uses a similar approach based on Word2Vec with an interesting combination of TF-IDF. The sequence of API syscall is used as a feature in the research. Word2Vec embeddings represent contextual information of feature and TF-IDF vector represents relevancy of feature. In the proposed method, Word2Vec embeddings of the feature are multiplied by the TF-IDF weight of respective features. This additional step makes contextual information stronger for a more relevant API syscall. Further, this information is used to perform classification using kNN, RF, and SVM. The limitation of the proposed solution is the use of dynamic features.

Word2Vec embeddings are used as a feature for training the BLSTM [18]. Bi-directional LSTM is improved version of LSTM [19]. The experiments achieved good accuracy for malware detection. In [20], the author proposed the word embedding method based on the graph. In this method, the graph is generated using opcode information. This graph is projected into vector space to generate word embeddings. In the classification task, graph generation shows slower performance.

In [21], the author states that word embedding techniques based on traditional machine learning methods are comparable to a neural-network approach. Their work shows that word embeddings can be generated using a matrix of pointwise mutual information (PMI) of the respective word and principal component analysis (PCA). In this research, we experiment with a similar approach based on the PMI matrix for generating word embeddings for malware.

In the survey of related work, we understood that opcode acts as a good feature in malware classification. We learned that machine learning techniques based on word embedding can be used for feature engineering. However, there are no relevant studies based on HMM models to extract word embeddings.

2.2 Background on Machine Learning Techniques

In this section, we present various machine learning techniques that are used in the experiments discussed in Chapter 3. We introduce HMMs and PCA, which form the basis for the word embedding techniques that we refer to as HMM2Vec and PCA2Vec, respectively. Finally, we introduce four classification techniques used in our experiments.

We also discuss HMM2Vec, PCA2Vec, and the neural network based word embedding technique, Word2Vec, in detail. For our experiments in Chapter 3, we use this three word embedding techniques along with other techniques to generate features vectors. Feature vectors are used in the classification experiments for comparison.

2.2.1 Hidden Markov Models

Hidden Markov model (HMM) is a machine learning model based on a statistical Markov model representing the probability distribution on the observation sequence [22]. HMM is a discrete hill-climbing algorithm represented using initial state probabilities, observation probabilities in hidden states, and state transition probabilities. Information related to the working of hidden Markov model, applications, and algorithm is mentioned in [22], which include detailed algorithms or Rabiner's classic paper [23].

2.2.2 Principal Component Analysis

Principal component analysis (PCA) is a machine learning technique usually used in dimensionality reduction problems [4]. Trained PCA represents the eigenspace where original data can be projected. This technique does not eliminate any feature but packs the entire feature in a more concentrated form with fewer dimensions. More information on PCA is mentioned in [4] and also to understand the math in detail behind the PCA refer [24]. The discussion at [25] also provides more insight into PCA.

2.2.3 Word Embedding Techniques

Word embeddings are often used in natural language processing as they provide a way to quantify relationships between words. Here, we use word embedding to generate higher-level features for malware classification.

In this section, we discuss three distinct word embedding techniques. First, we consider word embeddings derived from trained HMMs, which we refer to as HMM2Vec. Then we consider a word embedding technique based on PCA, which we call PCA2Vec. Finally, we discuss the popular neural network based technique Word2Vec.

2.2.3.1 HMM2Vec

We will discuss one simple example before jumping to word embedding, where we consider the letters instead of words and will call this approach as Letter2Vec. HMM model represented with three matrices A, B, and π which represents hidden state transition probabilities, observation probability distribution in hidden states and initial state probabilities respectively with row stochastic property. Notation-wise, N represents the number of hidden states, M represents count of distinct symbol in observation, and length of observation symbol is represented by T. Users can define the value of N whereas M and T are decided by the training data.

Consider the experiment where we train the HMM on English text with each letter as an observation symbol and ignore the case-sensitivity and characters other than alphabetical letters. Thus $M = 27$ (letters plus word-space), and we choose $N = 2$ hidden states, and $T = 50{,}000$ length of observation sequence. Table 1 represents $B^{\intercal}$

Table 1: Initial and Final $B^\intercal$

Observation	Initial		Final	
a	0.03735	0.03909	0.13845	0.00075
b	0.03408	0.03537	0.00000	0.02311
c	0.03455	0.03537	0.00062	0.05614
d	0.03828	0.03909	0.00000	0.06937
e	0.03782	0.03583	0.21404	0.00000
f	0.03922	0.03630	0.00000	0.03559
g	0.03688	0.04048	0.00081	0.02724
h	0.03408	0.03537	0.00066	0.07278
i	0.03875	0.03816	0.12275	0.00000
j	0.04062	0.03909	0.00000	0.00365
k	0.03735	0.03490	0.00182	0.00703
l	0.03968	0.03723	0.00049	0.07231
m	0.03548	0.03537	0.00000	0.03889
n	0.03735	0.03909	0.00000	0.11461
o	0.04062	0.03397	0.13156	0.00000
p	0.03595	0.03397	0.00040	0.03674
q	0.03641	0.03816	0.00000	0.00153
r	0.03408	0.03676	0.00000	0.10225
s	0.04062	0.04048	0.00000	0.11042
t	0.03548	0.03443	0.01102	0.14392
u	0.03922	0.03537	0.04508	0.00000
v	0.04062	0.03955	0.00000	0.01621
w	0.03455	0.03816	0.00000	0.02303
x	0.03595	0.03723	0.00000	0.00447
y	0.03408	0.03769	0.00019	0.02587
z	0.03408	0.03955	0.00000	0.00110
space	0.03688	0.03397	0.33211	0.01298

of trained HMM [26], Suppose that we represent a letter using the corresponding row of the converged matrix $B^\intercal$ in the last two columns of Table 1. Suppose that for a given letter ℓ, we define its Letter2Vec representation $V(\ell)$ to be the corresponding row of the converged matrix $B^\intercal$ in the last two columns of Table 1. Then, for example,

$$\begin{aligned} V(a) &= (0.13845, 0.00075) \quad & V(e) &= (0.21404, 0.00000) \\ V(s) &= (0.00000, 0.11042) \quad & V(t) &= (0.01102, 0.14392) \end{aligned} \tag{1}$$

These embedding can be used to calculate the cosine distance between the vectors.

The cosine similarity of vectors X and Y is given by

$$\cos(X,Y) = \frac{\displaystyle\sum_{i=0}^{n-1} X_i Y_i}{\sqrt{\displaystyle\sum_{i=0}^{n-1} X_i^2}\sqrt{\displaystyle\sum_{i=0}^{n-1} Y_i^2}}$$

Where $X = (X_0, X_1, \ldots, X_{n-1})$ and $Y = (Y_0, Y_1, \ldots, Y_{n-1})$ are the vectors. As the cosine similarity represents the cosine angle between the vector, a value closer to 1 represents a similar vector and vice versa. Here if we see, for the letter embedding in equation (1), vowels “a” and “e” are close as $\cos(V(a), V(e)) = 0.9999$ whereas vowel “a” and the consonant “t” are apart as $\cos(V(a), V(t)) = 0.0817$. These results show that letter embedding carries important information related to the letter. Similar to Letter2Vec embedding, HMM can be trained on observation sequence consisting of words and define the embedding using the resulting B matrix.

While HMM2Vec is possible to train there are some limitations to training. The most critical limitation is as HMM is based on a Markov model of order one, therefore the resulting vectors of HMM will have limited context information. The state of the art approach of Word2Vec is trained on data corresponding to $M = 10{,}000$, $N = 300$, and $T = 10^9$, and training HMM would be difficult because of the order of N^2T work in Baum-Welch re-estimation.

2.2.3.2 PCA2Vec

Another technique in this research for generating word embedding is to apply PCA on a special matrix. This special matrix used is constructed based on pointwise mutual information (PMI) using window size W. To construct a PMI matrix, we calculate $P(w_i, w_j)$ for all pairs of words (w_i, w_j) that occur within a window W of each other within out observation sequence dataset. In the process compute $P(w_i)$ for each individual word w_i. Then we define the PMI matrix as

$$X = \{x_{ij}\} = \log \frac{P(w_j, w_i)}{P(w_i)P(w_j)}$$

We represent column i of X, denoted X_i, as the feature vector for word w_i Next step is to perform PCA training on these X_i feature vectors, and then project the feature vectors X_i onto the resulting eigenspace. When projecting the feature vector into eigenspace, length N of final feature vector will be decided by choosing the N dominant eigenvalues. Similar properties of these embedding vectors is shown in [27] and also some research claims [28] that eliminating eigenvectors with dominant eigenvalues is beneficial to gain more information in embedding vectors. More details on using PCA to generate word embeddings can be found in [28] and [27].

2.2.3.3 Word2Vec

Word2Vec is a famous word embedding technique based on a shallow neural network which can be used for embedding any feature into a high-dimensional space. After the training of neural network, words that are more similar in context will be close to each other compared to words that are not similar in context. Another surprising thing about these embedded vectors is that they hold algebraic property. For example, according to [5], if we let

$$w_0 = \text{“king”}, w_1 = \text{“man”}, w_2 = \text{“woman”}, w_3 = \text{“queen”}$$

and we assume $V(w_i)$ to be the Word2Vec embedding of w_i, then it is observed that $V(w_3)$ is closest to

$$V(w_0) - V(w_1) + V(w_2)$$

And the closeness is defined by the cosine similarity between the vectors. Word2Vec and HMM2Vec hold similarity in the ways it is being used. In both cases, we are not interested directly in the model itself but rather in the learning of the model, i.e., we are more interested in model representation after training.

A general discussion of Word2Vec can be found in [29] and a good introduction is given in [30]. The original paper describing Word2Vec is given in [5] and improvements on original implementation are given in [31].

2.2.4 Classifiers

Features generated by word embedding techniques are used to classify malware into the respective family. Machine learning based classifiers can be used to analyze the quality of generated features. There are many existing machine learning based classifiers. In the research presented in this paper, we consider four different classifiers, namely, k-nearest neighbors (kNN), multilayer perceptron (MLP), random forest (RF), and support vector machine (SVM). Experiments discussed in Chapter 3 use these classifiers.

2.2.4.1 k-Nearest Neighbors

One of the simplest machine learning techniques is k-nearest neighbors where classification is based on the vote of nearest k samples in training data. The distance measure formula can be Euclidean distance, Manhatten distance or any other distance formula. This is also a lazy machine learning technique as it does not involve any training phase but at the same time, more training samples make the scoring phase slower. More information on kNN can be found in [26].

2.2.4.2 Random Forest

Random forest (RF) is the technique used to overcome the overfitting problem in the decision tree by generalizing the decision tree algorithm. An RF consists of multiple decision trees using subsets of features and samples and uses the majority vote of decision trees to make a final decision on classification. RF and classification using RF are discussed in more detail in [26] and [32].

2.2.4.3 Support Vector Machine

Support vector machine is the type of supervised learning method whose goal is to learn hyperplane to separate the input labeled data. This separating hyperplane is capable of maximizing the separation between classes and can work in higher dimensional space with the help of a kernel trick. More information on support vector machines and the kernel are given in [26]

2.2.5 Multilayer Perceptron

Multilayer perceptron (MLP) is a feedforward neural network and is one of the simplest neural network which can be used for classification and regression [33]. It consists of an input layer, an output layer, and multiple hidden layers. It learns the best value for weights in those layers to minimize the error. More details on the architecture and working of neural networks are given in [33] and [26]

2.2.5.1 Last Word on Classification Techniques

MLP and SVM are related as they share some similarities in the approach they work. Both of them are capable of creating nonlinear decision boundaries as in SVM nonlinear kernel can be used whereas MLP learns the non-linearity using the data. This implies MLPs have an advantage as there is no need to specify correct kernel and model can learn that based on data. On the other hand, MLP requires more computation and data to train as it has more things to learn than comparable SVM. Another observation is the similarity between kNN and RF. They both are the same in terms of algorithm based on neighbors but there is a structural difference in both the approach to look at neighbors [26].

Thus, considering the similarity we expect that kNN and RF to show similar results. Also, SVM and MLP should be closely related in terms of results. This relation can be used for a sanity check of experiments. If the results for SVM and

MLP have a significant difference then we should investigate experiments further. On the other hand, if the results are significantly different for SVM and RF then it should not raise the same concern.

CHAPTER 3

Experiments and Results

In this chapter, we explain malware families considered in the research. Also, we briefly discuss the dataset, machine learning techniques used for feature extraction and classification.

3.1 Dataset

Malware families in Table 2 represent the dataset used for the experiments in this study. In this research, 1000 samples are randomly selected from each of these families to keep balance in the dataset and considers 7 malware families making 7000 samples in total. These families have been used in many recent studies, including [34] and [35], for example. Malware families in Table 2 are of different types. We will briefly discuss each of these families used in our experiments.

Table 2: Malware Families

Family	**Type**	**Samples**
BHO	Trojan	1396
CeeInject	VirTool	1077
FakeRean	Rogue	1017
OnLineGames	Password stealer	1508
Renos	Trojan downloader	1567
Vobfus	Worm	1107
Winwebsec	Rogue	2302
Total	—	9974

BHO — Malware in this family are capable of a range of malicious actions. These actions can be specified by attacker [36].

CeeInject — Malware in this family are designed to avoid detection. Therefore many families use it as a shield to avoid detection. For example, CeeInject can be used to obfuscate a bitcoin mining client, making it possible to be installed on the user's system without their knowledge or consent [37].

FakeRean — Malware in this family shows fake issues in a user's system and ask them to pay to clean the system [38].

OnLineGames — Malware in this family are used to steal information such as login credentials of a user for online games and their keystroke activity [39].

Renos — Malware in this family states that the system has spyware and ask a user to pay money to remove the mentioned spyware [40].

Vobfus — Malware in this family damages user's computer by downloading other malware and tweak the system configurations that can not be restored easily by cleaning the downloaded malware [41].

Winwebsec — Malware in this family acts as an antivirus for the user's system and shows false information that the device has been compromised and asks a user to pay money to clean the system [42].

3.2 Data Pre-processing

Malware samples in our dataset are executable files in raw form. Executable malware samples are disassembled using objdump to extract features. Objdump is a part of GNU Binutils in Linux which can be used to disassemble the executable. We disassembled 7000 malware samples to extract the sequence of an opcode from the executable file. A large number of distinct opcodes are present in disassembled code. Considering all opcodes in experiments do not add value and increases the overhead in the training of machine learning models. There are more than 200 different types of opcodes and it is not efficient to use all the opcodes in the experiment. Previous research in [11] considered the top 20 opcodes in malware classification. We calculated the top 20 opcodes in our dataset based on frequency. Based on the those, the opcode sequence is filtered for 7000 samples. Figure 1 shows the top 20 opcodes and their percentage. It contribute to 69.8% of the total opcodes.

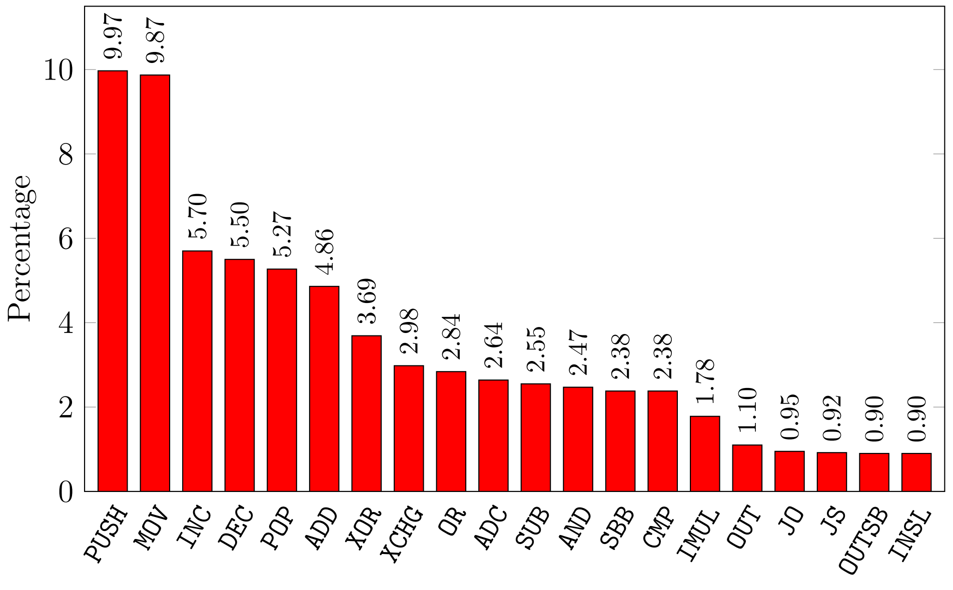


Figure 1: Top 20 Opcodes

Table 3: Top Opcode Percentage

Opcode	**Percentage**
Top 10	53.4%
Top 20	69.8%
Top 30	78.3%
Top 40	85.1%
Top 50	89.5%
Top 60	92.7%
Top 70	95.4%
Top 80	97.6%
Top 90	98.5%
Top 100	98.9%

We conducted binary classification experiments discussed in Section 3.3.5.1 with the top 20 and 30 opcodes. Figure A.19, and A.20 represents experiment result for a model trained on Renos family and tested against Onlinegames family. Results show that there is no significant improvement in using the top 30 opcodes over top 20. Based on these experiments and use of the top 20 opcode in [11], we select the top 20 opcodes.

3.3 Feature Engineering

In this section, we discuss the experiment for feature engineering using machine learning. The features are used as input for classifiers discussed in Section 2.2.4. We use multiple machine learning techniques to model the features for malware samples in our dataset. The focus of this section is to explain experiments for HMM2Vec, PCA2Vec, and Word2Vec. It also discusses the technique to reduce the dimension of features using PCA based pipeline, referred to as k-PCA. To compare the results of the proposed method, a simple baseline HMM score approach is used. We expect the proposed technique based on word embedding to perform better than the baseline approach.

3.3.1 HMM Score

First, we consider experiments based on HMMs and opcode sequences. We choose these HMM-based experiments for the baseline approach for comparison. The opcode feature for training HMM has shown good results in many studies [43, 44, 45, 46, 47]. In this experiment, we train the HMM model for 7 families. When training the HMM model for given family, the observation sequence is generated by appending 10 random samples. To avoid the convergence at local maxima, HMM is trained 10 times with different initial values of A and B matrix to select the best model.

To create a feature vector, 500 random samples from each family are scored using the HMM models. These scores from 7 HMM models form a feature vector. The position of score from a specific HMM model is fixed in the feature vector and does not change across samples. The feature vector will be represented as `<BHO score, CeeInject score, OnLine Games score, Renos score, Winwebsec score, FakeRean score, Vobfus score>`. Here we generated 3500 labeled samples (500 from each family) with a feature vector of length 7.

3.3.2 HMM2Vec

Another technique we use for feature extraction is the hidden Markov model. To train the hidden Markov model, an observation sequence and a number of states parameter are required. The opcode sequence is treated as an observation sequence in the HMM training. The remaining training parameters for HMM and respective values are given in Table 4. The complete process of training the HMM2Vec is shown in Figure 2.

Table 4: Parameters in HMM Training

Parameter	**Description**	**Value**
T	Length of observation sequence	500 to 200000
N	Number of states	2
M	Number of observation symbol	≤ 20

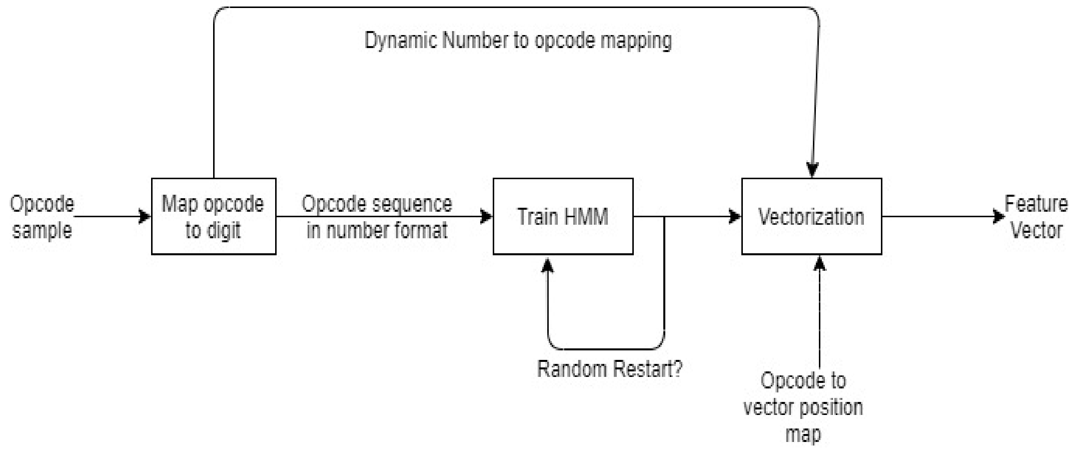

Dynamic Number to opcode mapping
Opcode sample
Map opcode to digit
Opcode sequence in number format
Train HMM
Vectorization
Feature Vector
Random Restart?
Opcode to vector position map

Figure 2: HMM2Vec Feature Generation

3.3.2.1 Adaptive Random Restart in HMM

HMM is a discrete hill climb algorithm. Therefore HMM training can sometimes converge at local minima. To avoid convergence at local minima, HMM can be trained from multiple starting points on hill, i.e., in our case with different initialization of A, B, and π matrix. From those multiple initialization, consider the model which reached the highest point on the hill and discard other models. This method is called as random restarts in HMM training. In this method, random restarts represent the number of models trained from different initialization. It is usually observed that shorter observation sequence requires more random restarts compared to longer observation sequence. Also, random restarts for longer observation sequence will take more time to train, which is unnecessary if convergence is achieved earlier. To address this issues, we implemented adaptive random restarts to ensure model convergence in training. Adaptive indicates that a number of random restarts is decided based on the length of the observation sequence. Table 5 represents the number of random restarts based on length when training HMM.

Table 5: Number of Random Restarts

Observations	**Restarts**
>30000	10
10000-30000	30
5000-10000	100
<500	500

3.3.2.2 Code Details

Hmmlearn python library is used for training the HMM. It requires valid observation sequence represented as a sequence of numbers in range 0 to $|V|-1$, where $|V|$ is the number of the distinct observation symbols, and each number exist at least once in the observation sequence. The constraint in the library is to map observation symbol in B matrix column with two dimensional array in python, for example, the 0th column represents state probabilities for observation symbol “0”. To satisfy the constraint of the library with opcode sequence, it is required to map opcode with number with help of mapping.

The opcode to digit mapping is not static for all observation sequences, i.e., we generate new mapping for each observation sequence. Every sample does not have all of the 20 opcodes. In cases, when opcode is absent in the observation sequence, the use of static mapping will invalidate the observation sequence constraint. Hence same static mapping can not be used for all samples. Consider static mapping is used and `ADD` is mapped with number 3 and a sample in the dataset does not have the `ADD` opcode, therefore number 3 will be missing in the observation sequence. This observation sequence is invalid for hmmlearn training as per the constraint discussed earlier. Hence we generate new mapping for each sample to represent opcode sequence as a sequence of numbers in range 0 to $|V| - 1$.

3.3.2.3 Parallel Restarts in Hidden Markov Model

In this research, HMM is used to train around 7000 models. Training the HMM is a computationally expensive. Adding the overhead of random restart on training increases the execution time. All the random restart instances of training the HMM are independent of each other. Available libraries to train the HMM do not consider the parallel training of random restarts. We implemented the parallel version of HMM training, which trains multiple models in parallel based on available CPU cores in the system. The multiprocessing module in python is used to leverage the multiple processors available in the system. Figure 3 shows that the parallel approach improves the execution time for more random restarts compared to a sequential approach. The improvement depends on the number of CPU cores and our experiment shows the parallel execution is 4 times faster with 8 core CPU than the serial execution. Initially, the graph shows that the parallel approach is slower than the serial for small values of the random restart. The reason is, overhead of dividing the task on CPU and collecting back the results is more than execution itself for a few random restarts.

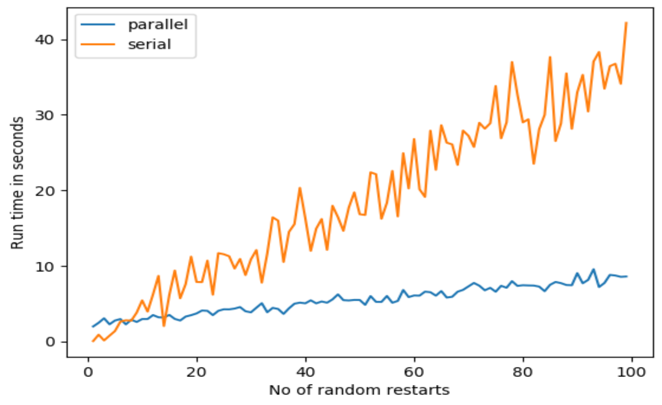


Figure 3: Parallel vs Serial Random Restarts

3.3.2.4 Vectorization of HMM

This step involves generating a feature vector from HMM which is consistent across all the samples. To maintain consistency in a feature vector, a position in the feature vector is fixed for a given feature. Any information from A matrix or B matrix in vectorization have a fixed position in a feature vector, for example, the observational probability of `MOV` opcode in state 0 take a fixed position in feature vector for every sample.

There are two challenges in maintaining consistency. First, the format of the B matrix is not consistent across all the models, i.e., trained HMM model can have different dimensions as a value of M is not fixed for all samples. Also, the order of columns in B matrix is inconsistent, i.e., column number for given opcode is not fixed B matrix. Second challenge is unknown state in the HMM as per the term "hidden." The `MOV` opcode in can converge in either state 0 or state 1 in a given sample.

To solve these challenges, our HMM model vectorization code incorporates logical steps. The first is to swap the rows in B matrix if required to maintain consistency in the state. We considered a specific opcode for which there is significant convergence in one of the states. For every sample, if that opcode has convergence in state 0 then we do not swap the state in the respective HMM model. If the matrix has convergence in state 1 (another state) we swapped the rows in A and B matrix. After swapping the convergence is consistent in state 0.

To solve the second challenge, we maintained the fixed mapping of opcode and position in the feature vector. This mapping is the same throughout all the experiments and does not change for any sample. This mapping is used in vectorization of the B matrix. For the cases, when B matrix has dimension less than 20, i.e., not all the opcodes are present in the observation sequence, 0 is appended.

3.3.2.5 Results for HMM2Vec Versions

Three different versions of HMM2Vec considered are plain HMM with no random restarts and no state swap, HMM with state swap and without A matrix, and HMM with random restarts. SVM based classification experiments were conducted on these versions of feature vector with k-PCA reduction. Figure A.21, A.22, and A.23 in the appendix represent the confusion matrices for the experiments. Experiment results in 0.79, 0.80, and 0.90 accuracy respectively for previously mentioned versions. It shows that B matrix does not contribute to feature and removing B matrix from vector does not reduce the accuracy. Also, the swapping states does not improve accuracy in a considerable amount. The reason can be a high correlation of features within states and relation is learned by the machine when classifying.

3.3.3 Word2Vec

Word2Vec is another technique used in this research for feature engineering. Word2Vec is most popularly used to learn word embedding with the help of the shallow neural network. In this experiment, Word2Vec is trained using the opcode sequence. After training the Word2Vec model, the feature vector is generated by appending opcode embedding. If the opcode is missing in the observation sequence, zero vector is appended, for example, append two zeros in feature vector for missing opcode when the length of the embedding vector is 2.

To keep it consistent with HMM with two states, Word2Vec with the length of embedding vector as 2 and window size as 10 is trained. We vectorize the trained Word2Vec model by appending the embedding of opcodes in feature vector. The length of this feature vector depends on the length parameter in the training of the Word2Vec model.

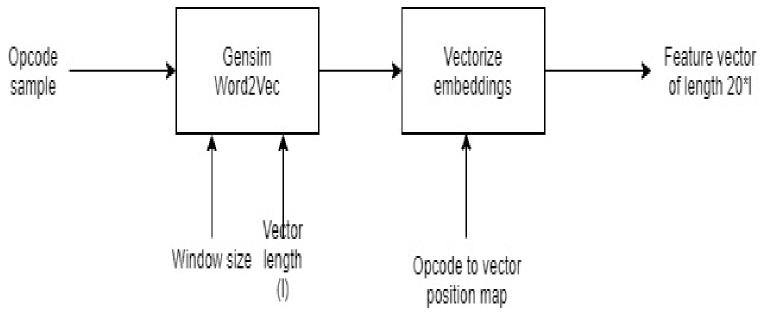


Figure 4: Word2Vec Feature Generation

3.3.3.1 Code Implementation

For training Word2Vec, we used the gensim module in python [48]. Gensim allows us to specify parameters such as window, size of embedding vector, and underlying training algorithm. Continuous bag of words (CBOW) and skip-gram are options for training algorithms in Word2Vec and the default training algorithm in the library is CBOW. In our experiments, we use default CBOW for training Word2Vec.

3.3.4 PCA2Vec

PCA2Vec is another technique used in this research for feature engineering. PMI matrix of 20×20 dimensions is calculated as our observation has a maximum of 20 distinct opcodes. For consistency with the HMM2Vec experiment discussed above, we used the two eigenvectors and for consistency with the Word2Vec model above, we used the window size of W as 10 when constructing the PMI matrix. The resulting projection into the eigenspace is 2×20 which we vectorize to obtain the feature of length 40.There is no library available to train PCA2Vec hence implemented python code to generate PMI matrix as discussed in Section 2.2.3.2.

3.3.4.1 PCA2Vec Variation

In trained PCA, the eigenvector corresponding to bigger eigenvalues is most influential. We experimented by eliminating eigenvectors with bigger eigenvalues. Research claims the elimination of bigger eigenvalue can give better projection [28]. In this experiment, we select two eigenvectors to project features into 2-d space.

- Case 1: PCA2Vec with no elimination
- Case 2: PCA2Vec with the elimination of first eigenvalue
- Case 3: PCA2Vec with the elimination of first and second eigenvalue

These versions of features are used for classification using MLP classifier after finding the best parameters in GridSearch.

Accuracy in Figure 5 shows that elimination of eigenvector corresponding to bigger eigenvalues do not help to improve performance. Rather, it reduces the accuracy which indicates that eigenvectors corresponding to bigger eigenvalues have more information.

3.3.5 *k*-PCA

There is another novel approach we introduced called as *k*-PCA for feature engineering. This is a two level feature engineering method where the first level can be

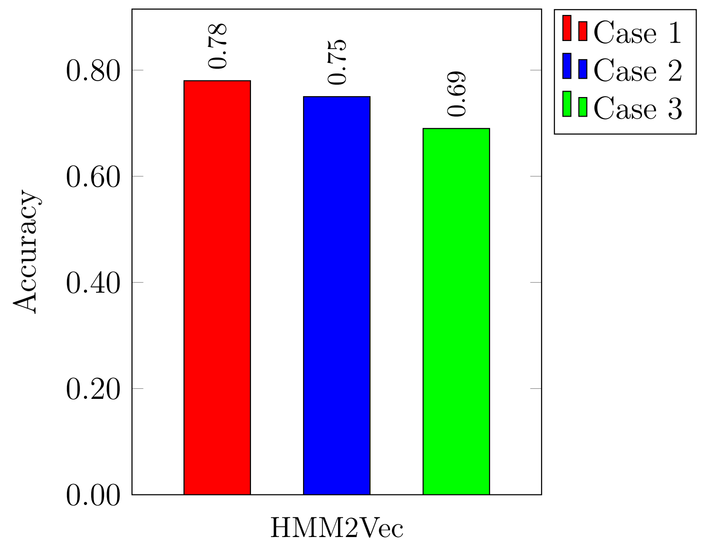


Figure 5: Regular PCA2Vec and Variants Accuracy

any feature engineering method to generate level 1 of the feature vector. The proposed method reduces the dimensionality of the feature vector from level 1 to the number of classes in the dataset. Figure 6 shows the process involved in feature engineering using k-PCA. We will discuss the steps involved in feature reduction by the k-PCA method.

Step 1 Generate the level 1 features for samples. In our experiments, level 1 can be HMM2Vec, PCA2Vec, or Word2Vec.

Step 2 Divide the data into a training set and feature engineering set. In our experiments, we divide each family into 500 samples for training and 500 samples for feature engineering.

Step 3 Train individual PCA model for classes in dataset using training data. In our experiments, we train 7 PCA model which belongs to 7 malware families.

Step 4 Project samples from feature engineering set into eigenspace of trained PCA models from Step 3.

Step 5 Calculate the score of a projected sample in the eigenspace of each class.

Step 6 Create a feature vector with scores from Step 5. Feature vector created in this space has a length equal to the number of classes in the dataset. In our experiments, feature vector is of length 7.

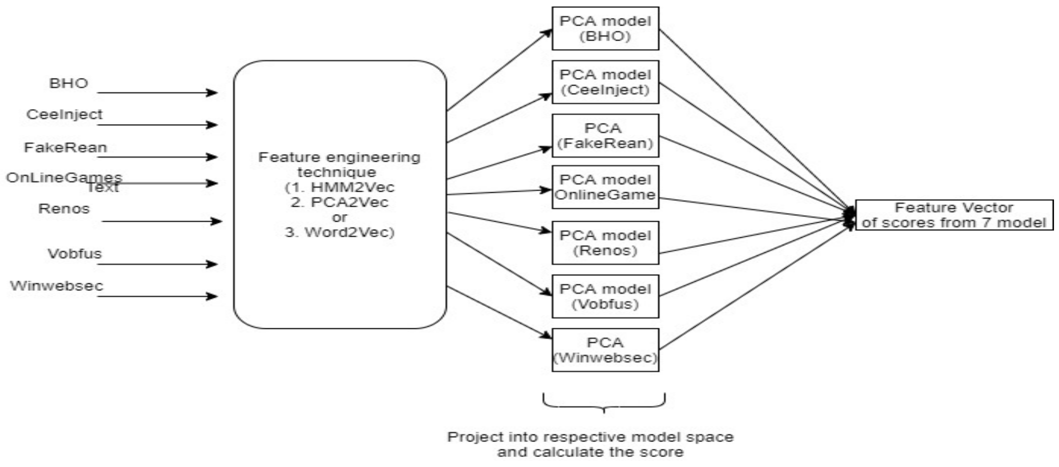


Figure 6: k-PCA Feature Generation

3.3.5.1 Binary PCA Classifier

Before experimenting with k-PCA, experiments on binary PCA classifier are conducted which supported use of k-PCA technique. In binary PCA experiment, we train the PCA model using 500 samples from one family. The test set consists of 500 positive samples of same family and 500 negative samples of another family.

After training the PCA, we project the test sample into eigenspace and calculate the minimum euclidean distance from vectors in projected space as a score. Projection can be tuned by selecting the number of eigenvectors. We experimented by training the model on one of the family and tested individually against the remaining 6 families. In each pair of test and train family, we experimented with eigenvectors in range 1 to 20. This results in a total of $7 \cdot 6 \cdot 20 = 840$ experiments. For all these experiments, we calculated accuracy and scatterplot, out of which results for an experiment on Winwebsec and OnLineGames is shown in Figure 7, 8 and 9.

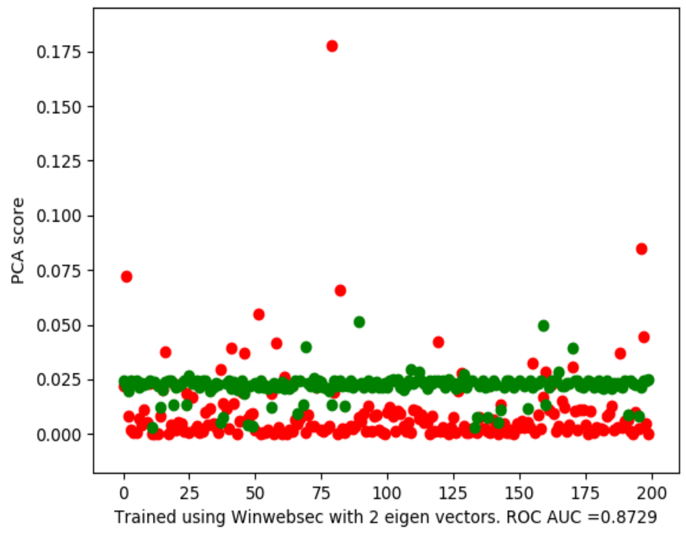


Figure 7: Scatterplot for PCA Trained on Winwebsec and Tested on OnLineGames with 2 Eigenvectors

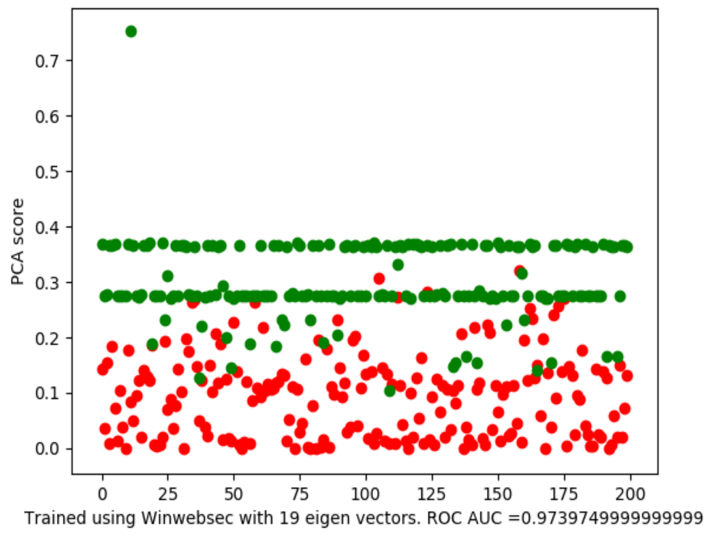


Figure 8: Scatterplot for PCA Trained on Winwebsec and Tested on OnLineGames with 19 Eigenvector

As we can see in Figure 7, 8 and 9, increasing the number of eigenvectors in eigenspace helps to get a clear boundary to classify the sample. Also, it shows that

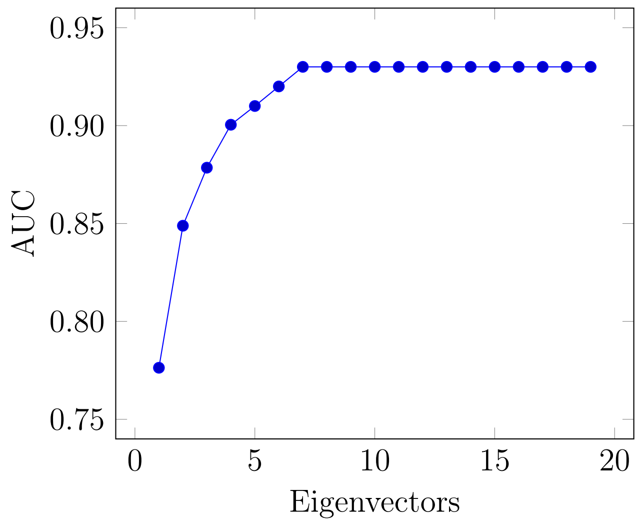


Figure 9: AUC vs Number of Eigenvectors

PCA is an effective binary classifier. This forms the basis for using multiple PCA models in k-PCA feature engineering technique.

3.3.6 Feature Vector Length

So far we have discussed different techniques to generate the feature vector. The length of the feature vector is decided based on different parameters. Here, we discuss the relation of the length of feature vectors and the parameters in the respective technique.

3.3.6.1 HMM baseline

The feature vector length is decided based on the number of classes. In our experiments, the length of the feature vector is 7 because of a number of malware families.

3.3.6.2 HMM2Vec

The feature vector length is based on the number of distinct observation symbols and the number of states used to train the HMM. The length of the feature vector is the product of a number of states and the number of observation symbols.

3.3.6.3 PCA2Vec

The feature vector length is decided based on distinct observation symbols and the number of significant eigenvectors selected in eigenspace. The resulting length of the feature vector is a product of both these parameters.

3.3.6.4 k-PCA

The special part about this technique is the reduction of feature vector length from level 1. The length is independent of feature vector in level 1. Any size of input feature vector length can be reduced to a feature vector of length k, where k is a number of the classes.

3.4 Effect of Embedding Length

In our main experiments of PCA2Vec and Word2Vec discussed in Section 3.3, word embedding is of length 2 for consistency with HMM2Vec. However, it is essential to know if increasing the size of the embedding vector can result in more information. We generated embedding vectors of length 20 and window size of 50 for Word2Vec and PCA2Vec. Results are compared with the vector length 2 and window size 10. These experiments incorporate k-PCA to reduce the dimension of the feature vector.

We observed that there is no significant improvement in accuracy when increasing the length of the feature vector in Figure 10, and the slight improvement comes with the overhead of dealing with a longer feature vector. Also, the small vector length does not harm the contextual information, and accuracy is not reduced.

3.5 Classification

The quality of features generated is determined by classification based on features. We experimented with multi-class classification to assess the feature quality. A good feature is supposed to yield better results compared to other features. In the classification experiment, we employ SVM, MLP, kNN, and RF. We use the

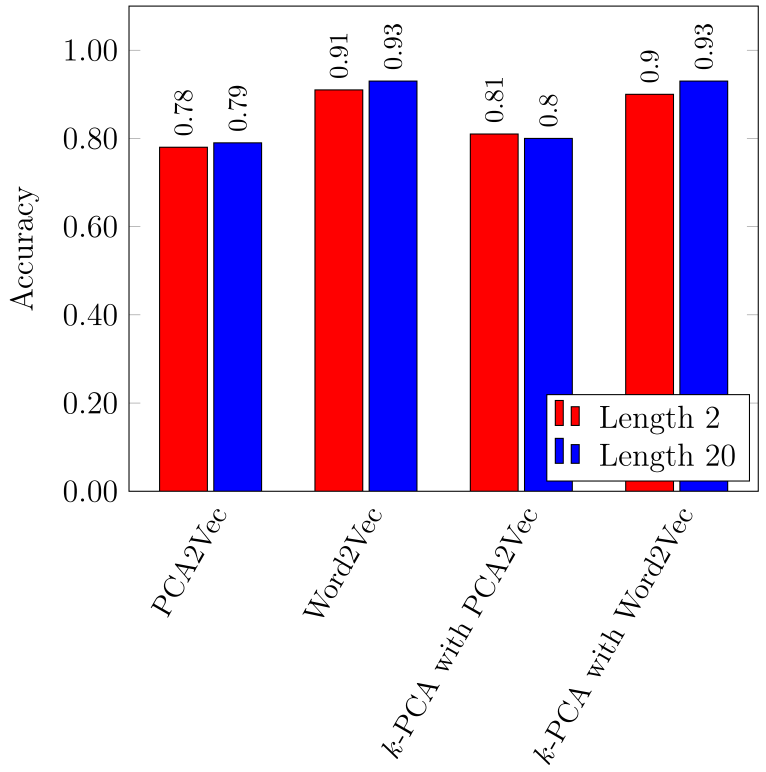


Figure 10: Effect of Embedding Length

scikitlearn [49] library for all the classifiers. Classification experiments involve two phases. First, to find the best suitable parameters for training the classifier and the second phase is training the classifier.

3.5.1 GridSearch Phase

To determine the best possible parameters for a given combination of feature and classifier, the GridSearch module from scikitlearn is used. Data is divided into a training set and test set using a train test split module. This module randomly splits data to 20% as a test set and 80% as a training set. When dividing the data, this module selects an equal number of samples from each family.

The training set is used to perform GridSearch operation on all possible combinations of parameters for a given classifier. When finding the best parameters, GridSerach divides the training data into 5 folds. GridSearch trains and tests the model on those 5 folds and finds the best parameter. The best parameter is decided based on the combination which gives the highest average of accuracy in each fold. This phase determines the best suited parameter for a given classifier and feature. Parameters tested in this phase are listed in Table 6

Table 6: Classifier Hyperparameters Tested

Classifier	**Hyperparameter**	**Tested values**
MLP	`learning_rate`	`constant`, `invscaling`, `adaptive`
	`hidden_layer_size`	$[(30,30,30),(10,10,10)]$
	`solver`	`sgd`, `adam`
	`activation`	`relu`, `logistic`, `tanh`
	`max_iter`	$[10000]$
SVM	`kernel`	`rbf`, `linear`
	`C`	$[1,10,100,1000]$
	`gamma` (`rbf` only)	$[0.001,0.0001]$
kNN	`n_neighbors`	$[3,5,11,19]$
	`weights`	`uniform`, `distance`
	`p`	`manhatten`, `euclidean`
RF	`n_estimators`	$[30,100,500,1000]$
	`max_depth`	$[5,8,15,25,30]$
	`min_samples_split`	$[2,5,10,15,100]$
	`min_samples_leaf`	$[1,2,5,10]$

3.5.1.1 GridSearch Results

In this section, we discuss the results of our GridSearch. Table 7 and 8 represents the best parameters found.

The parameters tested are listed in Table 6. Observe that for each of the three different word embedding techniques, three k-PCA techniques, and one baseline technique, we tested 36 combinations of parameters for MLP, 12 combinations for SVM, 16 combinations for kNN, and 400 RF combinations. Overall, we conducted

$$7 \cdot (36 + 12 + 16 + 400) = 3248$$

experiments to determine the parameters for the remaining experiments.

Table 7: Classifier Hyperparameters Selected for HMM2Vec, PCA2Vec and Word2Vec

Classifier	**Hyperparameter**	**HMM2Vec**	**Word2Vec**	**PCA2Vec**	**HMM Baseline**
MLP	`learning_rate`	`invscaling`	`constant`	`adaptive`	`constant`
	`hidden_layer_size`	$(30, 30, 30)$	$(30, 30, 30)$	$(30, 30, 30)$	$(30, 30, 30)$
	`solver`	`adam`	`adam`	`sgd`	`adam`
	`activation`	`relu`	`relu`	`relu`	`relu`
	`max_iter`	10000	10000	10000	10000
SVM	`kernel`	`linear`	`rbf`	`rbf`	`rbf`
	`C`	1000	1000	1000	10
	`gamma`	NA	0.001	0.001	0.0001
kNN	`n_neighbors`	3	3	3	3
	`weights`	`distance`	`distance`	`distance`	`distance`
	`p`	`manhatten`	`euclidean`	`manhatten`	`manhatten`
RF	`n_estimators`	100	500	1000	1000
	`max_depth`	25	30	30	30
	`min_samples_split`	2	2	2	2
	`min_samples_leaf`	1	1	1	1

Table 8: Classifier Hyperparameters Selected for k-PCA Features

Classifier	**Hyperparameter**	**HMM2Vec**	**Word2Vec**	**PCA2Vec**
MLP	`learning_rate`	`invscaling`	`constant`	`constant`
	`hidden_layer_size`	$(30, 30, 30)$	$(30, 30, 30)$	$(30, 30, 30)$
	`solver`	`adam`	`sgd`	`adam`
	`activation`	`relu`	`tanh`	`tanh`
	`max_iter`	10000	10000	10000
SVM	`kernel`	`linear`	`rbf`	`rbf`
	`C`	1000	1000	1000
	`gamma`	NA	0.001	0.001
kNN	`n_neighbors`	19	5	11
	`weights`	`distance`	`distance`	`distance`
	`p`	`euclidean`	`euclidean`	`manhatten`
RF	`n_estimators`	100	500	1000
	`max_depth`	25	15	25
	`min_samples_split`	5	5	2
	`min_samples_leaf`	2	1	2

The optimal parameters selected for each classifier and for each embedding technique are listed in Table 7 and Table 8. It is observed that overall there is considerable agreement between the parameters for the different word embedding techniques, but in two cases (`learning_rate` and `n_estimators`), a different parameter is selected for each of the three embedding techniques. In k-PCA version of the feature, different parameters selected for most cases.

3.5.2 Results

After deciding the best parameters as discussed in Section 3.5.1, we train the model using training data with best parameters. We calculate the accuracy of the trained model by classifying the test data. Test data is not used in GridSearch to avoid biased in finding best parameters. Also, cross-validation in the GridSearch phase reduces possibility of overfitting the model.

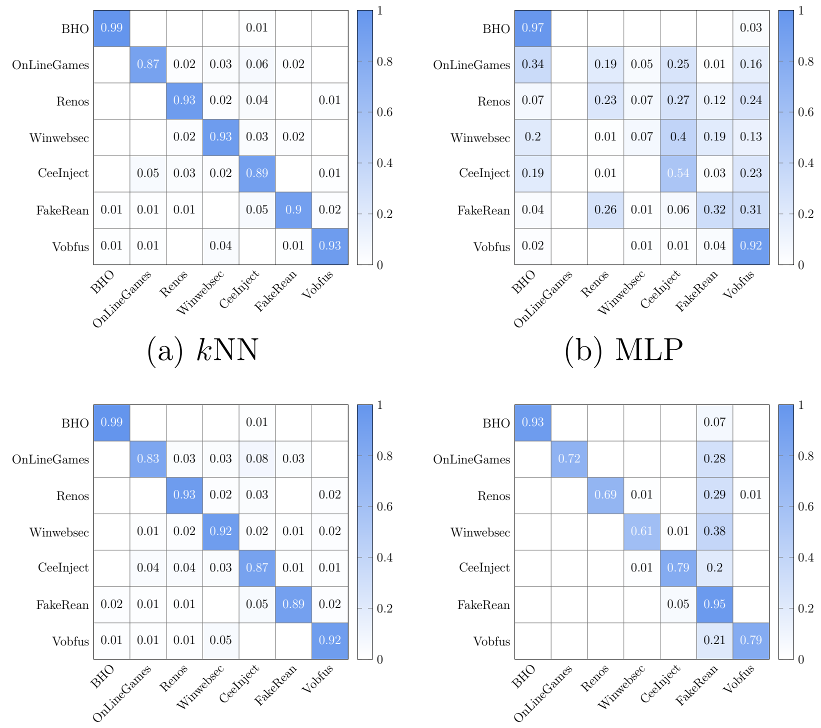
The confusion matrices are generated for a combination of feature vectors and classifiers. The classification result helps in comparing the quality of feature vector and the performance of different classifiers. In this section, we discuss the results of classification for the feature engineering techniques discussed in Section 3.3.

3.5.2.1 HMM Baseline

The confusion matrices for baseline HMM experiments are given in Figure 11 The accuracy achieved for kNN, MLP, RF, and SVM are 0.92, 0.44, 0.91, and 0.78, respectively. We observe that MLP and SVM both perform poorly, whereas the neighborhood-based techniques, namely, kNN and RF, are both strong, considering that we have 7 classes. Also, kNN and RF give very similar results.

3.5.2.2 HMM2Vec Results

From the confusion matrices in Figure 12, we infer that the greatest source of misclassifications is between FakeRean and Winwebsec families. In many—but not all—of our subsequent experiments, these two families will prove the most challenging to distinguish.



(a) kNN (b) MLP

(c) RF (d) SVM

Figure 11: Confusion Matrices for HMM Baseline Experiments

3.5.2.3 PCA2Vec Results

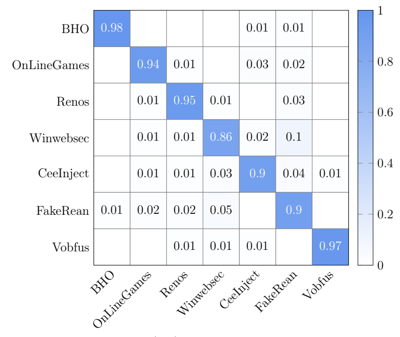
In the confusion matrices in Figure 13, we give the overall accuracy for each of our experiments. The results show that PCA2Vec performed poorly for each of the classifiers, as compared to HMM2Vec.

3.5.2.4 Word2Vec

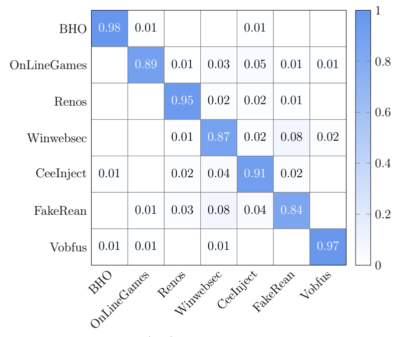
Analogous to the HMM2Vec and PCA2Vec experiments above, we classify 7000 feature vectors using four classifiers. The confusion matrices for these experiments are given in Figure 14. From the results in Figure 14, we can infer that the RF seems to perform particularly well.

3.5.2.5 *k*-PCA Results

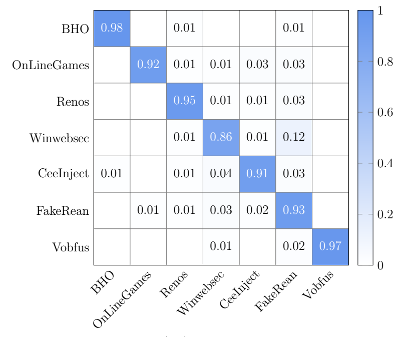
In the k-PCA experiment, level 1 of the feature can be HMM2Vec, Word2Vec, and PCA2Vec. Confusion matrices in Figure A.24, A.25, and A.26 in the appendix



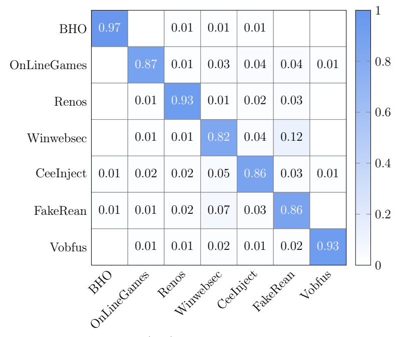
(a) kNN



(b) MLP



(c) RF



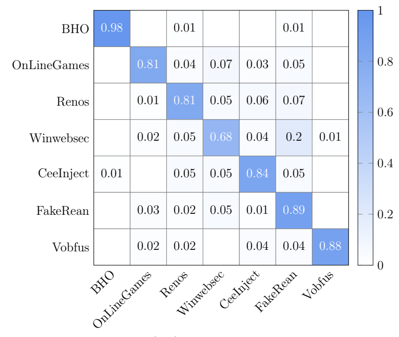
(d) SVM

Figure 12: Confusion Matrices for HMM2Vec Experiments

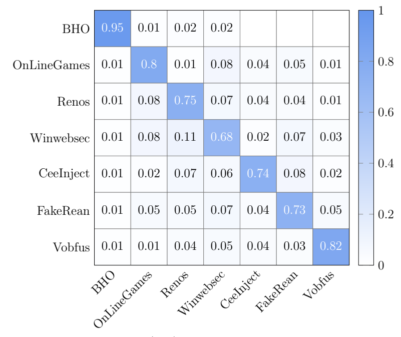
represent the classification result for three different experiments of k-PCA based on level 1 feature. The accuracy of the k-PCA method with the respective level 1 feature can be compared in Figure 15. The experiments with the reduced feature vector using k-PCA show approximately similar results compared to respective word embedding feature vector. The comparison shows that k-PCA does not reduce the accuracy after the reduction of dimensions in the feature vector.

3.5.2.6 Discussion

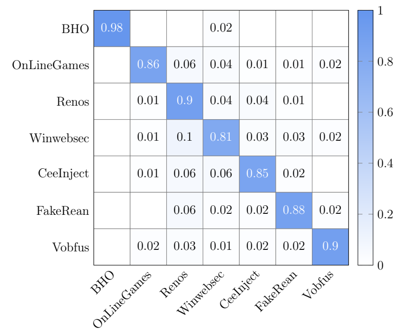
In this section, we discuss the results of all the feature engineering techniques. Figure 15 gives the overall accuracy for each of our multi-class experiments using kNN, MLP, RF, and SVM classifiers, for each of the HMM baseline, HMM2Vec, PCA2Vec, Word2Vec, and k-PCA derived features. From these 28 distinct results, we see that HMM2Vec and Word2Vec perform equally well, with PCA2Vec lagging far



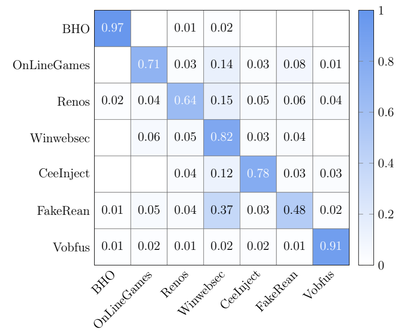
(a) kNN



(b) MLP



(c) RF



(d) SVM

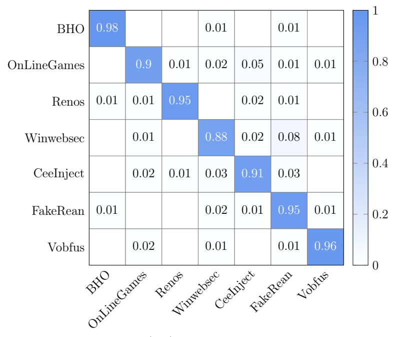
Figure 13: Confusion Matrices for PCA2Vec Experiments

behind. This relation between feature embedding techniques holds after dimensionality reduction using k-PCA.

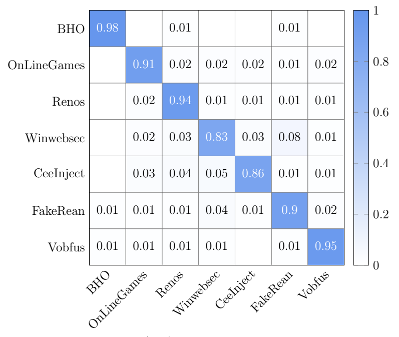
We also see that the neighborhood-based classifiers, namely, RF and kNN, perform better than SVM and MLP classifiers. In general, we expect that RF and kNN would perform similarly to each other, and that SVM and MLP would perform similarly as well. Another observation is k-PCA level of feature engineering gives almost same results as level 1. However, the length of the feature vector in k-PCA is less than the level 1 feature. This shows that k-PCA reduces the dimension of the feature vector without affecting the performance.

3.6 Overfitting in kNN and RF

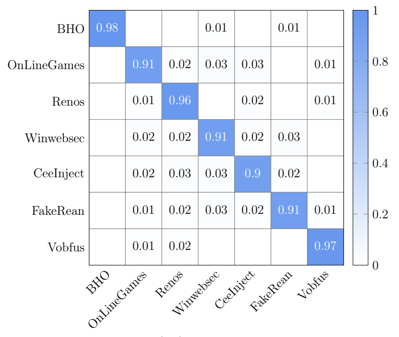
We performed additional experiments on the baseline HMM, HMM2Vec, and Word2Vec features to understand the effect of changing the parameters in kNN and



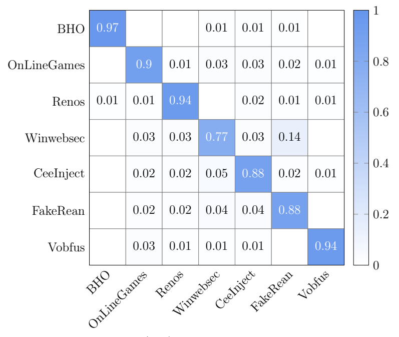
(a) kNN



(b) MLP



(c) RF



(d) SVM

Figure 14: Confusion Matrices for Word2Vec Experiments

RF. For both the experiments, we performed 10 fold cross-validation and used the average accuracy of all folds to smooth any bias that might appear in the results for the various folds. For our kNN experiments, we tested the number of nearest neighbors k in the range 1 to 150 with a step size of 5. For our RF experiments, we trained RF with the maximum depth of the tree in the range 1 to 30, and the number of trees ranging from 1 to 500, with a step size of 5 for both parameters.

Figure 16, 18, and 17 show the results for these experiments. RF results in Figure 17 and Figure 18 are truncated to make graph readable. Based on the results in Figure 15, the baseline HMM feature performs poorly in MLP and SVM classification and better with RF and kNN classifier. Features such as Word2Vec and HMM2Vec perform better for all classifiers and we can consider it as strong features compared to baseline HMM.

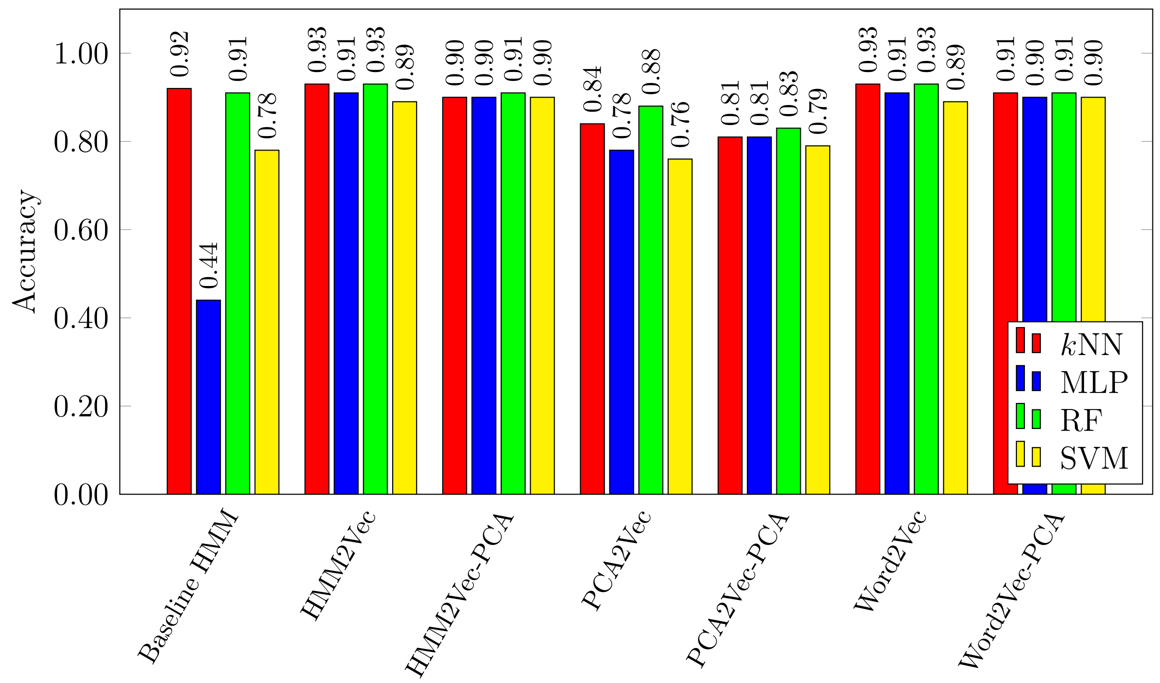


Figure 15: Accuracy for Various Features and Classifiers

In Figures 7 and 8, it is observed that GridSearchCV chooses a small k in kNN, a large number of estimators, and a large max depth of tree for RF in most cases. This leads to analyze the effects of parameters in kNN and RF against strong and weak features. This experiment will help us to identify if there are any potential over-fitting issues in kNN and RF for weak features.

In Figure 16, it is observed that the increasing value of k in kNN for weak feature increases the misclassification rate drastically. The graph is steeper for the HMM baseline than HMM2Vec and Word2Vec. Similarly, in Figure 17 and Figure 18, 3d graphs shows that the increasing number of trees for short tress is helping to reduce the misclassification rate more effectively with HMM2Vec features compared to the baseline HMM feature. Decision boundary for a trained model should be smooth if learning (As opposed to memorizing) is taking place. HMM baseline results with kNN and RF show irregular boundary as changing the training parameters radically

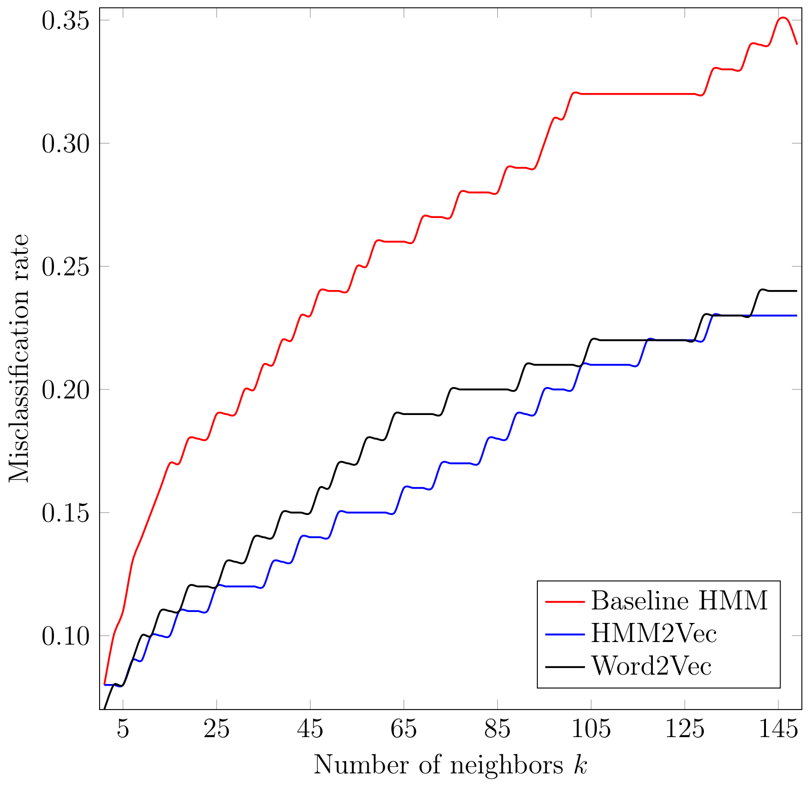


Figure 16: kNN Results as a Function of k

changes the misclassification rate. Hence we can say that classifier for HMM baseline features is overfitting for small values of k in kNN and large number of trees in RF. Experiments based on testing with robust samples can give more insights on our claim.

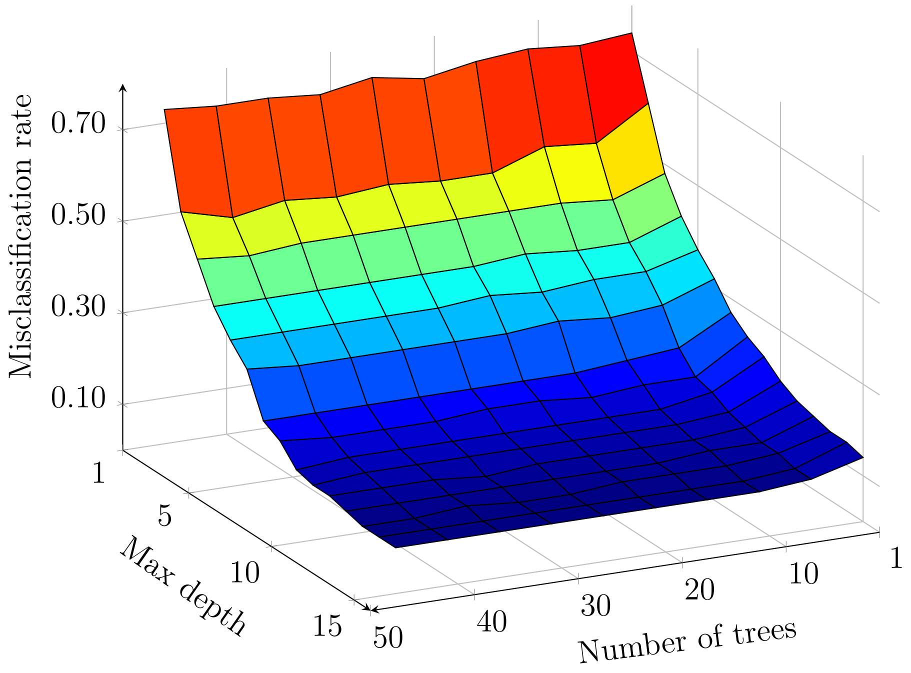


Figure 17: RF Results as a Function of Number of Trees and Max Depth for HMM Baseline Features

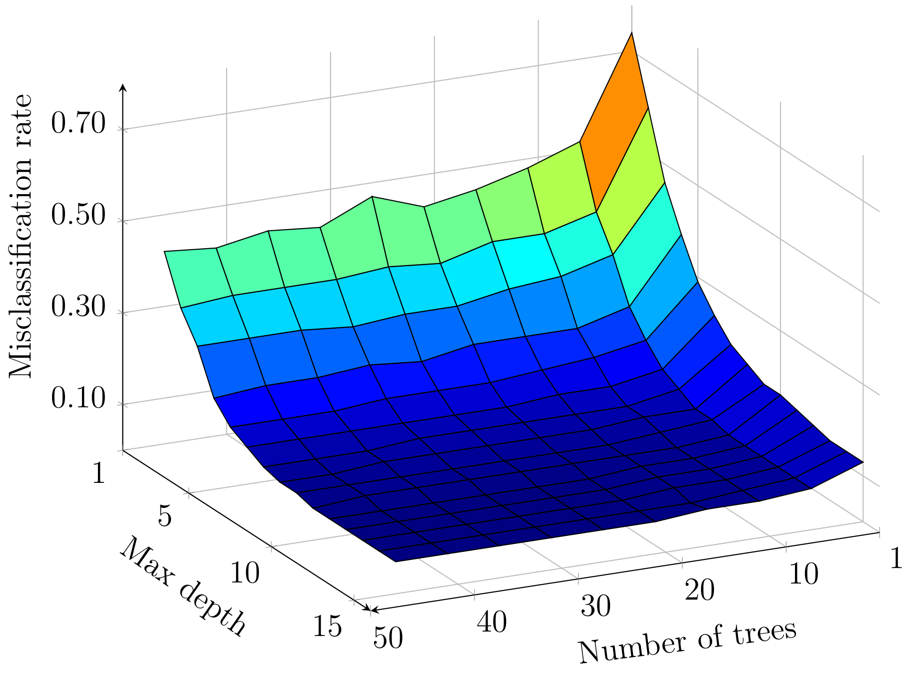


Figure 18: RF Results as a Function of Number of Trees and Max Depth for HMM2Vec Features

CHAPTER 4

Conclusion and Future Work

In this research, we conducted experiments to understand the significance of word embedding techniques for feature engineering. We considered word embedding techniques, including Word2Vec, HMM2Vec, and PCA2Vec. We used opcode sequence as a basic feature of malware and considered 7 different malware families for classification, with a substantial number of samples for each family. We extracted the opcode using the objdump tool in Linux and filtered the most important opcodes from samples. We also used a balanced dataset with 1000 samples in each family to avoid biased in experiments.

In this paper, we have presented the results of several experiments using word embedding techniques to generate features for malware family classification. In effect, we have applied machine learning techniques to generate results that are subsequently used as features for additional machine learning techniques. Such a concept is not entirely unprecedented as, for example, PCA is often used to reduce the dimensionality of data before applying other machine learning techniques. However, the authors are not aware of previous work involving the use of word embedding techniques in the manner considered in this work.

Our results show that word embedding techniques can be used to generate features that are more informative than the original data. This process of distilling useful information from the data before classifying samples is potentially useful, not only in the field of malware analysis but also in other fields where learning plays a prominent role. The experiments show that random restarts in HMM2Vec are effective in learning of word embeddings. In a comparative study of word embedding techniques, we conclude that HMM2Vec and Word2Vec perform similar whereas PCA2Vec performs poorly.

We also performed binary classification experiments on malware families in our dataset using the PCA model. From this experiment, we derived that the families in the dataset are separable and PCA scores can be used as a feature. Based on the results of PCA as a binary classifier, we proposed k-PCA technique to reduce the dimensionality of the feature vector which no other research has explored before. Results in the experiments show that k-PCA can be used effectively to reduce dimensions of the feature vector. We considered four different classifiers in our experiment SVM, MLP, kNN, and RF for classification and observed that results in SVM and MLP are related.

For future work, it would be interesting to consider other families and different types of malware. It would also be interesting work to use more complex and higher dimensional data—as with dimensionality-reduction techniques, such data would tend to offer more prospects for improvement using the word embedding strategy considered in this paper. Also, the robustness of the models can be analyzed by checking the effect of obfuscation on the word embedding techniques. The word embedding techniques can be used with features such a byte code n-gram, system API calls etc. as we only consider opcode in this study. Another direction to explore is to use the contextual information from word embedding for a task other than classification such as unsupervised clustering. Spherical k-means can be used for clustering based on word embeddings.

Our experiments are done on malware families of different types, hence experiments can be extended on malware families of the same type to understand the patterns within the same type of malware. In all our word embeddings, instead of generating feature vector, images can be generated and a deep learning model can be trained for classification using those images. In our experiments, feature selection is based on the frequency of opcodes. Research can be extended to use advanced feature selection techniques based on TF-IDF and SVM, for example.

LIST OF REFERENCES

[1] Norton. "Norton security center: Malware." https://us.norton.com/internetsecurity-malware.html. 2020.

[2] P. Vinod, R. Jaipur, V. Laxmi, and M. Gaur, "Survey on malware detection methods," in *Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09)*, 2009, pp. 74–79.

[3] P. M. Baggenstoss, "A modified baum-welch algorithm for hidden markov models with multiple observation spaces," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 4, pp. 411–416, 2001.

[4] J. Shlens, "A tutorial on principal component analysis," http://www.cs.cmu.edu/~elaw/papers/pca.pdf, 2005.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," https://arxiv.org/abs/1301.3781, 2013.

[6] Symantec, "Internet security threat report: Malware," https://interactive.symantec.com/istr24-web, 2019.

[7] K. D. Harris and A. General, "California data breach report," https://oag.ca.gov/privacy/databreach/reporting, 2016.

[8] Kaspersky, "Kaspersky security bulletin: The year in figures," https://securelist.com/ksb-overall-statistics-2017/83453/, 2017.

[9] A. Dhammi and M. Singh, "Behavior analysis of malware using machine learning," in *2015 Eighth International Conference on Contemporary Computing (IC3)*. IEEE, 2015, pp. 481–486.

[10] K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh, "Automatic generation of string signatures for malware detection," in *Recent Advances in Intrusion Detection*, E. Kirda, S. Jha, and D. Balzarotti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 101–120.

[11] A. Yewale and M. Singh, "Malware detection based on opcode frequency," *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pp. 646–649, 2016.

[12] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[13] O. Hachinyan, "Detection of malicious software on based on multiple equations of api-calls sequences," in *2017 IEEE Conference of Russian Roung Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 2017, pp. 415–418.

[14] B. Pechaz, M. V. Jahan, and M. Jalali, "Malware detection using hidden markov model based on markov blanket feature selection method," in *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, 2015, pp. 558–563.

[15] Y. Awad, M. Nassar, and H. Safa, "Modeling malware as a language," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[16] I. Popov, "Malware detection using machine learning based on word2vec embeddings of machine code instructions," in *2017 Siberian Symposium on Data Science and Engineering (SSDSE)*. IEEE, 2017, pp. 1–4.

[17] H. L. Duarte-Garcia, A. Cortez-Marquez, G. Sanchez-Perez, H. Perez-Meana, K. Toscano-Medina, and A. Hernandez-Suarez, "Automatic malware clustering using word embeddings and unsupervised learning," in *2019 7th International Workshop on Biometrics and Forensics (IWBF)*. IEEE, 2019, pp. 1–6.

[18] Y. Liu and Y. Wang, "A robust malware detection system using deep learning on api calls," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 1456–1460.

[19] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[20] H. Hashemi, A. Azmoodeh, A. Hamzeh, and S. Hashemi, "Graph embedding as a new approach for unknown malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, pp. 153–166, 2016.

[21] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2177–2185. [Online]. Available: http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf

[22] M. Stamp, "A revealing introduction to hidden Markov models," https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf, 2004.

[23] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989, https://www.cs.sjsu.edu/~stamp/RUA/Rabiner.pdf.

[24] C. Shalizi, "Principal component analysis," https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf, 2012.

[25] Stack Exchange, "Making sense of principal component analysis," https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues, 2015.

[26] M. Stamp, *Introduction to Machine Learning with Applications in Information Security.* Boca Raton: Chapman and Hall/CRC, 2017.

[27] C. Moody, "Stop using word2vec," https://multithreaded.stitchfix.com/blog/2017/10/18/stop-using-word2vec/.

[28] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 211–225, 2015, https://levyomer.files.wordpress.com/2015/03/improving-distributional-similarity-tacl-2015.pdf.

[29] S. Banerjee, "Word2vec — A baby step in deep learning but a giant leap towards natural language processing," https://medium.com/explore-artificial-intelligence/word2vec-a-baby-step-in-deep-learning-but-a-giant-leap-towards-natural-language-processing-40fe4e8602ba, 2018.

[30] C. McCormick, "Word2vec tutorial — The skip-gram model," http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/, 2016.

[31] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf, 2013.

[32] A. Liaw and M. Wiener, "Classification and regression by randomForest," *R news*, vol. 2/3, pp. 18–22, 2002.

[33] F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, no. 5, pp. 183 – 197, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0925231291900235

[34] S. Basole, F. Di Troia, and M. Stamp, "Multifamily malware models," *Journal of Computer Virology and Hacking Techniques*, pp. 1–14, 2020.

[35] M. Wadkar, F. D. Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Systems with Applications*, vol. 143, 2020.

[36] Microsoft Security Intelligence, "Trojan:win32/bho," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO&threatId=-2147364778, 2010.

[37] Microsoft Security Intelligence, "CeeInject," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject, 2007.

[38] Microsoft Security Intelligence, "Win32/fakerean," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/FakeRean, 2010.

[39] Microsoft Security Intelligence, "Pws:win32/onlinegames," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FOnLineGames, 2010.

[40] Microsoft Security Intelligence, "Trojandownloader:win32/renos," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Renos&threatId=16054, 2010.

[41] Microsoft Security Intelligence, "Vobfus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fvobfus, 2010.

[42] Microsoft Security Intelligence, "Winwebsec," https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec, 2010.

[43] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden Markov models for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59–73, 2015.

[44] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Exploring hidden Markov models for virus analysis: A semantic approach," in *46th Hawaii International Conference on System Sciences, HICSS 2013, Wailea, HI, USA, January 7-10, 2013.* IEEE Computer Society, 2013, pp. 5039–5048. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6479598

[45] A. Kalbhor, T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Dueling hidden Markov models for virus analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 103–118, 2015.

[46] A. Raghavan, F. D. Troia, and M. Stamp, "Hidden Markov models with random restarts versus boosting for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 2, pp. 97–107, 2019.

[47] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[48] R. Řehůřek and P. Sojka, “Software framework for topic modelling with large corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.* Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, “Scikit-learn: Machine learning in python,” *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

APPENDIX

Appendix

A.1 Additional Results

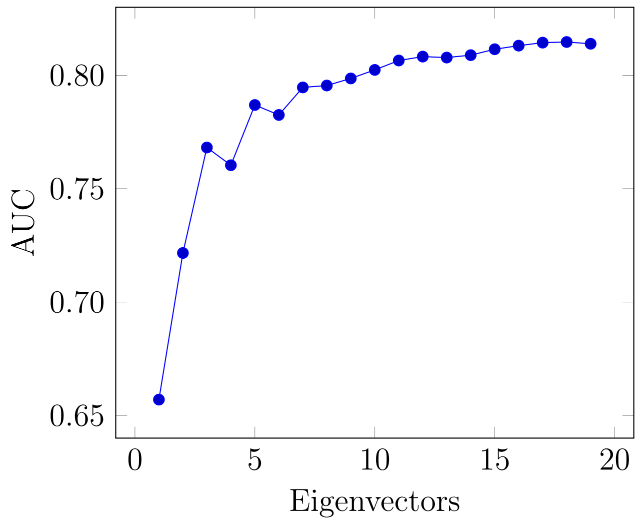


Figure A.19: AUC vs Number of Eigenvectors for Top 20 Opcodes in PCA Binary Classification

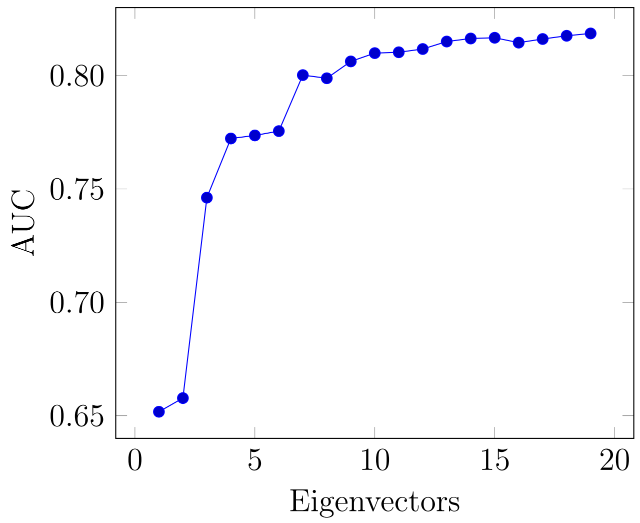
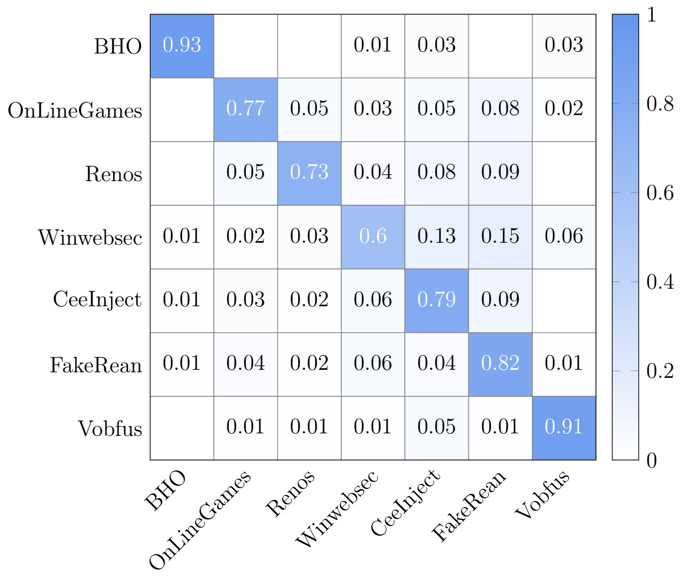
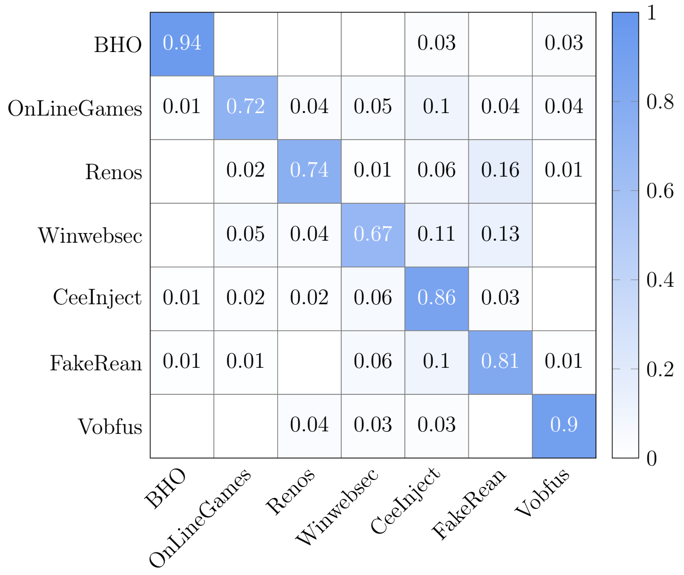


Figure A.20: AUC vs Number of Eigenvectors for Top 30 Opcodes in PCA Binary Classification



	BHO	OnLineGames	Renos	Winwebsec	CeeInject	FakeRean	Vobfus
BHO	0.93			0.01	0.03		0.03
OnLineGames		0.77	0.05	0.03	0.05	0.08	0.02
Renos		0.05	0.73	0.04	0.08	0.09	
Winwebsec	0.01	0.02	0.03	0.6	0.13	0.15	0.06
CeeInject	0.01	0.03	0.02	0.06	0.79	0.09	
FakeRean	0.01	0.04	0.02	0.06	0.04	0.82	0.01
Vobfus		0.01	0.01	0.01	0.05	0.01	0.91

Figure A.21: Confusion Matrices for HMM2Vec with No Random Restart and No State Swap



	BHO	OnLineGames	Renos	Winwebsec	CeeInject	FakeRean	Vobfus
BHO	0.94				0.03		0.03
OnLineGames	0.01	0.72	0.04	0.05	0.1	0.04	0.04
Renos		0.02	0.74	0.01	0.06	0.16	0.01
Winwebsec		0.05	0.04	0.67	0.11	0.13	
CeeInject	0.01	0.02	0.02	0.06	0.86	0.03	
FakeRean	0.01	0.01		0.06	0.1	0.81	0.01
Vobfus			0.04	0.03	0.03		0.9

Figure A.22: Confusion Matrices for HMM2Vec with State Swap and B Matrix

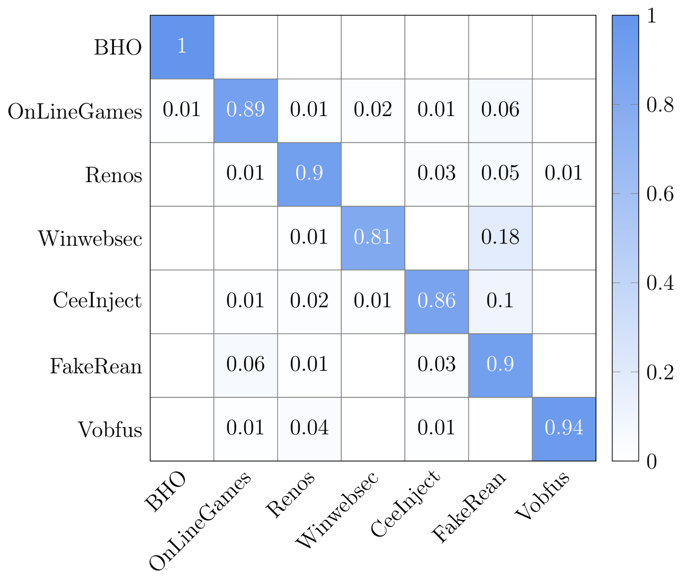
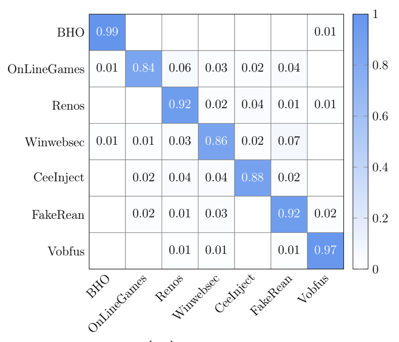
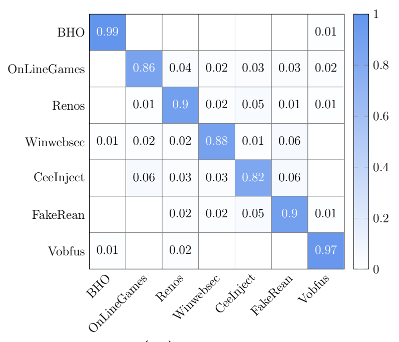


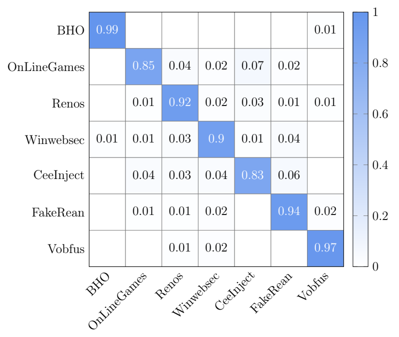
Figure A.23: Confusion Matrices for HMM2Vec with Random Restarts



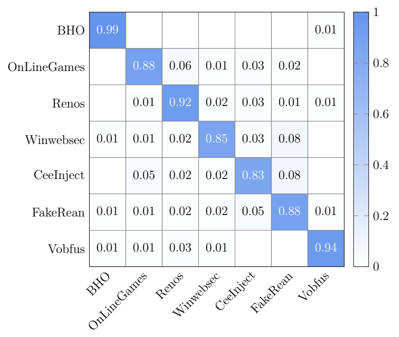
(a) kNN



(b) MLP

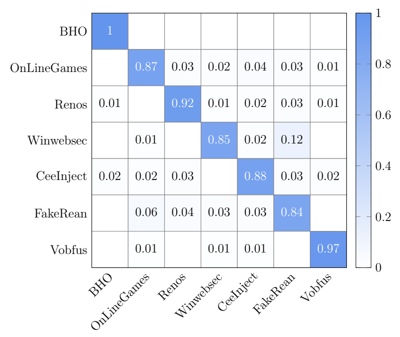


(c) RF

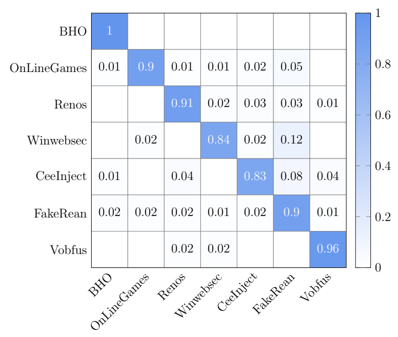


(d) SVM

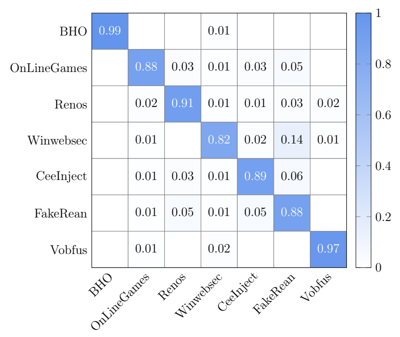
Figure A.24: Confusion Matrices for k-PCA with Word2Vec



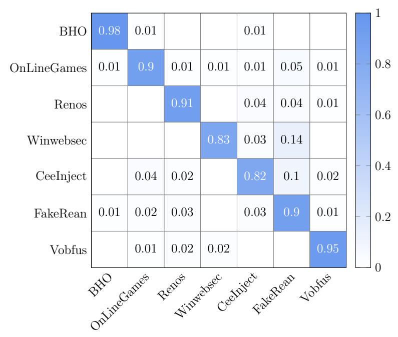
(a) kNN



(b) MLP



(c) RF



(d) SVM

Figure A.25: Confusion Matrices for k-PCA with HMM2Vec

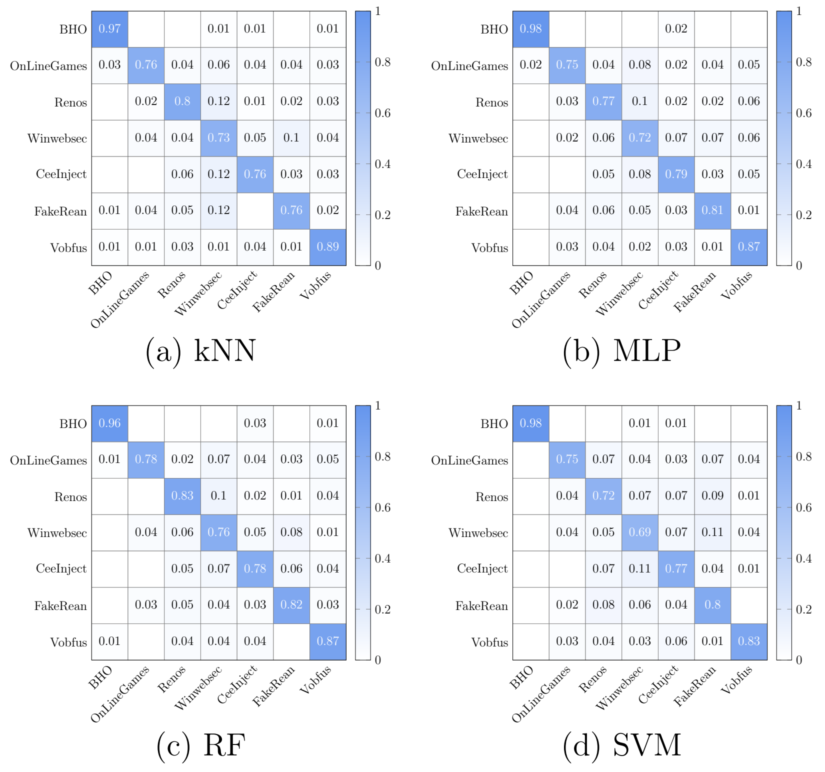


Figure A.26: Confusion Matrices for *k*-PCA with PCA2Vec