Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-022-00424-3

INVITED PAPER

Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo

Aparna Sunil Kale[1] · Vinay Pandya[1] · Fabio Di Troia[1] · Mark Stamp[1]

Received: 15 October 2021 / Accepted: 18 March 2022

Abstract

Malware classification is an important and challenging problem in information security. Modern malware classification techniques rely on machine learning models that can be trained on features such as opcode sequences, API calls, and byte n-grams, among many others. In this research, we consider opcode features and we implement machine learning techniques, where we apply word embedding techniques—specifically, Word2Vec, HMM2Vec, BERT, and ELMo—as a feature engineering step. The resulting embedding vectors are then used as features for classification algorithms. The classification algorithms that we employ are support vector machines (SVM), k-nearest neighbor (kNN), random forests (RF), and convolutional neural networks (CNN). We conduct substantial experiments involving seven malware families. Our experiments extend beyond previous related work in this field. We show that we can obtain slightly better performance than in comparable previous work, with significantly faster model training times.

1 Introduction

Malware is a software that is created with the intent to cause harm to computer data or otherwise adversely affect computer systems [1]. Detecting malware can be a challenging task, as there exist a wide variety of advanced malware that employ various anti-detection techniques.

Some advanced types of malware include polymorphic and metamorphic code. Such code enables attackers to easily generate vast numbers of new malware variants that cannot be detected using traditional techniques [1]. According to McAfee, some 60 million new malware samples were created in the first quarter of the year 2019 [2]. Malware detection—and the closely related problem of malware classification—are inherently challenging in such an environment.

Methods used to protect users from malware include signature-based and behavioral-based detection. Signature-based detection relies on specific patterns found in malware samples. In contrast, behavioral-based detection focuses on actions performed by code. Signature-based techniques cannot detect new malware variants, while behavioral-based techniques often results in a high false-positive rate. Both of these traditional malware detection strategies fail when confronted with advanced forms of malware [27]. Therefore, it is essential to consider alternative malware detection techniques.

Modern malware research often focuses on machine learning, which has shown better performance as compared to traditional methods, particularly in the most challenging cases. Machine learning models for malware classification can be trained on a wide variety of features, including API calls, opcodes sequences, system calls, and control flow graphs, among many others [5].

In this research, we consider four word embedding techniques, namely, HMM2Vec, Word2Vec, BERT, and ELMo. We employ each of these word embedding techniques as a form of feature engineering. For each embedding technique, we consider four classification algorithms: k-nearest neighbors (kNN), support vector machines (SVM), random forest (RF), and convolutional neural networks (CNN). We refer to each of these 16 techniques using the shorthand provided in Table 1.

For each of these techniques, extensive malware classification experiments are conducted over a set of seven challenging malware families. We use opcode sequences as the basic features from which the engineered word embedding features are obtained.

The work in this paper builds on that in [8], where the word embedding techniques of Word2Vec and HMM2Vec were successfully used as feature engineering steps in the malware classification problem. Here, we extend the results presented in [8] by also considering the embedding tech-

✉ Mark Stamp
mark.stamp@sjsu.edu

1 Department of Computer Science, San Jose State University, San Jose, CA, USA

Table 1 The machine learning techniques considered in this paper

Word embedding	Classifier			
	*k*NN	SVM	RF	CNN
HMM2Vec	HMM2Vec-*k*NN	HMM2Vec-SVM	HMM2Vec-RF	HMM2Vec-CNN
Word2Vec	Word2Vec-*k*NN	Word2Vec-SVM	Word2Vec-RF	Word2Vec-CNN
BERT	BERT-*k*NN	BERT-SVM	BERT-RF	BERT-CNN
ELMo	ELMo-*k*NN	ELMo-SVM	ELMo-RF	ELMo-CNN

niques known as BERT and ELMo, which are compared to the results obtained using Word2Vec and HMM2Vec. Note that Word2Vec is a popular technique in natural language processing, but it had not previously been used in the malware context. We chose BERT and ELMo since they provide context-dependent embeddings, unlike Word2Vec and HMM2Vec, which provide a fixed embedding for all occurrences of a "word."

The remainder of this paper is organized as follows. In Sect. 2 we provide a discussion of relevant background topics, with a focus on the machine learning techniques employed in this research. We also provide a selective survey of related work. Section 3 covers our feature engineering approach based on word embeddings that forms the focus of this research. In this section, we also provide information on the dataset that we have used. Section 4 gives our experimental results and analysis. Finally, Sect. 5 summarizes our results and includes a discussion of possible directions for future work.

2 Background

In this section, we introduce the machine learning models used in this research. We also provide a selective survey of relevant previous work.

2.1 Word embedding techniques

In natural language processing (NLP), word embeddings are an essential tool for dealing with text. As discussed below, a useful word embedding does much more than simply assigning arbitrary numbers to words. Instead, word embeddings capture some significant aspects of the semantics of the language. But, word embeddings are not limited to NLP applications, as such techniques can be applied to virtually any features. It is reasonable to expect that such embeddings will contain important information about relationships between features. This is the motivation for using word embeddings as a feature engineering step in the malware classification problem. Specifically, we generate word embeddings based on mnemonic opcodes, and we conduct extensive experiments to determine the utility of such embeddings.

2.1.1 HMM2Vec

A hidden Markov model (HMM) is a probabilistic machine learning algorithm that can be used for pattern matching applications in such diverse areas as speech recognition [26], human activity detection [29], and protein sequencing [12]. HMMs have also proven useful for malware analysis [33].

A discrete HMM is defined as $\lambda = (A, B, \pi)$, where A is the state transition matrix for the underlying Markov process, B contains probability distributions that relate the hidden states to the observations, and π is the initial state distribution. All three of these matrices are row stochastic.

In this research, we focus on the B matrix of trained HMMs. These matrices can be viewed as representing crucial statistical properties of the observation sequences that were used to train the HMM. Using these B matrices as input to classifiers is an advanced form of feature engineering, whereby information in the original features is distilled into a potentially more informative form by the trained HMM. We refer to the process of deriving these HMM-based feature vectors as HMM2Vec. We have more to say about generating HMM2Vec features from HMMs in Sect. 4.2.

2.1.2 Word2Vec

Word2Vec has recently gained considerable popularity in the field of natural language processing (NLP) [21]. This word embedding technique is based on a shallow neural network, with the weights of the trained model serving as embedding vectors—the trained model itself serves no other purpose. These embedding vectors capture significant relationships between words in the training set. Word2Vec can also be used beyond the NLP context to model relationships between more general features or observations.

When training a Word2Vec model, we must specify the desired vector length, which we denote as N. Another key parameter is the window length W, which represents the width of a sliding window that is used to extract training samples from the data. Algebraic properties hold for Word2Vec embeddings.

For example, suppose that we train a state-of-the-art Word2Vec model on English text. Further, suppose that we let

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

and we define $V(w_i)$ to be the Word2Vec embedding of word w_i. Then according to [21], the vector $V(w_3)$ is closest to

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

where “closeness” is in terms of cosine similarity. Results such as this indicate that in the NLP context, Word2Vec embeddings capture meaningful aspects of the semantics of the language.

In this research, we train Word2Vec models on opcode sequences. The resulting embedding vectors are used as feature vectors for several different classifiers. Analogous to the HMM feature vectors discussed in the previous section, these Word2Vec embeddings serve as engineered features that may be more informative than the raw opcode sequences.

2.1.3 BERT

The transformer-based NLP model BERT (Bidirectional Encoder Representations from Transformers) [4] has proven to be particularly effective in complex language-based tasks such as masked word prediction and sentiment classification. At the base of its architecture there is a stack of trained transformer encoders. This approach enables the model to generate contextualized word embedding by taking into consideration the context in which the specific words are used.

The BERT model identifies relations among words using attention, which is a process that helps to retain long-term dependencies in sentences up to a maximum of 512 words. While this length is sufficient for NLP applications, the opcode sequence from an executable file will typically far exceed this value. We found that the first 400 opcodes from each executable is sufficient to achieve good classification results, although we use 512 in all of the BERT experiments reported in this paper. Also, we employ the DistilBERT [6] implementation of BERT. DistilBERT is a computationally efficient and lightweight version of BERT.

2.1.4 ELMo

ELMo (Embeddings from Language Models) was first introduced in [24]. The embeddings generated by ELMo capture the context of a given word in the sentence by based on vectors that are derived from a trained bidirectional Long Short Term Memory (biLSTM) models. Its word representations are, in fact, a function of the internal layers of a biLSTM.

As with BERT, ELMo can generate different embeddings for the same word based on context. This is achieved by relying on both the higher-level and lower-level LSTM states—the higher-level states capture word meaning in relation to the context of the sentence, while lower-level states serve to model aspects of sentence syntax.

2.2 Classification techniques

2.2.1 Random forest

Random forest (RF) is a class of supervised machine learning techniques. A random forest is based on decision trees, which are one of the simplest and most intuitive “learning” techniques available. The primary drawback to a simple decision trees is that it tends to overfit—in effect, the decision tree “memorizes” the training data, rather than learning from the it. A random forest overcomes this limitation by a process known as “bagging,” whereby a collection of decision trees are trained, each using a subset of the available data and features [31].

A variety of hyperparameters are used to optimize the performance of a random forest. These parameters include the number of decision trees, the minimum number of samples required to define a split, the minimum number of samples on a leaf node, the maximum depth of the tree, and so on. For example, setting the maximum depth too large tends to result in overfitting.

2.2.2 *k*-nearest neighbors

Perhaps the simplest learning algorithm possible is k-nearest neighbors (kNN). In this technique, there is no explicit training phase, and in the testing phase, a sample is classified simply based on the nearest neighbors in the training set. This is a lazy learning technique, in the sense that all computation is deferred to the classification phase. The parameter k specifies the number of neighbors used for classification. Small values of k tend to results in highly irregular decision boundaries, which is a hallmark of overfitting.

2.2.3 Support vector machine

Support vector machines (SVM) are popular supervised learning algorithms [3] that have found widespread use in malware analysis [11]. A key concept behind SVMs is a separating hyperplane that maximizes the margin, which is the minimum distance between the classes. In addition, the so-called kernel trick introduces nonlinearity into the process, with remarkably little computational overhead.

There are two important parameters when training a support vector machine. The hyperparameter C is a cost or regularization parameter. This parameter trades off the misclassification of data against the decision boundary, i.e., allowing more misclassifications typically results in a larger margin. For nonlinear kernels, the parameter γ is the inverse of the radius of influence of the training vectors. A large value of γ indicates that two samples must be very close to each other to be considered similar, whereas small values of γ mean that two samples are considered similar even if they are relatively far from each other. In this way, γ influences the shape of the decision boundary.

Several popular nonlinear kernels are used in SVMs. In this research, we experiment with linear kernels and nonlinear radial basis function (RBF) kernels.

2.2.4 Convolutional neural network

Neural networks are a large and diverse class of learning algorithms that are loosely modeled on structures of the brain. A deep neural network (DNN) is a neural network with multiple hidden layers—such networks are state of the art for many learning problems. Convolutional neural networks (CNN) are DNNs that are optimized for image analysis, but have proven effective in many other problem domains. The architecture of a CNN consists of hidden layers, along with input and output layers. The hidden layers of a CNN typically include convolutional layers, pooling layers, and a fully connected output layer [21].

2.3 Selective survey of related work

For about the past 15 years, machine learning has been widely studied in the field of malware research. This section introduces representative examples from the literature that are related to the work considered in this paper.

In [32], the authors consider features based on API calls. Other research considers features such as opcodes, system calls, control flow graphs, and byte sequences for malware detection [11,33].

As discussed in [32], features obtained via static analysis (e.g., opcode sequences) results in more efficient and faster techniques as compared to those that rely on features that require dynamic analysis (e.g., API calls). However, dynamic analysis often provides a more accurate reflection of malware, and obfuscation techniques are generally less effective for dynamic features.

The malware literature has many examples where various feature engineering techniques are used in conjunction with machine learning models. For example, in [28], the author proposes a machine learning technique that uses HMM matrices as the input to a convolutional neural network to classify malware families. Researchers in [28] use SVMs to classify HMMs that have been trained on malware samples. In [13], the authors consider an ensemble model that combines predictions from `asm` and `exe`, files together—the predictions are stacked and fed to a neural network for classification. In [25], the authors use Word2Vec to generate embeddings from machine instructions. Moreover, they propose a proof-of-concept model to train a convolutional neural network based on the Word2Vec embeddings.

As mentioned above, the research in this paper builds on the work in [8], which is itself most closely related to [25], and tangentially related to [28,32]. Prior to [8], only the Word2Vec-CNN model in Table 1 had been considered. In [8], the following models from Table 1 were analyzed: HMM2Vec-kNN, HMM2Vec-SVM, HMM2Vec-RF, Word2Vec-kNN, Word2Vec-SVM, and Word2Vec-RF. Here, we extend this line of research to include BERT and ELMo embeddings. Moreover, we experiment with a wider array of parameters, as compared to previous related work. In the next section, we discuss each of the 16 techniques that we consider in this paper. Recall that these techniques are summarized in Table 1.

3 Implementation

In this section, we first give information about the dataset used in this research. Then we discuss the various machine learning techniques that are the focus of the experiments reported in Sect. 4.

3.1 Dataset

The raw dataset used for our experiments includes 2793 malware families with one or more samples per family [10]. Figure 1 lists all of the families in our dataset that have more than 300 samples.

We selected seven of the families listed in Fig. 1 that have more than 1000 samples, and randomly selected 1000 samples of each type, giving us a total of 7000 samples. The following seven families were selected for this research.

BHO can perform a wide variety of malicious actions, as configured by an attacker [15].

CeeInject is designed to conceal itself from detection, and hence various families use it as a shield to prevent detection. For example, CeeInject can obfuscate a bitcoin mining client, which might have been installed on a system without the user's knowledge or consent [17].

FakeRean pretends to scan the system, notifies the user of nonexistent issues, and asks the user to pay to clean the system [14].

OnLineGames steals login information and captures user keystroke activity [16].

Renos will claim that the system has spyware and ask for a payment to remove the nonexistent spyware [18].

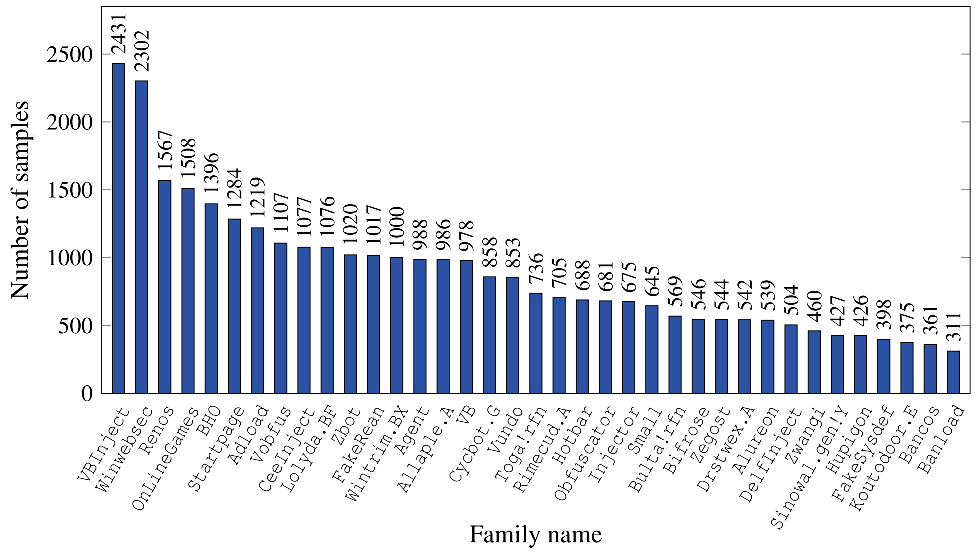


Fig. 1 Families in dataset with at least 300 samples

Vobfus is a family that downloads other malware onto a user's computer and makes changes to the device configuration that cannot be restored by simply removing the downloaded malware [19].

Winwebsec is a trojan that presents itself as antivirus software—it displays misleading messages stating that the device has been infected and attempts to persuade the user to pay a fee to free the system of malware [20].

For each sample, we train an HMM, a Word2Vec, a BERT, and an ELMo model using opcode sequences. The raw dataset consists of `exe` files, and hence we first extract the mnemonic opcode sequence from each malware sample. We use `objdump` to generate `asm` files from which we extract opcode sequences. For each opcode sequence we retain the M most frequent opcodes and remove all others. In previous work [8], the value $M = 31$ was used. To validate this selection, we experimented with $M \in \{20, 31, 40\}$, where "most frequent" is based on the opcode distribution over the entire dataset. For efficiency, the number of hidden states in each HMM was chosen to be $N = 2$, and the number of output symbols is given by M. For the Word2Vec models, we experiment with additional hyperparameters.

Our experiments involving $M \in \{20, 31, 40\}$ are discussed at the start of Sect. 4. Based on the results of such experiments, we selected $M = 31$ for all subsequent experiments. The 31 most frequent opcodes are listed in Fig. 2, along with the percentage of the total that each opcode represents.

From the numbers in Fig. 2, we see that any opcode outside of the top 31 accounts for less that 0.5% of the total opcodes. Since we are considering statistical-based feature engineering techniques, these omitted opcodes are highly unlikely to affect the results to any significant degree.

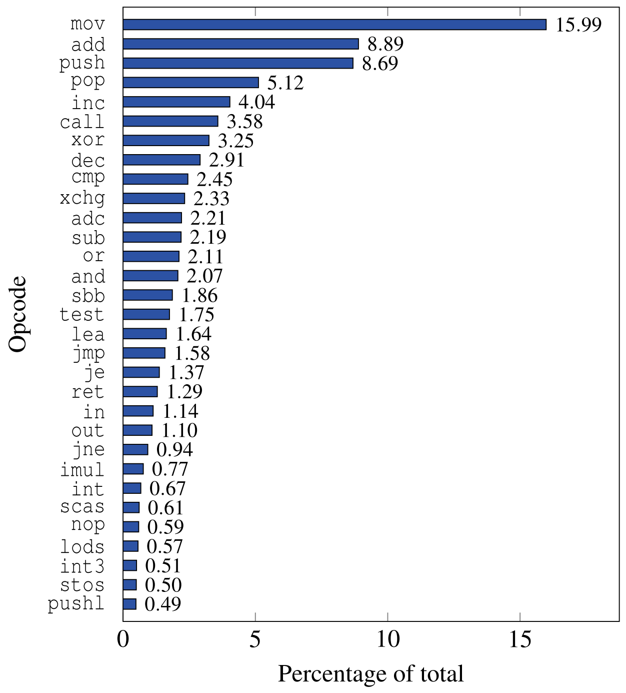


Fig. 2 The 31 most frequent opcodes

3.2 Classification techniques

In this section, we discuss the machine learning models that form the basis for the research in this paper. Specifically, we introduce the models that we refer to as HMM2Vec-SVM, HMM2Vec-RF, HMM2Vec-kNN, and HMM2Vec-CNN. We then briefly discuss the analogous Word2Vec, BERT, and ELMo techniques.

To train our hidden Markov models, we use the `hmmlearn` library [7], and we select the best HMM based on multiple random restarts. For all remaining machine learning techniques, except for CNNs, we used `sklearn` [23]. To train our CNN models, we use the Keras library [9].

3.2.1 HMM-based techniques

For our HMM2Vec-SVM technique, we first train an HMM for each sample, using the extracted opcode sequence as the training data. Then we use an SVM to classify the samples, based on HMM observation probability matrices—the B matrices in the standard HMM terminology [30]. Each converged B matrix is vectorized by simply concatenating the rows. Since $N = 2$ is the number of hidden states and M is the number of distinct opcodes in the observation sequence, each B matrix is $N \times M$. Consequently, the resulting engineered feature vectors are all of length NM. When training the SVM, we experiment with various hyperparameters and kernel functions.

Our HMM2Vec-RF, HMM2Vec-kNN, and HMM2Vec-CNN techniques are analogous to the HMM2Vec-SVM technique. Note that for our HMM2Vec-CNN experiments, we use a one-dimensional CNN. In each case, we tune the relevant parameters.

3.2.2 Word2Vec-based techniques

As mentioned above, Word2Vec is typically trained on a series of words, which are derived from sentences in a natural language. In our research, the sequence of opcodes from a malware executable is treated as a stream of “words.” Analogous to our HMM2Vec experiments, we concatenate the Word2Vec embeddings to obtain a vector of length NM, where M is the number of distinct opcodes in the training set and N is the length of the embedding vectors.

Once we have trained the Word2Vec models to obtain the engineered feature vectors, the classification process for each of Word2Vec-SVM, Word2Vec-RF, Word2Vec-CNN, and Word2Vec-kNN is analogous to that for the corresponding HMM-based technique. As with the HMM classification techniques, we tune the parameters in each case.

3.2.3 BERT-based techniques

The BERT model we use has been pre-trained on English text. After fine-tuning on the malware families used in this research, the resulting word embeddings for each sample are used to train our machine learning models following an analogous process as our previous techniques. In this case, though, we first tokenize the opcodes with the help of a wordpiece tokenizer, and then feed them into the BERT transformer model. We use 512 tokens, as that is the maximum vocabulary that BERT allows. We extract the CLS (aka classifier) token that is used to capture the information about the entire sentence, and used it to train our classifiers. These results were promising, but by further experimentation, we obtained even better accuracy by including the last four layers of the model concatenated to the embeddings. Hence, we use the concatenated last four layers and embedding as our feature vectors in the BERT experiments.

3.2.4 ELMo-based techniques

As with BERT, the ELMo model that we use was pre-trained on English, and the embeddings generated by ELMo are contextual. For ELMo, we used the first 64 tokens from each file to generate the word embedding. This value was chosen because represented an optimal trade-off between the outcome of the classification and the computational overhead. After passing the tokens through the model, we obtain a tensor of size $(64, 256)$ for each file. However, this feature only yielded about 80% of accuracy, hence, we included the tensors for individual token, which gives us a vector of length 256 for each file. We perform all experiments on these concatenated embeddings, with the exception of the CNN classifier which performed better without considering the individual tokens.

4 Experiments and results

In this section, we present the results of our machine learning experiments for malware classification. As discussed above, these experiments are based on opcode sequences, with feature engineering involving HMM2Vec, Word2Vec, BERT, and ELMo models. We consider four classifiers, giving us a total of 16 distinct sets of experiments, which we refer to using the terminology in Table 1. In cases where competitive results are obtained, we provide confusion matrices so that we can further analyze the results.

Before discussing our multiclass experimental results, we first consider binary classification experiments using various numbers of opcodes. The purpose of these experiments is to determine the number of opcodes to use in our subsequent multiclass experiments.

4.1 Binary classification

In this section, we classify samples from the Winwebsec and Fakerean malware families, both of which are examples of rogue security software that claim to be antivirus tools. We compare the accuracies when using the M most frequent opcodes, for $M \in \{20, 31, 40\}$.

For each of these binary classification experiments, we generate a Word2Vec model for each sample in both families, using a vector size of $N = 2$ and window sizes of $W \in \{1, 5, 10, 30, 100\}$. Thus, we conduct 15 distinct experiments, each involving 2000 labeled samples. In each

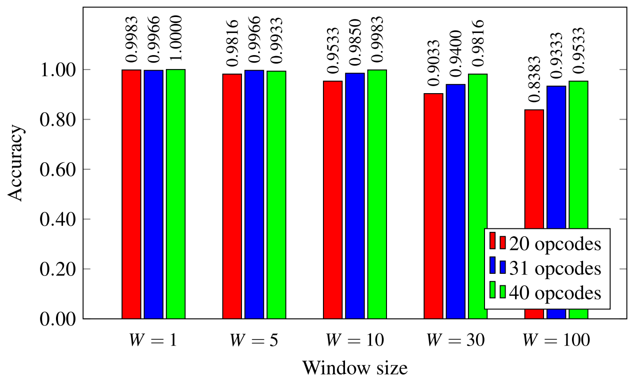


Fig. 3 Binary classification using Word2Vec-SVM (Winwebsec vs Fakerean)

case, we use a 80–20 training-testing split and 5-fold cross-validation. The results of these experiments are summarized in Fig. 3.

From Fig. 3, we see that good results are obtained for window size $W = 5$ and 31 or 40 opcodes. Both of these cases yield an accuracy in excess of 99%. But, the improvement when using 40 opcodes over 31 opcodes is relatively small, and with 40 opcodes, feature extraction and training times are greater. Therefore, in all of the multiclass experiments discussed in the next sections, we use the 31 most frequent opcodes.

4.2 HMM2Vec multiclass experiments

For all of our multiclass experiments, we consider the seven malware families that are discussed in Sect. 3.1, namely, BHO [15], Ceeinject [17], Fakerean [14], OnLineGames [16], Renos [18], Vobfus [19], and Winwebsec [20]. We extracted opcodes from 50 malware families and use the 31 most frequent to train HMMs for each sample in each of the seven families under consideration. For all HMMs, the number of hidden states is selected to be $N = 2$. Since we are considering 31 distinct opcodes, we have $M = 31$, giving us engineered HMM2Vec feature vectors of length 62.

As mentioned above, we train HMMs using the `hmmlearn` library [7] and we select the highest scoring model based on multiple random restarts. The precise number of random restarts is determined by the length of the opcode sequence—for shorter sequences in the range of 1000 to 5000 opcodes, we use 100 restarts; otherwise we select the best model based on 50 random restarts. The B matrix of the highest-scoring model is then converted to a one-dimensional vector.

To obtain the HMM2Vec features, we convert the B matrix of a trained HMM into vector form. A subtle point that arises in this conversion process is that the order of the hidden states in the B matrix need not be consistent across different models. Since we only have $N = 2$ hidden states in our experiments, this means that the order of the rows of the corresponding B matrices may not agree between different models. To account for this possibility, we determine the hidden state that has the highest probability with respect to the `mov` opcode and we deem this to be the first half of the HMM2Vec feature vector, with the other row of the B matrix being the second half of the vector. Since `mov` is by far the most frequent opcode, this will yield a consistent ordering of the hidden states.

4.2.1 HMM2Vec-SVM

Table 2 gives the results of a grid search over various parameters and kernel functions. As with all of our multiclass experiments, we use a 80–20 split of the data for training and testing, with 5-fold cross-validation. For the multiclass SVM, we use a one-versus-other technique. From the results in Table 2, we see that the RBF kernel performs poorly, while the linear kernel yields consistently strong results. Our best results are obtained using a linear kernel with $C = 100$ and $C = 1000$.

Figure 4 gives the confusion matrix for our HMM2Vec-SVM experiment, based on a linear kernel with $C = 100$. We see that BHO and Vobfus are classified with the highest accuracies of 94.2% and 96.6%, respectively. On the other hand, Winwebsec and Fakerean are the most challenging, with 9% and 7% misclassification rates, respectively. We also note that OnLineGames samples are frequently misclassified as Fakerean.

4.2.2 HMM2Vec-*k*NN

Recall that in kNN, the parameter k is the number of neighbors that are used to classify samples. We experimented with

Table 2 HMM2Vec-SVM accuracies

Kernel	Parameters		Accuracy
	C	γ	
Linear	1	N/A	0.83
	10	N/A	0.87
	100	N/A	0.88
	1000	N/A	0.88
RBF	1	0.0010	0.13
	1	0.0001	0.13
	10	0.0010	0.42
	10	0.0001	0.13
	100	0.0010	0.69
	100	0.0001	0.34
	1000	0.0010	0.83
	1000	0.0001	0.70

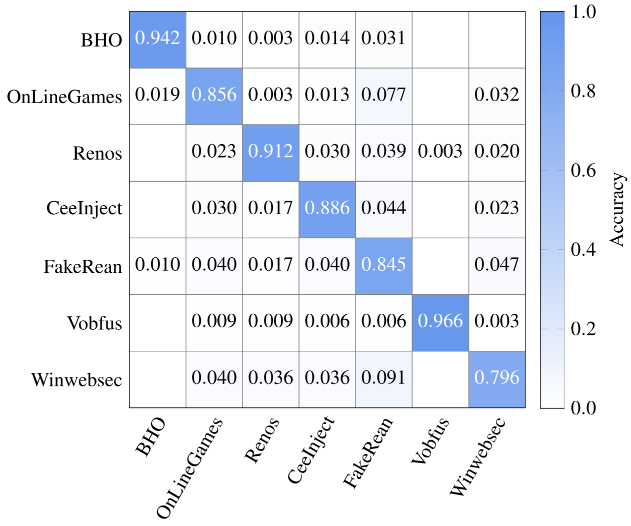


Fig. 4 Confusion matrix for HMM2Vec-SVM with linear kernel

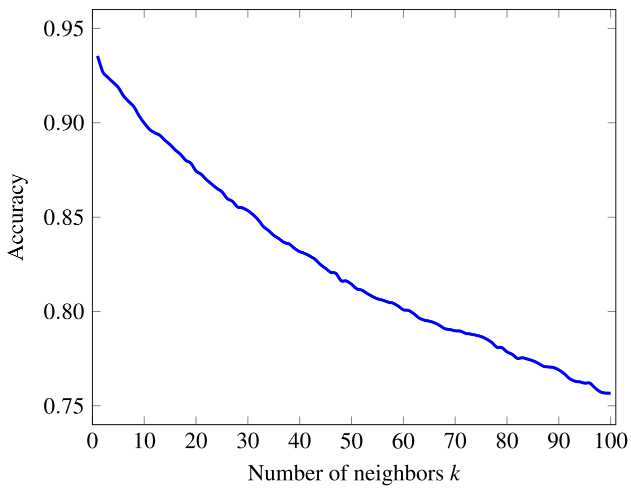


Fig. 5 HMM2Vec-kNN accuracy as a function of k

kNN classifiers using our engineered HMM2Vec features for each $k \in \{1, 2, 3, \ldots, 50\}$. Figure 5 gives the resulting multiclass accuracy for these HMM2Vec-kNN experiments as a function of k.

From Fig. 5, we see that as the accuracy declines as k increases. However, small values of k result in a highly irregular decision boundary, which is a sign of overfitting. As a general rule, we should choose $k \approx \sqrt{S}$, where S is the number of training samples. For our experiment, this gives us $k = 70$, for which we obtain an accuracy of about 79%.

4.2.3 HMM2Vec-RF

There are many hyperparameters to consider when training a random forest. Using our HMM engineered features, we performed a randomized search and obtained the best results with the parameter in Table 3.

Table 3 Randomized search parameters for HMM2Vec-RF

Hyperparameter	Value
`n-estimators`	1000
`min_samples_split`	2
`min_samples_leaf`	1
`max_features`	Auto
`max_depth`	50
`bootstrap`	False

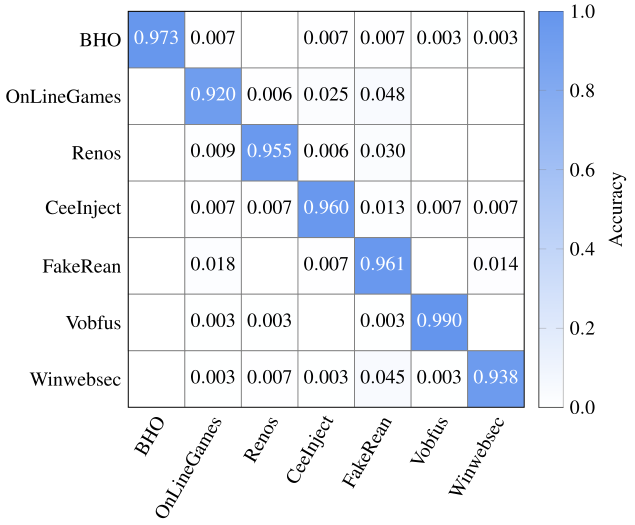


Fig. 6 Confusion matrix for HMM2Vec-RF using grid parameters

Using the hyperparameters in Table 3, our HMM2Vec-RF classifier achieves an overall accuracy of 96%. In Fig. 6, we give the results of this experiment in the form of a confusion matrix. From this confusion matrix, we see that BHO and Vobfus are classified with high accuracies of 97% and 99%, respectively. The misclassifications between OnLineGames and Fakerean are reduced, as compared to the SVM classifier considered above, as are the misclassifications between Winwebsec and Fakerean.

4.2.4 HMM2Vec-CNN

Next, we consider classification based on CNNs. There are numerous possible configurations and many hyperparameters in such models. Due to the fact that our feature vectors are one-dimensional, we use one-dimensional CNNs.

As with all of our experiments, we split the data 80–20 into training and validation sets, with 5-fold cross-validation. We train each model using the rectified linear unit (ReLU) activation function and 200 epochs. To construct these models, we used the Keras library [9].

For our first set of experiments, we train CNNs using one input layer of dimension 200, a hidden layer with 500 neurons, and our output layer has seven neurons, since we have seven classes. We use a mean squared error (MSE) loss function.

Using stochastic gradient descent (SDG) as the optimizer, we obtained an accuracy of about 50%. Switching to the Adam optimizer [34], we achieve a training accuracy of 97% and a testing accuracy of 92%. Consequently, we use Adam for all further experiments.

We train a CNN with two hidden layers, one input layer, and an output layer, with 20, 200, and seven neurons, respectively. In this case, using categorical cross-entropy (CC) as the loss function, we achieved a testing accuracy of 88%, but the model showed a 40% loss.

Next, we expand the hidden layer to 500 neurons and perform a grid search to identify the best hyperparameters. We experimented with various loss functions, we use 200 neurons in the input layer, one hidden layer with 500 neurons, ReLU activation functions, and an output layer with seven neurons, followed by a softmax activation. In this setup, we achieved a testing accuracy of 93.8% using the CC loss function. However, the training accuracy reached 100%, which indicates overfitting. Figure 7(a) and (b) show model accuracy and loss, respectively, for this experiment. As can be seen in Fig. 7(a), the gap between training accuracy and validation accuracy is increasing, while in Fig. 7(b), the gap between validation loss and training loss is also increasing. Since the training accuracy continues to increase for additional epochs, these results indicate that the model likely overfits the training data.

There are several possible ways to mitigate overfitting—we employ regularization based on a dropout layer. Figure 8 illustrates the role of dropouts. Intuitively, when a neuron is dropped at a particular iteration, it forces other neurons to become active, which reduces the tendency of some neurons to dominate during training. By spreading the training over more neurons, we reduce the tendency of the model to overfit the data.

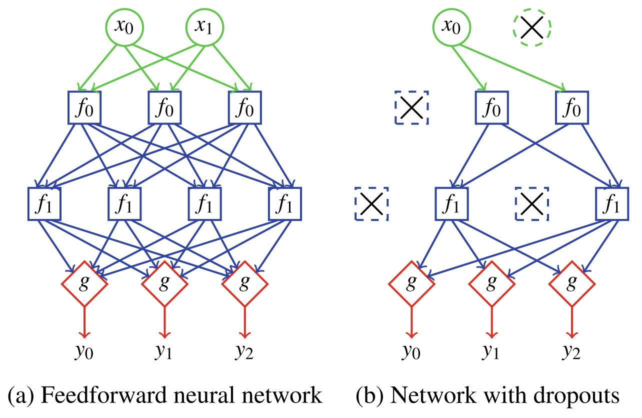


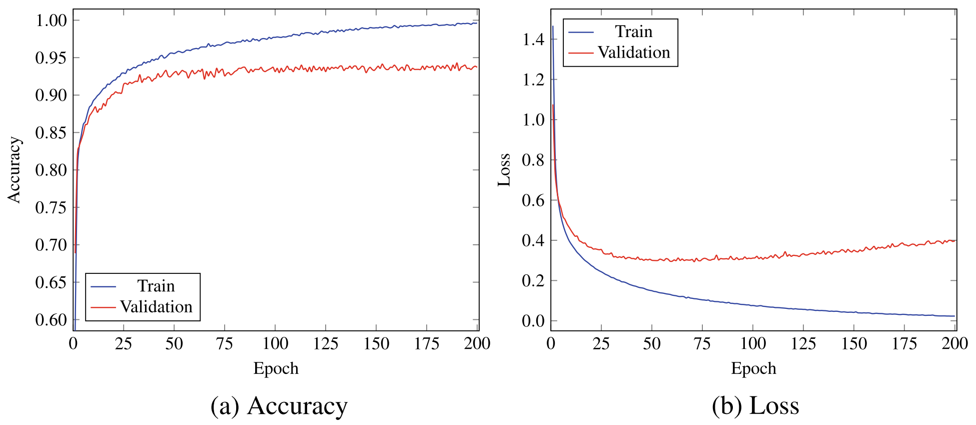
Fig. 8 Neural network with dropouts

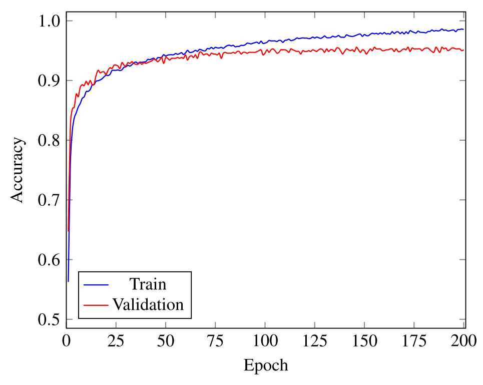
When we set the dropout rate to 0.5, we achieve a testing accuracy of 94.2% with a training accuracy of 98%. In this case, we have eliminating the overfitting that was observed in our previous models. Figure 9(a) and (b) shows model accuracy and and loss for this case. Training and validation accuracies increase as we increase the number of epochs. From Fig. 9(b), we see that the validation accuracy and training accuracy better track each other, which indicates that we have reduced the overfitting.

4.3 Word2Vec multiclass experiments

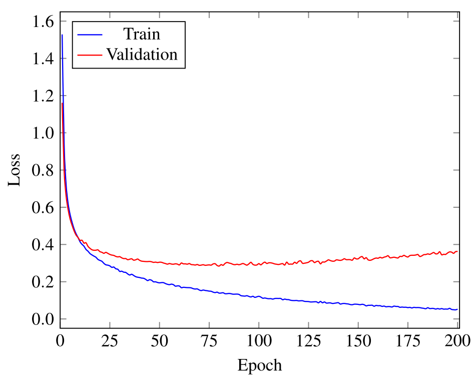
The experiments in this section are analogous to the HMM2Vec experiments in Sect. 4.2. However, Word2Vec includes more parameters that we can easily adjust, as compared to HMM2Vec, and hence we experiment with these parameters. Specifically, for our Word2Vec models, we experiment with different window sizes W and different lengths N of the embedding vectors. Since we are considering feature vectors with 31 distinct opcodes, for the $N = 2$ case, we will have Word2Vec engineered feature vectors of length 62, which is the same size as the HMM2Vec feature vectors considered above. However, for $N > 2$, we have

Fig. 7 HMM2Vec-CNN overfittling





(a) Accuracy



(b) Loss

Fig. 9 Accuracy and loss for HMM2Vec-CNN

larger feature vectors. Also, the window size allows us to consider additional context in Word2Vec models, as compared to our HMM2Vec features.

4.3.1 Word2Vec-SVM

Here, we generate feature vectors using Word2Vec, and apply an SVM classifier. As mentioned above, Word2Vec gives us the flexibility to choose the vector embedding and window sizes, and hence we experiment with these parameters. As in all of the multiclass cases, we consider 1000 malware samples from each of seven families. In all cases, we split the input data 80–20 for training and testing, with 5-fold cross validation. For the SVM experiments, we use a one-versus-other technique.

As with our HMM2Vec-SVM experiments, we first perform a grid search over the parameters for linear and RBF kernels. For these experiments, we use vectors size of $N = 2$ and a window of size $W = 30$. Table 4 summarizes the results of these experiments. We observed that the RBF kernel achieves the highest accuracy.

For our next set of Word2Vec-SVM experiments, we consider a linear kernel. For the Word2Vec features, we use vector lengths $M \in \{2, 31, 100\}$ and windows of size $W \in \{1, 5, 10, 30, 100\}$, giving us a total of 15 distinct Word2Vec-SVM experiments using linear kernels.

The results of these Word2Vec-SVM experiments are summarized in the form of a bar graph in Fig. 10 (a). Note that our best accuracy of 95% for the linear kernel was achieved with input vectors of size $N = 31$ and, perhaps surprisingly, a window of size $W = 1$. These results show that the accuracies significantly improve for embedding vector sizes $N > 2$.

Next, we consider the RBF kernel in more detail. Based on the results in Table 4, we select $C = 1000$ and $\gamma = 0.001$. We generate Word2Vec vectors of sizes $N \in \{2, 31, 100\}$ and window sizes $W \in \{1, 5, 10, 30, 100\}$. The results of these 15 experiments are summarized in Fig. 10(b). In this case, we achieve a best accuracy of 95% with a vector length of $N = 31$ and a window size of either $W = 1$ or $W = 10$. Note that the results improve when the vector size N is increased from 2 to 31, but the accuracy does not improve for $N = 100$.

4.3.2 Word2Vec-kNN

For our Word2Vec-kNN experiments, we again consider the 15 cases given by vector lengths $N \in \{2, 31, 100\}$ and window sizes $W \in \{1, 5, 10, 30, 100\}$. In each case, we consider $k \in \{1, 2, 3, \ldots, 100\}$. We find that for all cases with vectors with sizes $N \in \{2, 31, 100\}$ and window sizes $W \in \{1, 5, 10, 30, 100\}$, we achieve about 94% classification accuracy. In Fig. 11 we give a line graph for Word2Vec-kNN accuracy as a function of k. As in our HMM2Vec-kNN experiments, to avoid overfitting, we choose $k = 70$, which in this case gives us an accuracy of about 89%.

Table 4 Word2Vec-SVM grid search accuracies ($N = 2$ and $W = 30$)

Kernel	Parameters		Accuracy
	C	γ	
Linear	1	N/A	0.86
	10	N/A	0.85
	100	N/A	0.85
	1000	N/A	0.85
RBF	1	0.0010	0.87
	1	0.0001	0.70
	10	0.0010	0.91
	10	0.0001	0.84
	100	0.0010	0.92
	100	0.0001	0.88
	1000	0.0010	0.92
	1000	0.0001	0.90

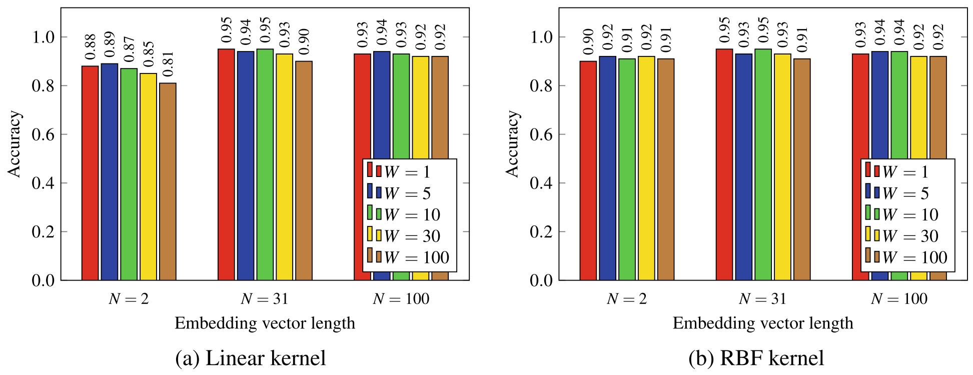


Fig. 10 Word2Vec-SVM experiments

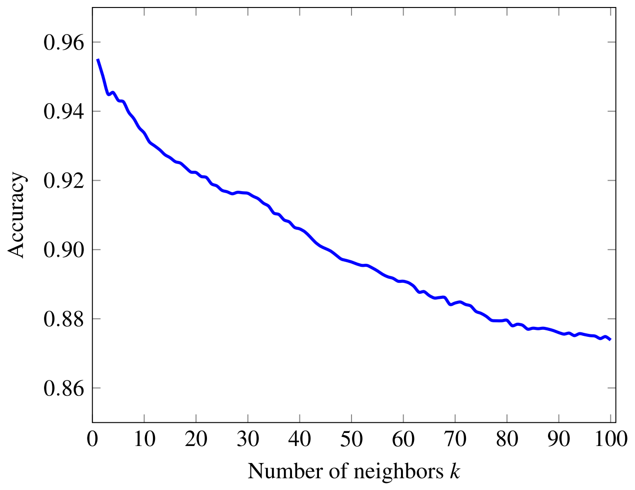


Fig. 11 Word2Vec-kNN accuracy as a function of k ($N = 100$ and $W = 1$)

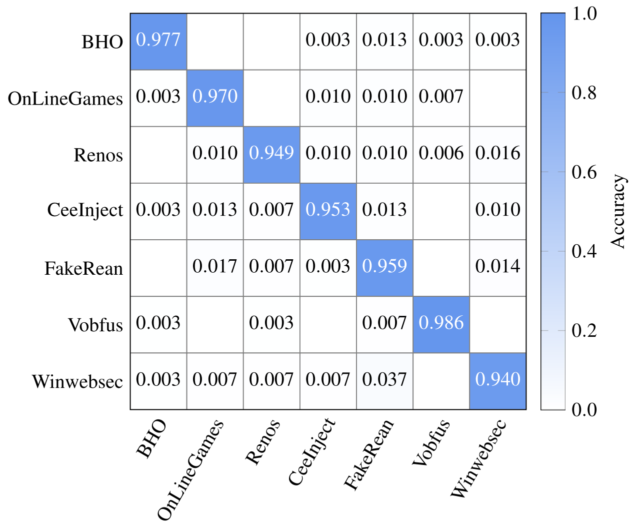


Fig. 12 Confusion matrix for Word2Vec-RF

Table 5 Randomized search parameters for Word2Vec-RF

Hyperparameter	Value
`n-estimators`	1400
`min_samples_split`	2
`min_samples_leaf`	1
`max_features`	Auto
`max_depth`	40
`bootstrap`	False

4.3.3 Word2Vec-RF

In this set of experiments, we consider the same 15 combinations of Word2Vec vector sizes and window sizes as in the previous experiments in this section. In each case, the number of trees in the random forest is set to 1000. We find that the best result for Word2Vec-RF occurs with a vector size of $N = 100$ and a window size of $W = 30$, in which case we achieve an accuracy of 96.2%. The confusion matrix for this case is given in Fig. 12. The worst misclassification is that Winwebsec is misclassified as Fakerean for a mere 3% of the samples tested.

We also conduct experiments on the RF parameters, using a Word2Vec vector size of $N = 100$ and a window size of $W = 30$. Table 5 lists the best parameters obtained based on a grid search. With these parameters, we obtain an accuracy of 93.17%.

4.3.4 Word2Vec-CNN

Using the same parameters as in the previous Word2Vec experiments, that is, vector lengths $N \in \{2, 31, 100\}$ and window sizes $W \in \{1, 5, 10, 30, 100\}$, we consider the same CNN architectures as in the HMM2Vec-CNN experiments, above.

Figure 13(a) and (b) give model accuracy and loss, respectively, for an experiments with $N = 31$ and $W = 1$, where we have trained for 200 epochs. In this case, we achieve 95% testing accuracy, but validation loss increases dramatically, which is a clear indication of overfitting.

To deal with the overfitting that is evident in Fig. 13, we reduce the number of epochs and we tune the learning rate. Specifically, we reduce the number of epochs to 50, we set the learning rate to 0.0001, and we let $\beta_1 = 0.9$ and $\beta_2 = 0.999$, as per the suggestions in [34]. Figure 14 (a) and (b) give model accuracy and loss, respectively, for an experiment based on this selection of parameters. In this case, we achieve 94% testing accuracy, and the loss is reduced significantly. The loss has been reduced, and there is no indication of overfitting in this improved model.

Figure 15 summarized the 15 experiments we conducted using Word2Vec-CNN. For these experiment, as we increase the window size, generally we must decrease the number of epochs to keep the model loss within acceptable bounds.

From Fig. 15, we see that our best accuracy achieved using a Word2Vec-CNN architecture is 94%. Figure 16 gives the confusion matrix for this best Word2Vec-CNN model. We see that the Fakerean family is relatively often misclassified as OnLineGames or Winwebsec. In our previous experiments, we have observed that Fakerean is generally the most challenging family to correctly classify.

4.4 BERT multiclass experiments

Our experiments applying BERT follow the same general structure as the Word2Vec experiments in Sect. 4.3. However, here we also test the information stored in the different model layers. We find that the most important layer for classification is the CLS token, that is, the last hidden state of the first token of the sequence. With further testing, we found that better classification results could be achieved by adding the last four layers of the model, and then concatenating the embeddings. For each of the four classifiers, the best combination of parameters was found through Optuna [22].

4.4.1 BERT-SVM

We use the RBF kernel with parameter $C = 239.8$. In Fig. 17(a) we give the confusion matrix for this case. The families BHO and Vobfus have the highest classification accuracy, which is the same as the HMM2Vec case. However, for BERT-SVM, these families are both classified correctly 100% of the time, and the overall accuracy is 94%.

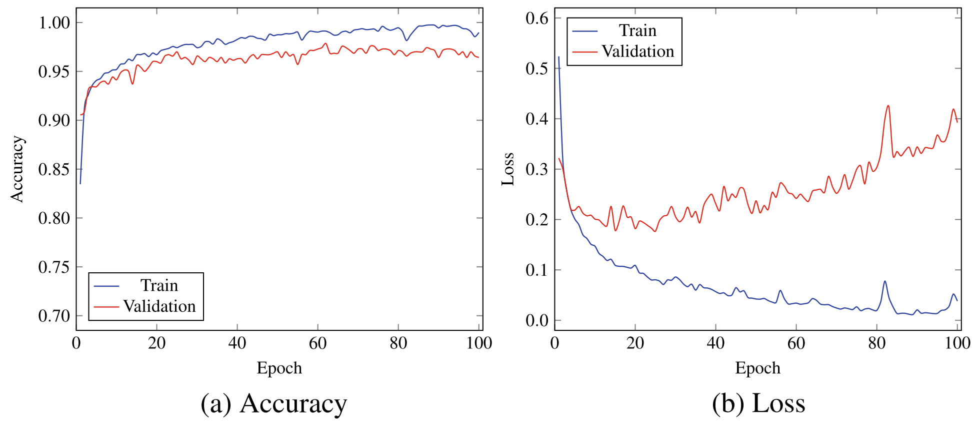


Fig. 13 Word2Vec-CNN overfittling

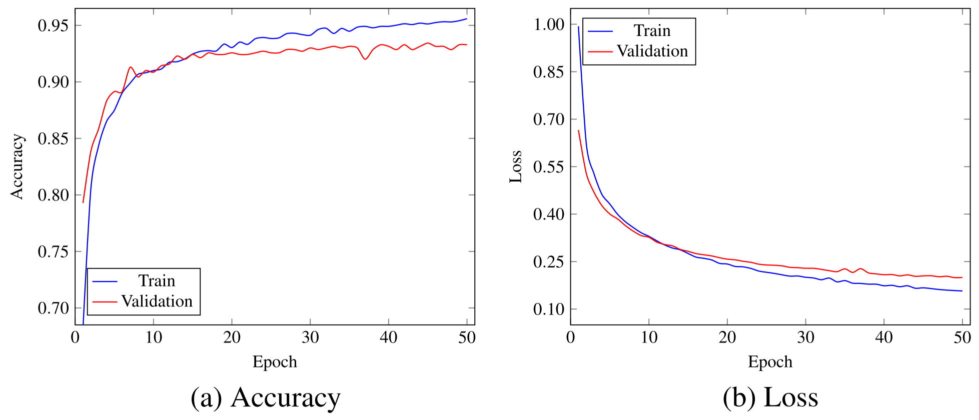


Fig. 14 Model accuracy and loss for Word2Vec-CNN

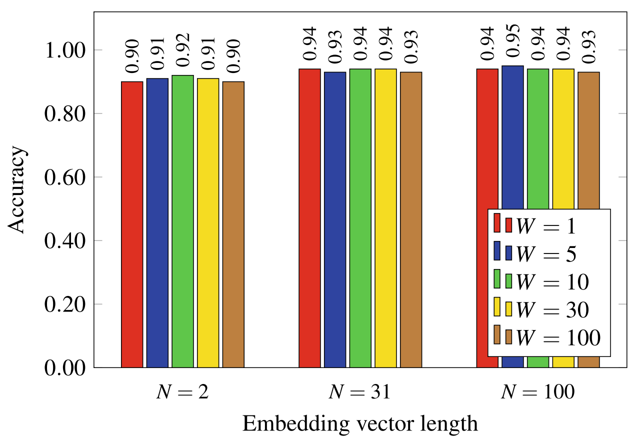


Fig. 15 Accuracies for Word2Vec-CNN experiments

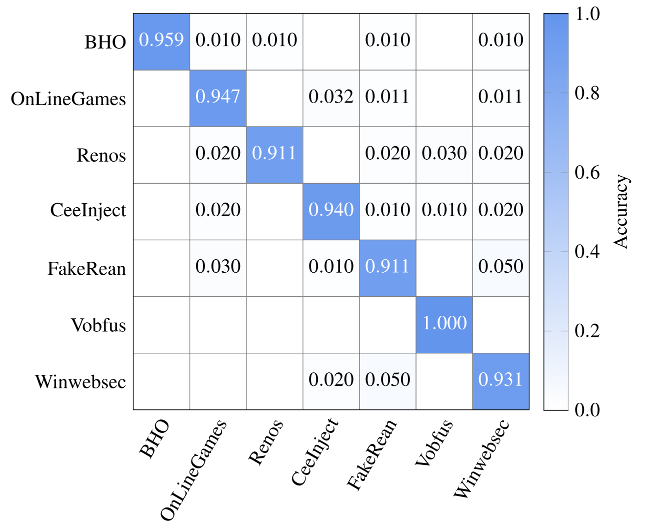
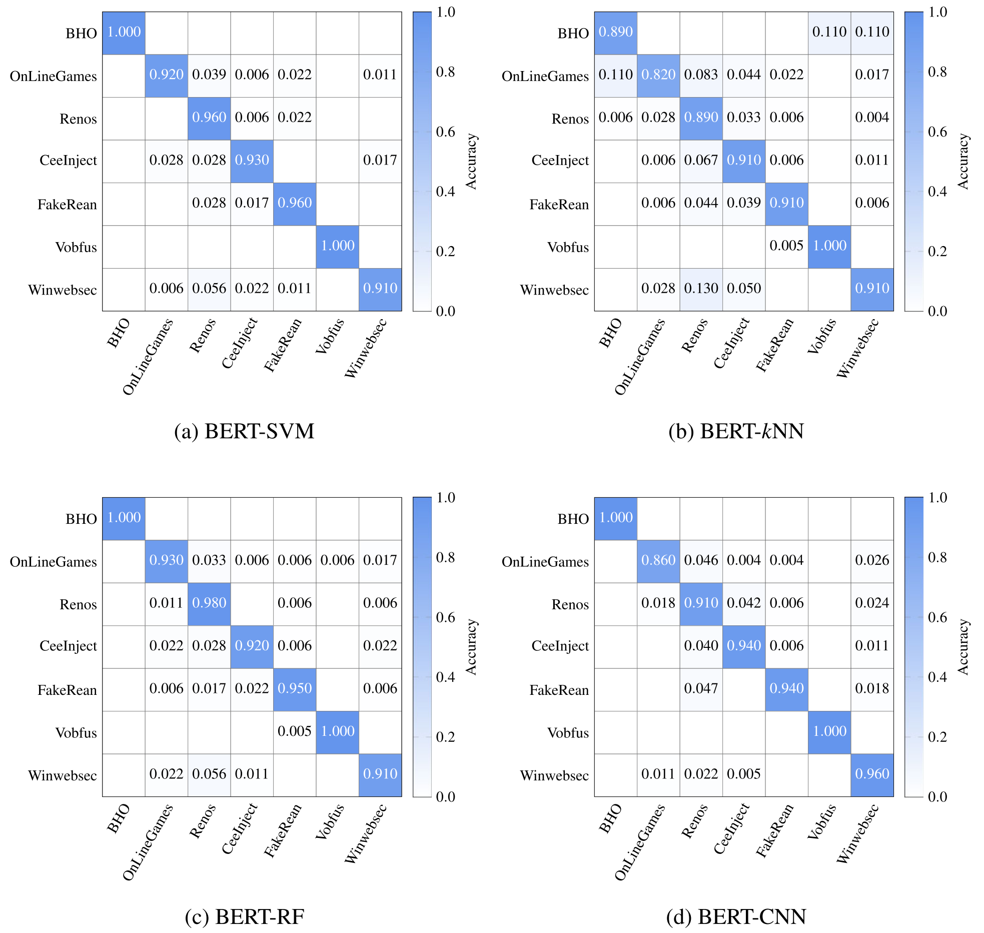


Fig. 16 Confusion matrix for Word2Vec-CNN

4.4.2 BERT-*k*NN

For our BERT-*k*NN experiments, the best results were obtained with $k = 25$. The *k*NN accuracy was 82%, which is inferior to the results obtained in our other BERT experiments. The confusion matrix for this case is given in Fig. 17(b). We see that Winwebsec samples have the worst classification rate, while Vobfus is the easiest family to classify.

Fig. 17 BERT confusion matrices



4.4.3 BERT-RF

For this experiment, the best results were obtained with the numbers of trees set to 716 and the maximum depth equal to 15. We obtained an overall accuracy of 95%. The confusion matrix of this experiment is given in Fig. 17(c).

4.4.4 BERT-CNN

This architecture consists of a ResNet18 backbone and a feed forward network of size (512, 7), where the 7 is due to the number of classes in the experiment. The testing accuracy is the same as the previous experiments with the CNN classifier, namely, 94%, and there are no signs of overfitting. The confusion matrix for this case is given in Fig. 17(d). We see that the CeeInject family has the lowest accuracy. Interestingly, CeeInject is accurately classified in several of our other experiments.

4.5 ELMo multiclass experiments

Again, our ELMo experiments follow the same structure as the Word2Vec experiments in Sect. 4.3. The embedding used in this set of experiments was obtained by using the first 64 tokens from each file. Even though larger values marginally improved the accuracy, they also introduce a considerable overhead in term of computational requirements. All the embeddings were created concatenating the tensors for the individual tokens in the sentence, obtaining feature vectors of length 256 for each sample. For each the four classifiers, the best combination of parameters was found through Optuna [22].

4.5.1 ELMo-SVM

For our SVM results using ELMo word embeddings, we use the RBF kernel with parameter $C = 201.6$. Figure 18(a) is the confusion matrix for this case. The Vobfus family again has the best classification accuracy. In this case, the BHO family did not reach the 100% accuracy observed in the BERT experiment, while the overall accuracy is 94%.

4.5.2 ELMo-*k*NN

The value $k = 45$ was found to result in the best classification, with an overall accuracy of 85%. This is slightly superior to the BERT case. The confusion matrix for ELMo-*k*NN is given in Fig. 18(b). We see that the FakeRean family has a classification accuracy of 72%, which is the worst among all our experiments. Again, Vobfus was classified with the highest accuracy.

4.5.3 ELMo-RF

In this case, we found that setting the number of estimators to 20 and the maximum depth to 26.9 yielded the best results. As in all of our BERT and ELMo experiments, these parameters were selected by Optuna. For ELMo-RF, we attain a classification accuracy of 93%. The confusion matrix of this experiment is given in Fig. 18(c).

4.5.4 ELMo-CNN

The architecture for our ELMo-CNN is the same as for BERT-CNN case, i.e., a ResNet18 backbone and a feed forward network of (512, 7). In this case, the testing accuracy is 95%, with no signs of overfitting. The confusion matrix for this experiment is given in Fig. 18(d). We note that the FakeRean family has the lowest classification accuracy, which is also the case for the ELMo-*k*NN experiments.

5 Conclusion and future work

In this paper, we considered engineered features for malware classification. These engineered features were derived from opcode sequences via four distinct word embedding techniques, namely, HMM2Vec, Word2Vec, BERT, and ELMo. We experimented with a diverse set of seven malware families. We also used four distinct classifiers with each of the engineered feature sets, and we conducted a significant number of experiments to tune the various hyperparameters of the machine learning algorithms.

Figure 19 summarizes the best accuracies for our HMM2Vec, Word2Vec, BERT, and ELMo based classification techniques. From Fig. 19 we see that BERT-SVM and BERT-*k*NN attained the best results, with 96% accuracy, with HMM2Vec-*k*NN, HMM2Vec-RF, BERT-RF, and ELMo-CNN just behind at 95%, and HMM2Vec-CNN, as well as all of the remaining ELMo-based techniques at 94%. Only Word2Vec embeddings were consistently below 90% for all four classifiers. This latter results is somewhat surprising, given that only Word2Vec was considered in work prior to the work reported in [8].

Almost all our machine learning techniques classified samples from BHO, Vobfus, and Renos with relatively high accuracies. We observed that the Winwebsec samples were often misclassified as FakeRean. For example, although overall the accuracies are high, the HMM2Vec techniques tended to misclassify Winwebsec and OnLineGames as FakeRean.

In [8] it was found that a major advantage of Word2Vec was its faster training time in comparison to HMM2Vec. In the experiments presented in this paper, we found the same to be true—generating HMM2Vec features was slower than generating Word2Vec features by a factor of about 15. This

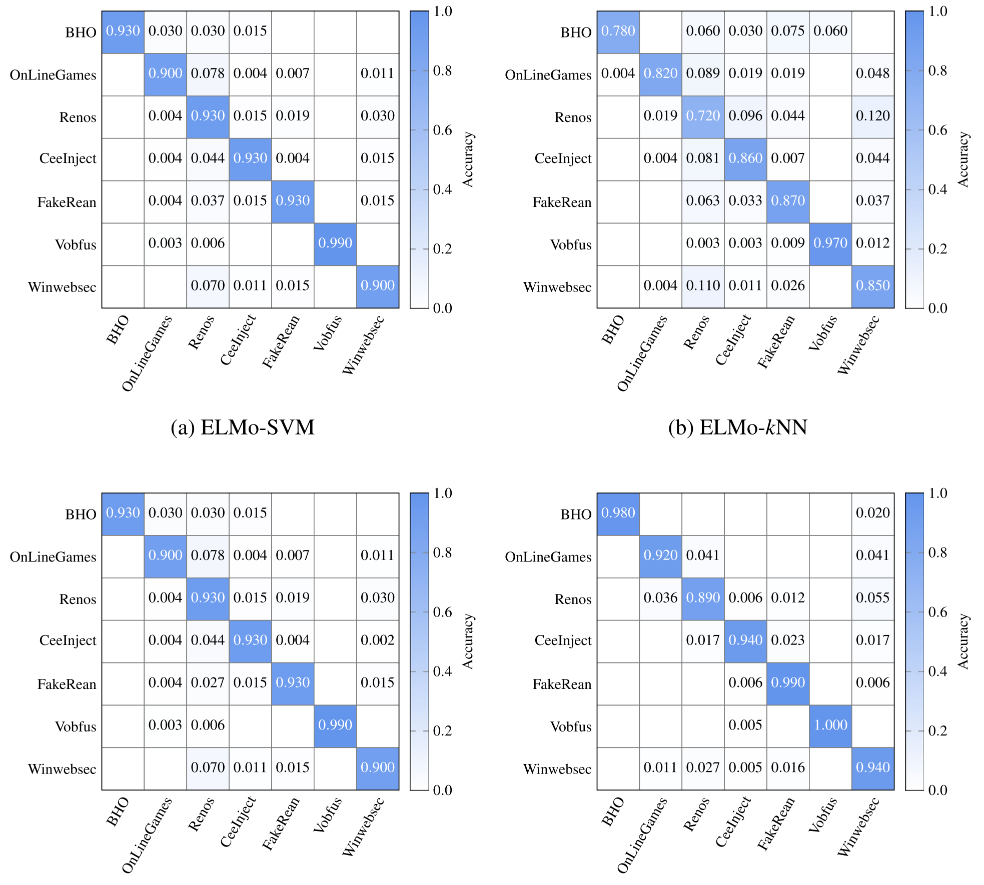


Fig. 18 ELMo confusion matrices

(a) ELMo-SVM

(b) ELMo-*k*NN

(c) ELMo-RF

(d) ELMo-CNN

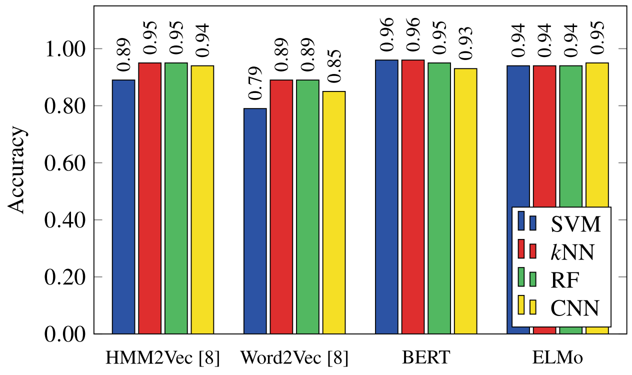


Fig. 19 Comparison of machine learning approaches

vast difference between the two cases was primarily due to the need to train many HMMs, via multiple random restarts, which is particularly important in cases where the amount of training data is relatively small. In contrast, Word2Vec can easily be trained on short opcode sequences, since a larger window size W effectively inflates the number of training samples that are available.

We found that training times for BERT and ELMo are comparable to those for Word2Vec which, as noted above, is 15 times faster than HMM2Vec. Thus, we have shown that we can obtain results that are better than HMM2Vec using vector embedding techniques that are much faster to train. This is a significant improvement over the previous work in [8].

As future extension of this research, similar experiments could be performed on a larger and more diverse set of features. In this research, we only considered opcode sequences—analogous experiments on other features, such as byte n-grams or dynamic features such as API calls would be interesting.

Combinations of features and techniques could likely drive down the error rates. In particular, BERT and ELMo generally make different classification errors, and hence ensembles involving both of these word embedding techniques would have the potential for significantly improved performance.

Further experiments involving the many hyperparameters found in the various machine learning techniques considered here would also be worthwhile. To mention just one of many such examples, additional combinations of window sizes and feature vector lengths could be considered in Word2Vec. Finally, considering word embedding techniques for malware analysis from an adversarial learning perspective would be interesting.

References

1. Aycock, J.: Computer Viruses and Malware. Springer, New York (2006)
2. Beek, C. et al.: McAfee labs threats report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf, August (2019)
3. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
4. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805 (2018)
5. Dhanasekar, D., Di Troia, F., Potika, K., Stamp, M.: Detecting encrypted and polymorphic malware using hidden Markov models. In: Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach, pp. 281–299. Springer (2018)
6. DistilBERT. https://huggingface.co/transformers/model_doc/distilbert.html (2021)
7. Gael, V.: hmmlearn. https://github.com/hmmlearn/hmmlearn (2014)
8. Kale, A.S., Di Troia, F., Stamp, M.: Malware classification with word embedding features. In: Mori, P., Lenzini, G., Furnell, S. (eds.) Proceedings of the 7th International Conference on Information Systems Security and Privacy, ICISSP, pp. 733–742 (2021)
9. Keras. https://github.com/fchollet/keras (2015)
10. Kim, S.: PE header analysis for malware detection. Master's thesis, San Jose State University, Department of Computer Science. https://scholarworks.sjsu.edu/etd_projects/624/ (2018)
11. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**(99), 2721–2744 (2006)
12. Krogh, A., Brown, M., Mian, I.S., Sjölander, K., Haussler, D.: Hidden Markov models in computational biology: applications to protein modeling. J. Mol. Biol. **235**(5), 1501–1531 (1994)
13. Lo, W.W., Yang, X., Wang, Y.: An Xception convolutional neural network for malware classification with transfer learning. In: 10th IFIP International Conference on New Technologies, Mobility and Security, NTMS, pp. 1–5 (2019)
14. Microsoft Security Intelligence. Rogue:Win32/FakeRean. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Rogue:Win32/FakeRean&threatId=124161 (2020)
15. Microsoft Security Intelligence. Trojan:Win32/BHO.BO. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO.BO (2020)
16. Microsoft Security Intelligence. Trojan:Win32/OnLineGames.A. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/OnLineGames.A (2020)
17. Microsoft Security Intelligence. VirTool:Win32/CeeInject. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject (2020)
18. Microsoft Security Intelligence. Win32/Renos. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FRenos (2020)
19. Microsoft Security Intelligence. Win32/Vobfus. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fvobfus (2020)
20. Microsoft Security Intelligence. Win32/Winwebsec. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FWinwebsec (2020)
21. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781 (2013)
22. Optuna. https://optuna.org/ (2021)
23. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
24. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. https://arxiv.org/abs/1802.05365 (2018)
25. Popov, I.: Malware detection using machine learning based on word2vec embeddings of machine code instructions. In: Siberian Symposium on Data Science and Engineering, SSDSE, pp. 1–4 (2017)
26. Rabiner, L.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Proceedings of the IEEE, vol. 77(2), pp. 257–286 (1989)
27. Santos, I., Brezo, F., Nieves, J., Penya, Y.K., Sanz, B., Laorden, C., Bringas, P.G.: Idea: Opcode-sequence-based malware detection. In: International Symposium on Engineering Secure Software and Systems, pp. 35–43 (2010)
28. Sethi, A.: Classification of malware models. Master's thesis, San Jose State University, Department of Computer Science. https://scholarworks.sjsu.edu/etd_projects/703/ (2019)
29. Shaily, S., Mangat, V.: The hidden Markov model and its application to human activity recognition. In: 2nd International Conference on Recent Advances in Engineering Computational Sciences, RAECS, pp. 1–4 (2015)
30. Stamp, M.: A revealing introduction to hidden Markov models. http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf (2004)
31. Stamp, M.: Introduction to Machine Learning with Applications in Information Security. Chapman and Hall, CRC, Boca Raton (2017)
32. Vemparala, S., Di Troia, F., Visaggio, C.A., Austin, T.H, Stamp, M.: Malware detection using dynamic birthmarks. In: Verma, R.M., Rusinowitch, M. (eds.) *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, pp. 41–46 (2016)
33. Wong, W., Stamp, M.: Hunting for metamorphic engines. J. Comput. Virol. **2**(3), 211–229 (2006)
34. Zhang, Z.: Improved Adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service, IWQoS, pp. 1–2 (2018)