1-1-2021

Malware classification with word embedding features

Aparna Sunil Kale
San Jose State University

Fabio Di Troia
San Jose State University, fabio.ditroia@sjsu.edu

Mark Stamp
San Jose State University, mark.stamp@sjsu.edu

Recommended Citation

Aparna Sunil Kale, Fabio Di Troia, and Mark Stamp. "Malware classification with word embedding features" *ICISSP 2021 - Proceedings of the 7th International Conference on Information Systems Security and Privacy* (2021): 733-742. https://doi.org/10.5220/0010377907330742

Malware Classification with Word Embedding Features

Aparna Sunil Kale[a], Fabio Di Troia[b] and Mark Stamp[c]
Department of Computer Science, San Jose State University, San Jose, California, U.S.A.

Keywords: Malware, Machine Learning, Word2Vec, HMM2Vec, CNN.

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. We implement hybrid machine learning techniques, where we engineer feature vectors by training hidden Markov models—a technique that we refer to as HMM2Vec—and Word2Vec embeddings on these opcode sequences. The resulting HMM2Vec and Word2Vec embedding vectors are then used as features for classification algorithms. Specifically, we consider support vector machine (SVM), *k*-nearest neighbor (*k*-NN), random forest (RF), and convolutional neural network (CNN) classifiers. We conduct substantial experiments over a variety of malware families. Our experiments extend well beyond any previous related work in this field.

1 INTRODUCTION

Malware is a software that is created with the intent to cause harm to computer data or otherwise adversely affect computer systems (Aycock, 2006). Detecting malware can be a challenging task, as there exist a wide variety of advanced malware that employ various anti-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 (Dhanasekar et al., 2018).

In this research, we focus on hybrid techniques, in the sense that we perform sophisticated feature engineering based on hidden Markov models and Word2Vec embeddings. In both of these cases, we consider a variety of classifiers. For each of the resulting hybrid techniques, extensive malware classification experiments are conducted over a set of seven challenging malware families. Again, our experiments are based on engineered features derived from opcode sequences.

[a] https://orcid.org/0000-0002-2666-9103
[b] https://orcid.org/0000-0003-2355-7146
[c] https://orcid.org/0000-0002-3803-8368

The remainder of this paper is organized as follows. In Section 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 the novel hybrid machine learning techniques that are 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, Section 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 Machine Learning Techniques

A wide variety of machine learning techniques are considered in this research. We train hidden Markov models and generate Word2Vec embeddings, which are subsequently used as features in various classification algorithms. The classification algorithms considered are random forest, *k*-nearest neighbor, support vector machines, and convolutional neural networks.

Kale, A., Di Troia, F. and Stamp, M.
Malware Classification with Word Embedding Features.
DOI: 10.5220/0010377907330742
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 733-742
ISBN: 978-989-758-491-6

Due to space limitations, each of these techniques is introduced only very briefly in this section.

2.1.1 Hidden Markov Models

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 (Rabiner, 1989), human activity detection (Shaily and Mangat, 2015), and protein sequencing (Krogh et al., 1994). HMMs have also proven useful for malware analysis.

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 Section 4.2.

2.1.2 Word2Vec

Word2Vec has recently gained considerable popularity in natural language processing (NLP) (Mikolov et al., 2013b). 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; see (Mikolov et al., 2013a) for further information.

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 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 the process of "bagging," whereby a collection of decision trees are trained, each using a subset of the available data and features (Stamp, 2017).

2.1.4 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.

2.1.5 Support Vector Machine

Support vector machines (SVM) are popular supervised learning algorithms (Cortes and Vapnik, 1995) that have found widespread use in malware analysis (Kolter and Maloof, 2006). 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 minimal computational overhead.

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

2.1.6 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 (Mikolov et al., 2013b).

2.2 Selective Survey of Related Work

Machine learning has been widely used in malware research. This section introduces representative examples from the literature that are directly related to the work considered in this paper.

The literature is replete with hybrid machine learning techniques for malware classification. For example, in (Sethi, 2019), the author proposes a hybrid machine learning technique that uses HMM matrices as the input to a convolutional neural network to classify malware families. Researchers in (Sethi, 2019) use SVMs to classify trained HMMs. In (Lo et al., 2019), 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 (Popov, 2017), 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.

The research in this paper builds on the work in (Popov, 2017; Sethi, 2019; Vemparala et al., 2016). We propose hybrid machine learning techniques for malware classification using HMM2Vec and Word2Vec engineered features which are derived from opcode sequences. Four different classifiers are considered, giving us a total of eight distinct experiments that we refer to as HMM2Vec-*k*NN, HMM2Vec-SVM, HMM2Vec-RF, HMM2Vec-CNN, Word2Vec-*k*NN, Word2Vec-SVM, Word2Vec-RF, and Word2Vec-CNN. As far as the authors are aware, only one of these eight combinations, namely, Word2Vec-CNN, has been considered in previous work. Moreover, we experiment with a much wider array of window sizes and vector lengths for our Word2Vec models as compared to prior related work. In the next section, we discuss our eight proposed techniques in detail.

3 IMPLEMENTATION

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

3.1 Dataset

The raw dataset used for our experiments includes 2793 malware families with one or more samples per family (Kim, 2018). We selected seven of the families from this dataset that have more than 1000 samples, and randomly selected 1000 samples of each type, giving us a total of 7000 samples. Specifically, the following seven families were selected for this research.

BHO: can perform a wide variety of malicious actions, as specified by an attacker (Microsoft Security Intelligence, 2020b).

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 (Microsoft Security Intelligence, 2020d).

FakeRean: pretends to scan the system, notifies the user of nonexistent issues, and asks the user to pay to clean the system (Microsoft Security Intelligence, 2020a).

OnLineGames: steals login information and captures user keystroke activity (Microsoft Security Intelligence, 2020c).

Renos: will claim that the system has spyware and ask for a payment to remove the nonexistent spyware (Microsoft Security Intelligence, 2020e).

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 (Microsoft Security Intelligence, 2020f).

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 (Microsoft Security Intelligence, 2020g).

For each sample, we train an HMM and a Word2Vec 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. We experiment with the M most frequent opcodes for $M \in \{20, 31, 40\}$, where "most frequent" is based on the opcode distribution over the entire dataset. 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 parameters.

Experiments involving $M \in \{20, 31, 40\}$ are discussed at the start of Section 4. Based on the results of such experiments, we selected $M = 31$ for all subsequent HMM2Vec and Word2Vec experiments. We note that opcodes outside of the top 31 accounts for less that 0.5% of the total. Since we are considering statistical-based feature engineering techniques, these omitted opcodes are highly unlikely to affect the results to any significant degree.

3.2 Hybrid Classification Techniques

In this section, we discuss the hybrid machine learning models that are the basis for the research in this paper. Specifically, we consider HMM2Vec-SVM, HMM2Vec-RF, HMM2Vec-*k*NN, and HMM2Vec-CNN. We then briefly discuss the analogous Word2Vec techniques, namely, Word2Vec-SVM, Word2Vec-RF, Word2Vec-*k*NN, and Word2Vec-CNN.

To train our hidden Markov models, we use the `hmmlearn` library (Gael, 2014), and we select the best HMM based on multiple random restarts. For all remaining machine learning techniques, except for CNNs, we used `sklearn` (Pedregosa et al., 2011). To train our CNN models, we use the Keras library (Chollet, 2015).

3.2.1 HMM Hybrid Techniques

For our HMM2Vec-SVM hybrid 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 the B matrices of the converged HMMs. Each converged B matrix is vectorized by simple 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-*k*NN, and HMM2Vec-CNN techniques are analogous to the HMM2Vec-SVM hybrid technique. For the HMM2Vec-CNN, we use a one-dimensional CNN. In each case, we tune the relevant parameters.

3.2.2 Word2Vec Hybrid 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-*k*NN is analogous to that for the corresponding HMM-based technique. As with the HMM classification techniques, we tune the parameters in each case.

4 EXPERIMENTS AND RESULTS

In this section, we present the results of several hybrid machine learning experiments for malware classification. As discussed above, these experiments are based on opcode sequences, with feature engineering involving HMM and Word2Vec models. We consider four classifiers, giving us a total of eight different experiments, which we denote as HMM2Vec-SVM, HMM2Vec-RF, HMM2Vec-*k*NN, HMM2Vec-CNN, Word2Vec-SVM, Word2Vec-RF, Word2Vec-*k*NN, and Word2Vec-CNN.

Before discussing our hybrid multiclass results, we first consider binary classification experiments using different 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 involv-

ing 2000 labeled samples. In each case, we use a 70-30 training-testing split. The results of these experiments are summarized in Figure 1.

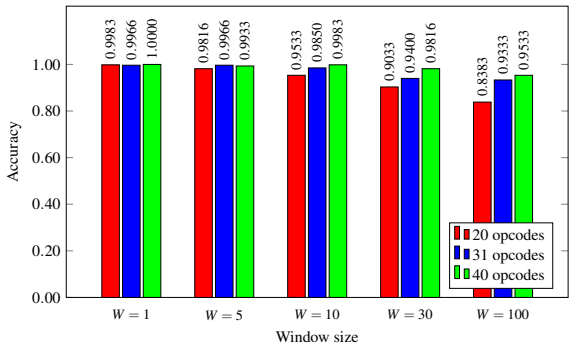


Figure 1: Binary classification using Word2Vec-SVM model (Winwebsec vs Fakerean).

From Figure 1, 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 31 opcodes.

4.2 HMM2Vec Multiclass Experiments

For all of our multiclass experiments, we consider the seven malware families that are discussed in Section 3.1, namely, BHO (Microsoft Security Intelligence, 2020b), Ceeinject (Microsoft Security Intelligence, 2020d), Fakerean (Microsoft Security Intelligence, 2020a), OnLineGames (Microsoft Security Intelligence, 2020c), Renos (Microsoft Security Intelligence, 2020e), Vobfus (Microsoft Security Intelligence, 2020f), and Winwebsec (Microsoft Security Intelligence, 2020g). 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 (Gael, 2014) 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 1 gives the results of a grid search over various parameters and popular SVM kernel functions. As with all of our multiclass experiments, we use a 70-30 split of the data for training and testing. For the multiclass SVM, we use a one-versus-other technique. From the results in Table 1, 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 $C = 1000$.

Table 1: HMM2Vec-SVM accuracies.

Kernel	Parameters		Accuracy
	C	γ	
linear	1	N/A	0.83
linear	10	N/A	0.87
linear	100	N/A	0.88
linear	1000	N/A	0.88
RBF	1	0.001	0.13
RBF	1	0.0001	0.13
RBF	10	0.001	0.42
RBF	10	0.0001	0.13
RBF	100	0.001	0.69
RBF	100	0.0001	0.34
RBF	1000	0.001	0.83
RBF	1000	0.0001	0.70

Figure 2 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.

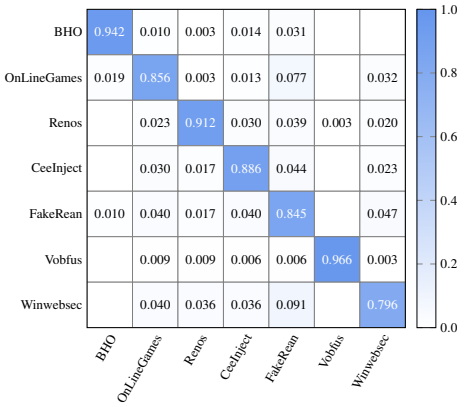


Figure 2: Confusion matrix for HMM2Vec-SVM with linear kernel.

4.2.2 HMM2Vec-*k*NN

Recall that in *k*NN, the parameter k is the number of neighbors that are used to classify samples. We experimented with *k*NN classifiers using our engineered HMM2Vec features for each $k \in \{1,2,3,\ldots,50\}$. We find 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 2.

Table 2: 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

Using the hyperparameters in Table 2, our HMM2Vec-RF classifier achieves an overall accuracy of 96%. In Figure 3, 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.

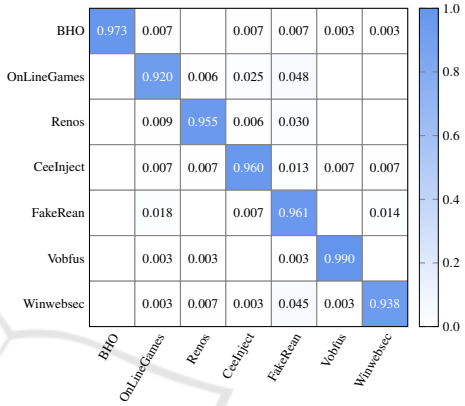


Figure 3: Confusion matrix for HMM2Vec-RF using grid parameters.

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.

We split the data into 80% training, 10% validation, and 10% testing. With this split, we have 5600 training samples, 700 validation samples, and 700 testing samples. We train each model using the rectified linear unit (ReLU) activation function and 200 epochs. To construct these models, we used the Keras library (Chollet, 2015).

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 (Zhang, 2018), 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.

There are several possible ways to mitigate overfitting—we employ regularization based on a dropout layer. 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.

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.

4.3 Word2Vec Multiclass Experiments

The experiments in this section are analogous to the HMM2Vec experiments in Section 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 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 70-30 for training and testing. 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 3 summarizes the results of these experiments. We observed that the RBF kernel achieves the highest accuracy.

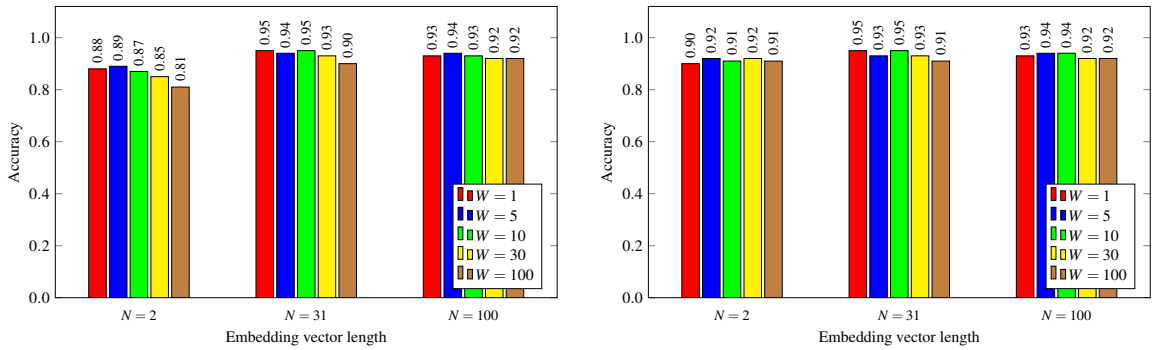
Table 3: Word2Vec-SVM grid search accuracies ($N = 2$ and $W = 30$).

Kernel	Parameters		Accuracy
	C	γ	
linear	1	N/A	0.86
linear	10	N/A	0.85
linear	100	N/A	0.85
linear	1000	N/A	0.85
RBF	1	0.001	0.87
RBF	1	0.0001	0.70
RBF	10	0.001	0.91
RBF	10	0.0001	0.84
RBF	100	0.001	0.92
RBF	100	0.0001	0.88
RBF	1000	0.001	0.92
RBF	1000	0.0001	0.90

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 Figure 4 (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 3, we select $C = 1000$ and $\gamma = 0.001$. We generate Word2Vec vectors of sizes $N \in \{2, 31, 100\}$ and we also consider window sizes $W \in \{1, 5, 10, 30, 100\}$. The results of these 15 experiments are summarized in Figure 4 (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



(a) Linear kernel

(b) RBF kernel

Figure 4: Word2Vec-SVM experiments.

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. As in our HMM2Vec-kNN experiments, to avoid overfitting, we choose $k = 70$, which in this case gives us an accuracy of about 89%.

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 Figure 5. 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 4 lists the best parameters obtained based on a grid search. With these parameters, we obtain an accuracy of 93.17%.

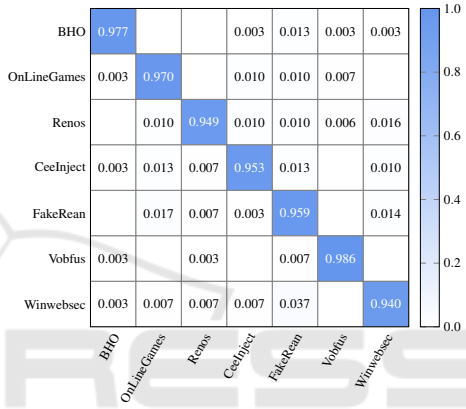


Figure 5: Confusion matrix for Word2Vec-RF.

Table 4: 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

4.3.4 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.

To deal with the overfitting that was evident in our initial experiments, 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 (Zhang, 2018). 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 6 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.

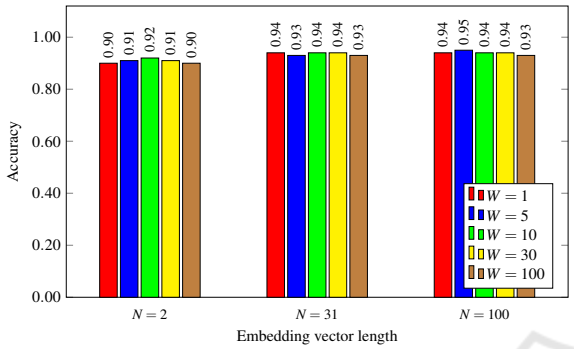


Figure 6: Accuracies for Word2Vec-CNN experiments.

From Figure 6, we see that our best accuracy achieved using a Word2Vec-CNN architecture is 94%. Figure 7 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.

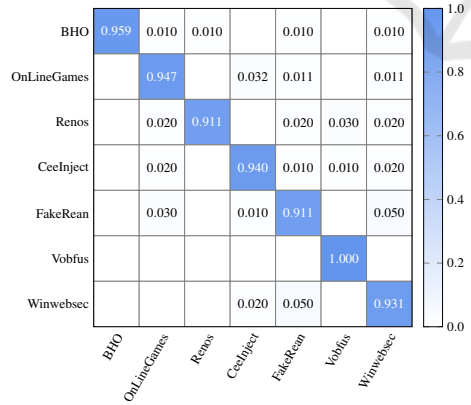


Figure 7: Confusion matrix for Word2Vec-CNN.

5 CONCLUSION AND FUTURE WORK

In this paper, we considered engineered features for malware classification. These engineered features were derived from opcode sequences via a technique we refer to as HMM2Vec, and a parallel set of experiments was conducted using Word2Vec embeddings. We experimented with a diverse set of seven malware families. We also used four different classifiers with each of the two engineered feature sets, and we conducted a significant number of experiments to tune the various hyperparameters of the machine learning algorithms.

Figure 8 summarizes the best accuracies for our Word2Vec and HMM2Vec hybrid classification techniques. From Figure 8 we see that that HMM2Vec-RF and Word2Vec-RF attained the best results, with 96% accuracy when classifying a balanced set of samples from seven families. All of the hybrid machine learning techniques based on Word2Vec embeddings performed well, while the HMM2Vec results were more mixed. This may be due to the relatively limited number of options considered when training HMMs in our experiments, as compared to Word2Vec.

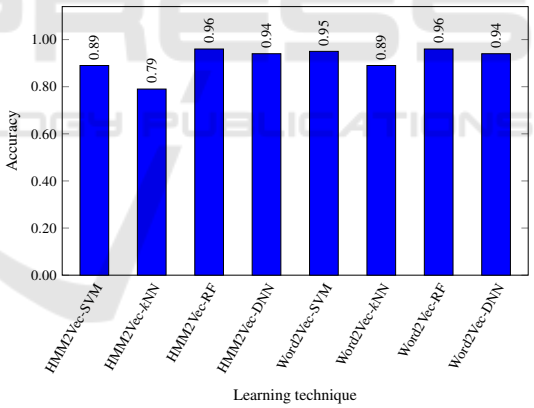


Figure 8: Best accuracies for HMM2Vec hybrid machine learning.

Almost all our hybrid machine learning techniques classified samples from BHO, Vobfus, and Renos with very high accuracy. We observed that the Winwebsec and OnLineGames samples were often misclassified as Fakerean. The percentage of these misclassification was higher in HMM2Vec than Word2Vec, and accounts for most of the difference between these classification techniques.

A major advantage of Word2Vec was its faster training time. We found that generating HMM2Vec features was slower than training comparable Word2Vec models by a factor of about 15. This

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

As future extension of this research, similar experiments could be performed on a larger and more diverse set of malware families. Also, here 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. In addition, other word embedding techniques could be considered, such as those based on principal component analysis (PCA), as considered, for example, in (Chandak et al., 2021).

Further experiments involving the many parameters found in the various machine learning techniques considered here would 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, other machine learning paradigms would be worth considering in the context of malware detection based on vector embedding features. Examples of other machine learning approaches that could be advantageous for this problem include adversarial networks and reinforcement learning.

REFERENCES

Aycock, J. (2006). *Computer Viruses and Malware*. Springer, New York.

Chandak, A., Lee, W., and Stamp, M. (2021). A comparison of Word2Vec, HMM2Vec, and PCA2Vec for malware classification. In *Malware Analysis using Artificial Intelligence and Deep Learning*. Springer.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Dhanasekar, D., Di Troia, F., Potika, K., and Stamp, M. (2018). Detecting encrypted and polymorphic malware using hidden Markov models. In *Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach*, pages 281–299. Springer.

Gael, V. (2014). hmmlearn. https://github.com/hmmlearn/hmmlearn.

Kim, S. (2018). PE header analysis for malware detection. https://scholarworks.sjsu.edu/etd_projects/624/.

Kolter, J. Z. and Maloof, M. A. (2006). Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2721–2744.

Krogh, A., Brown, M., Mian, I., Sjölander, K., and Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531.

Lo, W. W., Yang, X., and Wang, Y. (2019). An xception convolutional neural network for malware classification with transfer learning. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5.

Microsoft Security Intelligence (2020a). Rogue:Win32/FakeRean.

Microsoft Security Intelligence (2020b). Trojan:Win32/BHO.BO.

Microsoft Security Intelligence (2020c). Trojan:Win32/OnLineGames.A.

Microsoft Security Intelligence (2020d). VirTool:Win32/CeeInject.

Microsoft Security Intelligence (2020e). Win32/Renos threat description.

Microsoft Security Intelligence (2020f). Win32/Vobfus.

Microsoft Security Intelligence (2020g). Win32/Winwebsec threat description.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781.

Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013b). Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Popov, I. (2017). Malware detection using machine learning based on word2vec embeddings of machine code instructions. In *2017 Siberian Symposium on Data Science and Engineering (SSDSE)*, pages 1–4.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Sethi, A. (2019). Classification of malware models.

Shaily, S. and Mangat, V. (2015). The hidden Markov model and its application to human activity recognition. In *2015 2nd International Conference on Recent Advances in Engineering Computational Sciences (RAECS)*, pages 1–4.

Stamp, M. (2017). *Introduction to Machine Learning with Applications in Information Security*. Chapman and Hall CRC, 1st edition.

Vemparala, S., Troia, F. D., Visaggio, C. A., Austin, T. H., and Stamp, M. (2016). Malware detection using dynamic birthmarks. In Verma, R. M. and Rusinowitch, M., editors, *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, pages 41–46. ACM.

Zhang, Z. (2018). Improved Adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2.