Journal of Computer Virology and Hacking Techniques (2019) 15:195–208
https://doi.org/10.1007/s11416-019-00331-0

ORIGINAL PAPER

Nonnegative matrix factorization and metamorphic malware detection

Yeong Tyng Ling[1] · Nor Fazlida Mohd Sani[1] · Mohd Taufik Abdullah[1] · Nor Asilah Wati Abdul Hamid[1]

Received: 5 August 2018 / Accepted: 20 March 2019 / Published online: 26 April 2019

Abstract
Metamorphic malware change their internal code structure by adopting code obfuscation technique while maintaining their malicious functionality during each infection. This causes change of their signature pattern across each infection and makes signature based detection particularly difficult. In this paper, through static analysis, we use similarity score from matrix factorization technique called Nonnegative Matrix Factorization for detecting challenging metamorphic malware. We apply this technique using structural compression ratio and entropy features and compare our results with previous eigenvector-based techniques. Experimental results from three malware datasets show this is a promising technique as the accuracy detection is more than 95%.

Keywords Metamorphic malware · Nonnegative matrix factorization · Dimension reduction · Structural analysis

1 Introduction

Malware are threatening today's computing world by its rapid evolving. According to latest report in [1], malware implant to legitimate software were up to 200% in year 2017. Therefore, this leads to urge for new detection technique.

It is generally believed that the detection of metamorphic malware remains a difficult task in the area of research [2,35]. Metamorphic malware utilize code obfuscation techniques to disguise themselves from traditional pattern matching signature based detection. During each propagation, they change their internal syntax or structure while maintaining underlying malicious functionality. A recent study has shown that a well-designed metamorphic malware was capable of evading statistical analysis using Hidden Markov Model (HMM) [29] detection. Due to this reason, several file structural analysis approaches such as in [10,14,19,33], have shown to be very effective at classifying this type of metamorphic malware.

Motivated by this, we adopt structural analysis on two types of file feature, namely, the compression ratio and entropy from previous works in this paper. We also modified the implementation of [14] through inspiration of facial recognition technique based on matrix factorization known as Nonnegative Matrix Factorization (NMF) [17]. NMF learns a parts-based representation of facial images by representing faces with a set of basis images correspond to parts of faces. This recognition technique bears a resemblance to malware detection where detecting a set of malware can actually be represented by detecting a smaller subset of their own. Hence, our contribution is to adapted NMF technique using structural analysis of file compression ratio and entropy features on three types of challenging metamorphic malware families, in particularly, on real malware sample. Our approach use the similarity scores derived from NMF technique to find the similarity of structural pattern that exist in both malware and benign files. As expected, we show that NMF technique yields comparable result when test on small scale datasets with both features. We observed that using structural entropy can produce better detection than using structural compression ratio feature.

The rest of this paper is organized as follows. In Sect. 2 we briefly reviewed metamorphic techniques and related work in detection, the difference between SVD and NMF techniques. Then, in Sect. 3 the application of NMF on our malware detection will be described in details. Section 4 presents the experimental and result analysis. Finally, we conclude and suggest future work in Sect. 5.

✉ Yeong Tyng Ling
ling.yt@student.upm.edu.my

Nor Fazlida Mohd Sani
fazlida@upm.edu.my

[1] Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia

2 Background and related work

In this section, background of metamorphic malware techniques, some recent works on malware detection will be described. Then, mechanism of SVD and NMF techniques will be presented.

2.1 Metamorphic techniques

Metamorphic malware use obfuscation techniques to hide themselves by changing their structure on each infected program while keeping the same functionality. In lieu of encryption, they replace their malicious code sequences with semantically equivalent code during infection and the mutations tend to exhibit statistical properties indistinguishable from other nonencrypted, benign files [23]. According to [7], the obfuscation techniques can be divided into two types, *data flow obfuscation* (junk or dead code insertion, variable or register substitution, instruction replacement or permutation) and *control flow obfuscation* (code transposition).

Data flow obfuscation using junk or dead code insertion is to make the malware codes look different and thus possibly evade hexadecimal string matching. It does not change a program's functionality but to confuse and exhaust the emulator during code analysis [32]. Research had shown that by inserting a quite amount of dead code from benign files can cause the statistical properties of the resulting morphed code indistinguishable from benign codes [29].

Variable or register substitution is another simple technique that a virus variable or instruction operands are stored in different registers for each new infection. It replaces the use of a register in an instruction with another unused register. Even though this has no impact on a program behavior, it does evade signature-based detection.

Instruction replacement or permutation technique is where malware evolve its original code by replacing some instructions with other equivalent ones.

Control flow obfuscation such as code transposition is where it reorder its original virus code sequence by using 'jump' instruction or complex transfer of control as backbone for obfuscation. This causes confusion during malware analysis.

2.2 Metamorphic malware detection

There have been several works on detecting metamorphic malware in the past. Our study here will focus on the most recent works which used static analysis on file bytes as the features representing a given file.

Lee et al. [19] extracted structural compression ratio using Gzip (based on Lempel–Ziv) with wavelet analysis on raw bytes of a file to detect metamorphic malware. They used file sequence similarity to compared between malicious and benign files. The limitation of the technique claimed in their study was when there are many compression or packing of a file, detection will be difficult. What we learned from their work was that compression ratio can be used as alternative to entropy when representing a file.

Follow from [19] idea was Ekhtoom et al. [11] where they classify metamorphic malware family using Approximate Minimum Description Length (AMDL) and Best-Compression Neighbor (BCN). However, their experimental results shown no better than 77% accuracy. Another compression based technique combined with machine learning by Alshahwan et al. [3] where a random forest classifier was built using Normalized Compression Distance (NCD) and compressibility ratio features on binary files to detect malware.

Jidigam et al. [14] expanded the eigenvalue analysis from [10,27] using Singular Value Decomposition (SVD) on raw bytes to detect metamorphic malware. They used a metamorphic worm built under Linux operating system by [29] and shown with high detection rate. However, it classified poorly on compiler file generation when compared with Hidden Markov Model (HMM) classification [4].

Bhattacharya et al. [6] exploited information theoretic similarity measures in tandem with wavelet analysis for malware detection. They extracted overlap n-gram of binary bytes and compared using cross entropy, Kullback–Leibler divergence and mean squared error as classifiers. Their experimental data consisted of Kaggle malware [15] with 82.1% accuracy but manage to achieve 100% precision.

When an executable file changes between contents formats (native, encrypted or compressed), its file's representation in entropy also changed. Hence, Wojnowicz et al. [33] applied a wavelet decomposition on file's entropy on byte-level to obtain the entropic wavelet energy spectrum. With this, they can quantify the suspicious pattern variations of a file which can yield predictive information for malware detection.

Radkani et al. [25] introduced entropy based distance (PDME) in K-Nearest Neighbors (KNN) method to classify metamorphic malware families. PDME computes the entropy of opcode frequencies of two files and compare their distance. They results shown high detection on MWOR with padding ratio up to 4.0. However, as the paper mentioned, PDME distance measure might not be able to consider two malware files are similar if they have different opcodes.

Khammas et al. [16] extracted n-gram term frequency directly from binary files and used Support Vector Machine (SVM) to classify metamorphic malware. They first reduced the n-gram feature search space by masking with known malware signature n-gram (Snort) [28] before sending to SVM. Their result achieved high detection rate with low false positive rate and shown superior than commercial anti-virus. However, the Snort sub-signatures have to regularly updated in order to achieve high detection rate.

2.3 Singular value decomposition

We briefly present here the mechanism of Singular Value Decomposition (SVD) which is based on concept of Principal Component Analysis (PCA). In matrix perspective, most often numeric data are represented in rectangular tables form, i.e. $n \times m$ data matrix, with n-dimensional space for n samples. As the size of the matrix becomes really large, it challenges the traditional multivariate analysis approaches. To solve this problem, low-rank matrix approximation or factorization (MF) was introduced. The key idea of MF is that there exists latent structures in the data, by uncovering which we could obtain a compressed representation of the data. By factorizing an original data matrix to low-rank matrices, MF provides a unified method for dimension reduction, clustering, and matrix completion.

Singular Value Decomposition (SVD) is the most familiar way of getting a low-rank matrix approximation using eigenvector analysis and has been widely used in image processing. It is a *eigen-* decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. Hence, given a real-valued $n \times m$ data matrix A, it can be factorized into

$$A = USV^T \tag{1}$$

with $A \in \Re^{n\times m}$, $U \in \Re^{n\times k}$, $S \in \Re^{k\times k}$, and $V^T \in \Re^{k\times m}$, where k is the size of the low-rank or reduced dimensions.

As illustrated in Fig. 1, the matrix U is the row singular vectors of A, the matrix V contains the corresponding column singular vectors of A, and the matrix S is a diagonal matrix of the singular values, in decreasing order of magnitude, which can be viewed as "strength" of corresponding eigenvectors of each specified rank. Columns in both matrices U and V always form the orthogonal set, which can result a square and invertible matrix A.

2.4 Nonnegative matrix factorization

The motivation of having NMF over SVD is because the basis vectors are not orthogonal, which mean each vector can have overlap group and the nonnegative values is more interpreted than negative values. The historical development of NMF has

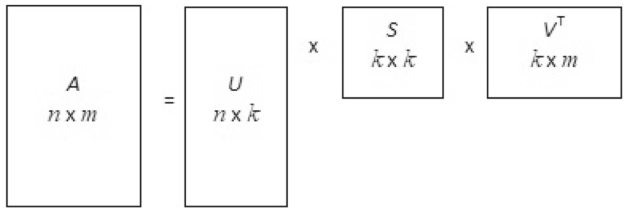


Fig. 1 The SVD decomposition of an $n \times m$ matrix

to go back to [24]. Later, [17] popularized it as a way to find a set of basis images for representing nonnegative images data in facial recognition.

NMF is a dimension reduction technique that decomposes or factors the matrix into a low-rank, sparse and nonnegative factors. The way of NMF "learning" is based on dictionary learning mechanism [12] in which finding a sparse representation of the input data in the form of a linear combination of certain basic elements or patterns. The general formulation of NMF is shown in (2), given a real-valued $V \in \Re^{n\times m}_{\geq 0}$ a matrix consists of m objects and each object is represented with n-dimensional of features, the value of rank, k is selected for dimension reduction, where $k(n + m) < nm$, find nonnegative basis matrix $W \in \Re^{n\times k}_{\geq 0}$ and nonnegative coefficient matrix $H \in \Re^{k\times m}_{\geq 0}$, such that their product approximates the original matrix V.

$$V \approx WH \tag{2}$$

An expanded matrix version of (2) is shown below:

$$\overbrace{\begin{pmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,m} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ V_{n,1} & V_{n,2} & \cdots & V_{n,m} \end{pmatrix}}^{V \text{ with } m \text{ sets of input, } n\times m} \approx \overbrace{\begin{pmatrix} W_{1,1} & \cdots & W_{1,k} \\ W_{2,1} & \cdots & W_{2,k} \\ \vdots & \ddots & \vdots \\ W_{n,1} & \cdots & W_{n,k} \end{pmatrix}}^{W \text{ is } k \text{ basis sets, } n\times k} \times \overbrace{\begin{pmatrix} H_{1,1} & H_{1,2} & \cdots & H_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ H_{k,1} & H_{k,2} & \cdots & H_{k,m} \end{pmatrix}}^{H \text{ is representation, } k\times m} \tag{3}$$

In the approximation (2), W is the dictionary of recurring patterns, which is characteristic of the original data, and every column h_m contains the decomposition or activation coefficients that approximate every v_m onto the dictionary. The nonnegativity of W ensures interpretability of the dictionary, because column patterns W_k and original column V_m belong to the same column space, remain nonnegative. Hence, W is called the basis matrix because its row contains set of basis vectors which can be referred as the local features of V. The nonnegativity of H ensures that WH is nonnegative, tends to produce part-based representations, because subtractive combination is forbidden. The result WH can be interpreted as weighted sum of each of the basis vectors in W, the weights been the corresponding columns of H.

NMF uses the following two-block coordinate descent scheme, that is, they optimize alternatively over one of the two factors, W or H, while keeping the other fixed. The reason is that the subproblem in one factor is convex. The matrices W and H are initialized with positive random values, given a rank k, it learns through multiplicative updates rule by guaranteeing monotonical convergence to a local maximum [18]:

$$W_{ik} \leftarrow W_{ik} .* [V H_{ik}^T ./ (W H H^T)_{ik}] \quad (4)$$
$$H_{kj} \leftarrow H_{kj} .* [W^T V_{kj} ./ (W^T W H_{kj})] \quad (5)$$

for $1 \leq i \leq n$, and $1 \leq j \leq m$, where $.*$ and $./$ denote the element-wise product and division between matrices, respectively. Equations (4), (5) iterate until converge to a local maximum by the following objective function:

$$f(W, H) = \frac{1}{2} \parallel V - WH \parallel_F^2 \quad (6)$$

After the iterations, matrix W contains the learned NMF basis vectors. A new (previously unseen) data matrix V_{new} are projected onto k-dimensional space by fixing W and using one of the following algorithms [20]:

1. randomly initializing H_{new} and iterating (4) and (5) until convergence; or
2. as a least-square solution of $V_{new} = WH_{new}$, that is, $H_{new} = (W^T W)^{-1} W^T V_{new}$

The second algorithm can produce negative entries of H_{new}, since the goal of this study is not to accelerate convergence, we will use the first algorithm.

3 Design and implementation

In this section, we present the details implementation of metamorphic malware detection based on structural analysis through NMF approach using compression ratio and entropy features. An overall architecture of malware detection using NMF technique is shown in Fig. 2.

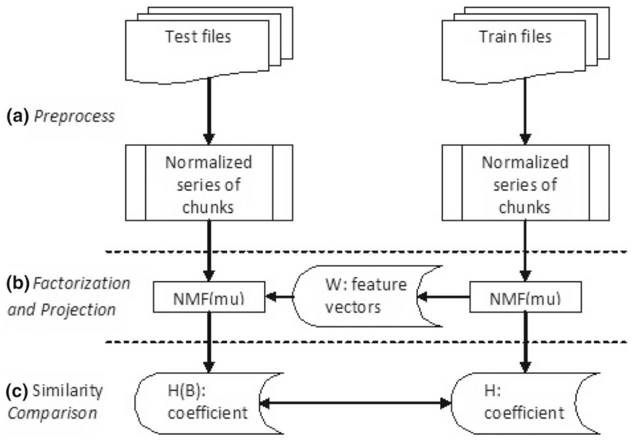

Test files
Train files
(a) Preprocess
Normalized series of chunks
Normalized series of chunks
(b) Factorization and Projection
NMF(mu)
W: feature vectors
NMF(mu)
(c) Similarity Comparison
H(B): coefficient
H: coefficient

Fig. 2 Overall architecture of NMF malware detection

3.1 Determine file size

The first step in our technique is to determine the file size of interest. This is to compromise with the building of data matrix later. Given two sets of files, malware and benign sets, we compute the average file size from each set and use the smaller average size as the file size of interest. We do this because we assume the average file size can represent individual file in each set.

3.2 Splitting a file into byte windows

Next, we split a file into a series of n numbers of byte chunks. A chunk consists of a string of consecutive bytes, with each chunk contains the same number of bytes, N. When dividing a file into a series of chunks, chunks will overlap some amount. Hence, we slide a specified number of byte window before allocating the next chunk. We have determined a chunk size N = 512-byte and sliding window size w = 128-byte as the optimal sizes in our experiment. To facilitate further usage, we also make sure that the series n is of even number extracted from each file. To do this, we used Proscrustean assignment to choose evenly spread chunks from each file in malware or benign sets. This happens when the sliding window reading N bytes at last chunk of a file, it either chopped the last byte if that file is larger than the average file size, or if that file is smaller than average file size, it will append null byte to the last chunk to retain the characteristics of the original data as possible. We then normalized the series to produce same range of values for each file as in Fig. 2(a) shown.

3.3 Feature extraction

Each chunk of bytes that we extracted is then transformed into compression ratio and entropy values. In this step, we adopt gzip utility function [30] from *python* programming language and Shannon's Entropy [8] as in [5,19]. Gzip utilises Lempel Ziv (lz77) coding by replacing repeated instances with references to their earlier occurrences in the data stream. The compression ratio is computed by having the compressed chunk over the uncompressed chunk, this way the distribution of unique byte sequences in each chunk can be measured. Entropy is the measure of uncertainty of the data in a file on a value from 0-8. The lower the entropy, the lower the chances are that the code has been obfuscated or encrypted. Whereas, the higher the entropy, the greater the chances are that the code is obfuscated or encrypted [21]. We used these two feature extraction on the chunk of bytes to represent the structure of a file.

Each file is now represented as a n-vector of compression ratio or entropy values. We then normalize the vectors and form into the data matrix, i.e. malware files matrix V, with compression ratio, as in the form of (7) with n chunks of rows and m files of columns. Test files matrix B is constructed the same way for both features.

$$V = \begin{pmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,m} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ V_{n,1} & V_{n,2} & \cdots & V_{n,m} \end{pmatrix}, B = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,m'} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,m'} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n,1} & B_{n,2} & \cdots & B_{n,m'} \end{pmatrix} \tag{7}$$

where $m\prime$ is the number of test files.

3.4 Nonnegative matrix factorization

By first considering using compression ratio feature, we construct data matrix V by collecting malware files from training set, with column vectors represent m files. The testing set consists of two parts, the malware files that does not used in training set and the benign files set.

We then apply NMF on V by selecting the low-rank k as mentioned in 2.4. The factorization and projection of NMF is shown in Fig. 2(b) where first, we randomly initialize W and H with positive elements. By using equations (4) and (5) we iterate with 500 times. During the update iteration, we added an epsilon 10^{-16} to the denominator of W and H, respectively, for smoothing division. After the iteration, this will give the basis vectors, dictionary matrix W, which contain the basic pattern of malware, and H, the activation or coefficient of malware dictionary W.

Then we project testing data matrix B onto W feature space by first randomly initialize coefficient matrix H_B with positive elements. H_B is with k rows and $m^{'}$ columns iteratively updated by the multiplicative update rule while keeping W intact. Now we are ready for malware detection.

3.5 Malware detection

The detection of malware as shown in Fig. 2 part (c) is done by comparing column distance between H and H_B using *Euclidean* distance measure. This distance could be in range of $[0, \infty]$. So, we convert this distance to similarity score with range $[0, 1]$ by

$$\text{score} = \frac{1}{1 + d(p_1, p_2)} \tag{8}$$

where, $d(p_1, p_2)$ is the distance between each pair of vectors in H with H_B. So, the closer the score to 1, the better the projection of a test file onto the feature space W, and hence the better it is matches the structure of these malware space. Thus, we have a similarity matrix S with rows contain the $m^{'}$ test files and columns contain the m train files. We then take the maximum similarity score for each row to represent the prediction score of each test files.

Having the maximum coefficient similarity scores, we can then set a threshold value to determine if the similarity score is higher or equal to the threshold, which mean very close to vectors in H, then it will be considered as malware; otherwise, it is classified as benign file. We repeat Sects. 3.4 and 3.5 for entropy feature.

4 Experiments and results analysis

Results obtained from the experiments on the similarity score as discussed above which applied to classes of metamorphic malware will be presented in this section. All works are conducted under Windows 10 Professional 64-bit and NMF algorithm [18] are implemented in *Python* language. We will first consider the datasets and then discuss the performance metrics. Then, varies low-rank k will be analyzed and discussed.

4.1 Datasets

Three types of small scale metamorphic malware datasets were considered to evaluate our method and the number of files used are shown in Table 1. These datasets are selected because they contain known metamorphic malware as ground truth and have been used in the literatures [6,14]. The corresponding testing set of benign files is shown in Table 2.

MWOR A highly metamorphic worm generator under Linux OS developed for the purpose of research [29] that employs stealth technique to defeat statistical analysis. Specifically, it inserts arbitrary amounts of dead code extracted from benign files into the generated malware file. By doing so, MWOR retains its malicious functionality while

Table 1 Metamorphic malware datasets

Family	OS	Files		
		Total	Training	Testing
MWOR	Linux	600	480	120
VCL32	Windows	65	52	13
Vundo	Windows	475	380	95

Table 2 Benign datasets

OS	Files	
	Type	Number
Linux	Linux utilities	20
Windows	Cygwin utilities	39
Windows	Applications	2000

has the effect of making the statistics properties of the malware more closely match that of the benign files. The MWOR uses "padding ratio" to specify the ratio of dead code inserted to actual functional worm code. For example, a padding ratio of 4.0 means there is 4 times as much dead code as worm code were inserted in the malware file. We choose and tested 100 MWOR variants from each set of padding ratios of 1.0, 1.5, 2.0, 2.5, 3.0, and 4.0, respectively. Since MWOR is a Linux worm, we collected 20 files from Linux OS under *bin* to represent the benign dataset. The number of files used in benign dataset and MWOR dataset are the same as in [14].

VCL32 Virus Creation Lab for Win32 (VCL32) the only virus generators which we managed to collect from VX Heavens website [13] before it was closed down due to dangerous threaten virus availability. This type of virus family infect the host file by adding a new section or hook on to the API of the host file. To evade detection, it uses simple XOR encryption, simple polymorphic engine or KME32 advanced engine to obfuscate the infected file. According to [34] VCL32 is not a strong metamorphic generator. One of the reason we use it here is to investigate the performance of NMF at the fundamental level of obfuscation. Since this is a windows virus, we collected 34 windows utility files from open source Cygwin [9] to test along with VCL32.

Vundo According to [22], it is a multiple-component family of malware that downloads files without user permission and displays pop-up advertisements. It has been observed that this family uses encryption technique to escape detection and to hinder removal. We collected this malware from [15] under the train subset. There are a total of 475 variants under this metamorphic malware family, therefore, we extracted all the variants in byte representation (hexdump) only format and used xdd [31] to convert the hexdumps to binary executables. For the testing purpose, we collected 2000 windows applications from *filehippo.com* which is a open source of trusted windows applications.

4.2 Evaluation metrics

To evaluate the detection performance of NMF technique, we use receiver operating characteristic (ROC) curves. An ROC curve plots the balance between true positive and false positive rates at every possible threshold. By calculating the area under ROC curve (AUC), we implicitly calculate the false positive rate (FPR) or *1-specificity*, and true positive rate (TPR) or *sensitivity*, at different threshold and combine them into one single metric in a graph. If the AUC is near 0.5, it indicates that the evaluated results of a binary classifier are no better than flipping a coin. An AUC of 1.0 represents ideal performance, in the sense that we can set a threshold for which there will be no false positive or false negatives.

For imbalanced datasets, we use the AUC of the precision-recall curve (APRC) for more informative evaluation as stated in [26]. The precision expresses the ability of our classification to return only relevant instances, i.e. how much malware instances are captured among all those which were captured as malware in this dataset. The precision is defined as:

$$Precision = \frac{tp}{tp + fp}$$

where *fp* indicates the number of false positive.

The recall expresses the ability of a classification to find all relevant instances in a dataset, it is essentially is the true positive rate, i.e. how much malware instances are captured among malware group in this dataset. The recall is defined as:

$$Recall = \frac{tp}{tp + fn}$$

where *tp* indicates the number of true positives and *fn* is the number of false negatives. Similar to ROC, APRC depicts the balance between the positive predictive value and true positive rate.

4.3 Metamorphic malware results

We use five-fold cross validation for each experiment. That is, we partition the malware set into five equal size subsets. We then use four of the subsets for training, reserving the fifth for testing and we repeat the training and testing five times, once for each possible arrangement of the subsets. To obtain a single estimate, we compute the average of the five results from the folds. During our experiments, we assumed low-rank of NMF will carry the most significant information, therefore, we randomly choose ranks k = 1,2,3,7 through out the experiments.

4.3.1 MWOR

As mentioned in Sect. 4.1, MWOR employs a stealth technique which can effectively defeats statistical analysis [29], particularly, MWOR inserts dead code extracted from benign files, which has the effect of making the statistics pattern of malware very close to the benign files. Different chunks distribution using compression ratio and entropy feature extraction of a specific MWOR file before projection onto NMF are shown in Figs. 3 and 4, respectively. The blue color refers to the benign files whereas the red color refers to the malware files. It can be observed that Linux utility files tend to have widen dispersion of compression ratio and entropy values compared to malware files. From the distribution, feature using entropy has more compact and condensed value than compression ratio. This means each chunk of entropy value contains number of bits which are not vary too much

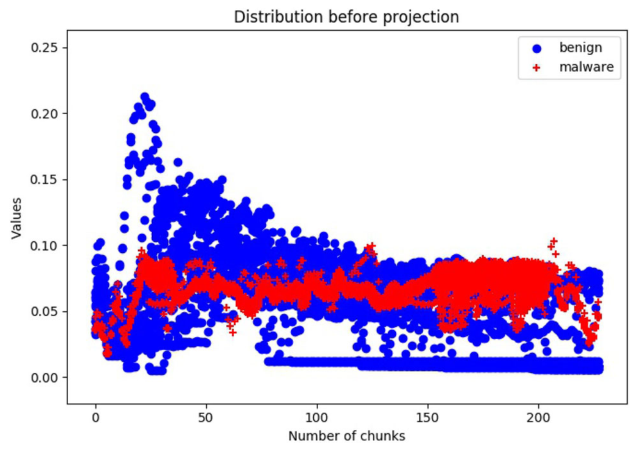


Fig. 3 Compression ratio distribution of malware and benign files for padding 1.0

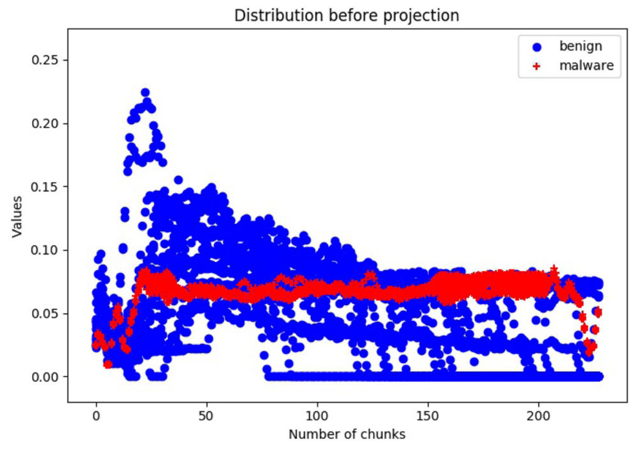


Fig. 4 Entropy distribution of malware and benign files for padding 1.0

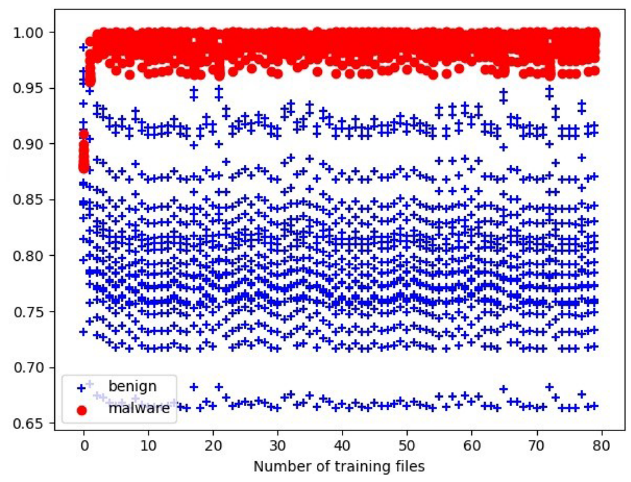


Fig. 5 Pairwise similarity score of compression ratio for padding 1.0

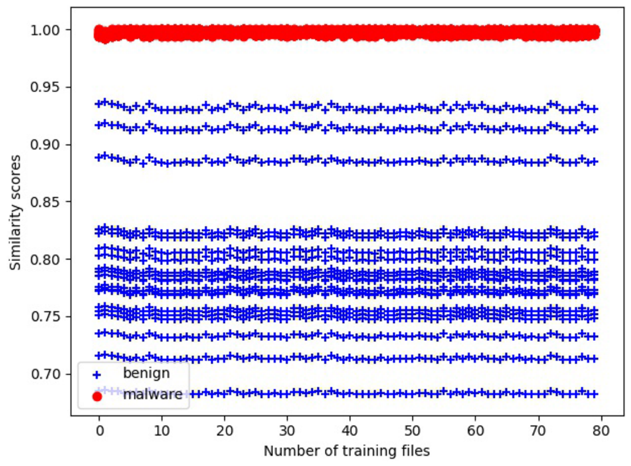


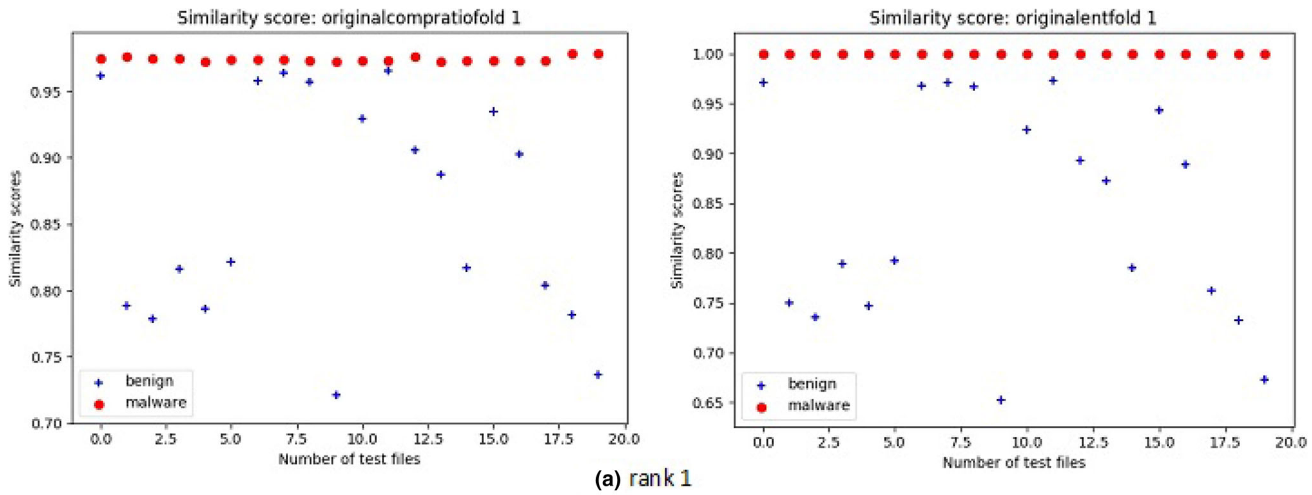
Fig. 6 Pairwise similarity score of entropy for padding 1.0

across themselves. As for compression ratio feature, each chunk contains number of bits differently from each other.

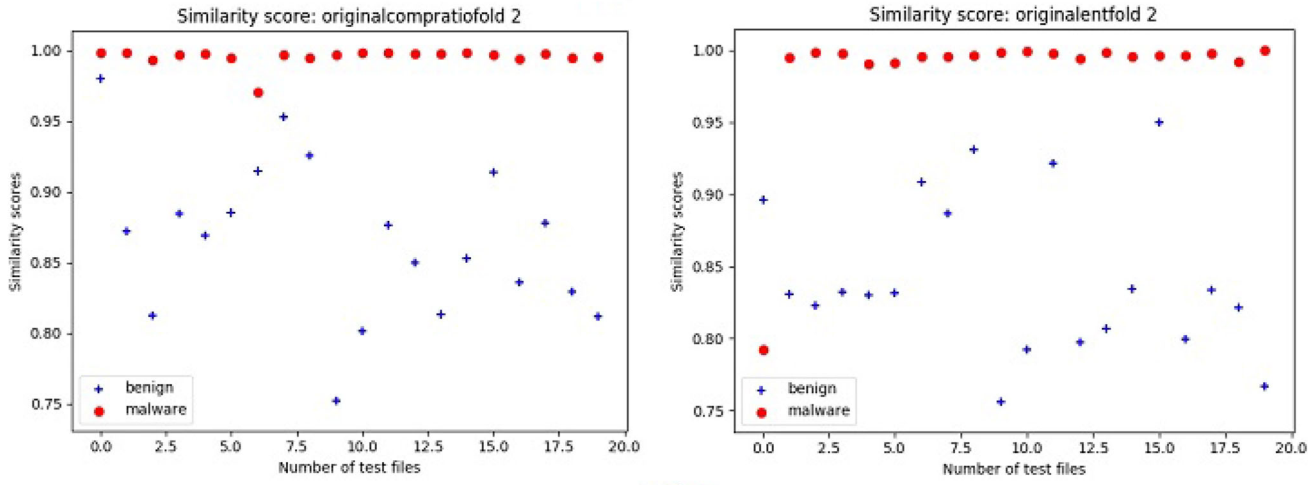
After all the testing files are projected onto NMF following the steps in Sect. 3.4, we get the distance between each coefficient of training set H and each projected coefficient H_B of testing set, as shown in Figs. 5 and 6 for compression ratio and entropy feature, respectively. We can clearly see that malware coefficient scores in testing files condense close to 1.0, which mean they are very similar to the training set, on the other hands, majority of benign files tend to have smaller similarity coefficient scores as they are relatively far from 1.0.

Figure 7 shows the scatterplots of maximum coefficient similarity score of each testing set with the training set for padding ratio 1.0. The left column is the similarity score using compression ratio and the right column is the similarity scores using entropy. Each figure contains 40 test files with red colors are the MWOR files and blue colors are the benign files. We observed that malware files in the testing files have higher similarity score with the ones in training dataset, as for benign files the similarity scores are more scatter around the range of 0.98–0.66. From here, we can see that as rank number increases, says, to 7, it will “expand” the maximum similarity scores as well, this is obvious in Fig. 7(c) which can cause a spoiled optimal threshold.

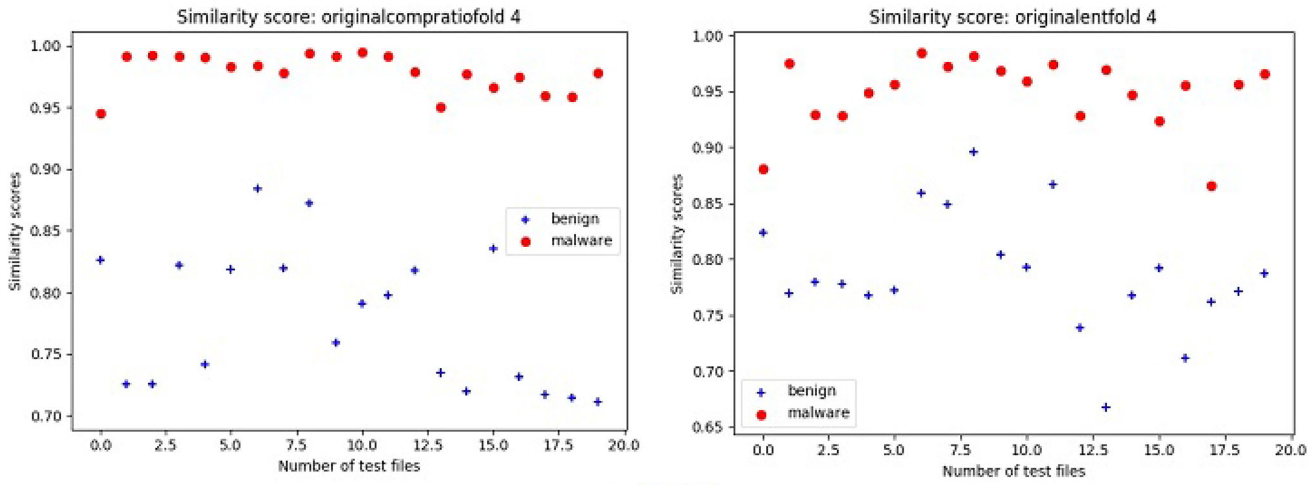
We plot the ROC and collect the mean AUC over the five experiments conducted in the five-fold cross validation as in Table 3. By using the maximum similarity score, we can get 100% accurate prediction at $k = 1$ for entropy feature across all the padding ratios and as for compression ratio feature, it is at padding ratio 1.0 only. This means with low rank, it carries the most significant pattern in the feature W. The lowest AUC for entropy feature 98.30% is at $k = 3$ for padding ratio 1.0. This is not as expected and we need to further investigate the result. As for compression ratio feature, its accuracy performance maintains above 99%. Table 4 shows a comparison of using NMF technique on entropy feature in this paper with previous technique [10,14] which used byte feature on SVD and Eigenvector, respectively, with rank 1



(a) rank 1



(b) rank 3



(c) rank 7

Fig. 7 MWOR padding ratio 1.0 with different NMF ranks

Table 3 MWOR average AUC

Rank	Padding ratio	Compression ratio	Entropy
$k = 1$	1.0	1.000	1.000
	1.5	.9995	1.000
	2.0	.9995	1.000
	2.5	.9995	1.000
	3.0	.9980	1.000
	4.0	.9995	1.000
$k = 2$	1.0	1.000	1.000
	1.5	.9995	1.000
	2.0	.9995	1.000
	2.5	.9990	1.000
	3.0	.9975	.9990
	4.0	.9975	.9975
$k = 3$	1.0	.9975	.9830
	1.5	.9995	.9955
	2.0	.9960	.9925
	2.5	.9970	.9950
	3.0	.9970	.9960
	4.0	.9975	.9995
$k = 7$	1.0	.9965	.9975
	1.5	.9975	.9905
	2.0	.9910	.9990
	2.5	.9975	.9960
	3.0	.9965	.9980
	4.0	.9980	.9940

Table 4 MWOR mean AUC for rank 1

Padding ratio	AUC		
	NMF (entropy)	SVD	Eigenvalue
1.0	1.000	.9999	1.000
1.5	1.000	.9999	1.000
2.0	1.000	.9975	1.000
2.5	1.000	.9966	1.000
3.0	1.000	.9935	1.000
4.0	1.000	.9834	1.000

because it was the best result obtained in their experiments. Entropy feature extraction shown a steady AUC even with high dead code insertion at padding ratio 4.0 as compared to SVD.

Tables 5 and 6 show the results of average AUC of precision-recall curve (APRC) from the five experiments are evaluated at different ranks, k, of NMF using both features. The first column of the tables refers to the padding ratios from 1.0 to 4.0, the second column is the performance metrics where accuracy rate, *AR* and APRC, *PA* are used. We observed with lower rank of $k = 1$ for compression ratio

Table 5 MWOR APRC for compression ratio feature

Padding ratios		$k = 1$	$k = 2$	$k = 3$	$k = 7$
1.0	AR	1.000	1.000	.9900	.9950
	PA	.9983	.9983	.9976	.9966
1.5	AR	.9950	.9950	.9950	.9950
	PA	.9983	.9988	.9999	.9974
2.0	AR	.9950	.9950	.9900	.9950
	PA	.9983	.9991	.9964	.9963
2.5	AR	.9750	.9750	.9000	.9250
	PA	.9981	.9990	.9963	.9966
3.0	AR	.9750	1.000	1.000	1.000
	PA	.9969	.9977	.9977	.9958
4.0	AR	1.000	1.000	.9250	1.000
	PA	.9983	.9977	.9975	.9968

Table 6 MWOR APRC for entropy feature

Padding ratios		$k = 1$	$k = 2$	$k = 3$	$k = 7$
1.0	AR	1.000	1.000	.9700	.9800
	PA	1.000	1.000	.9901	.9959
1.5	AR	1.000	1.000	.9800	.9700
	PA	1.000	1.000	.9955	.9932
2.0	AR	1.000	1.000	.9800	.9950
	PA	1.000	1.000	.9953	.9994
2.5	AR	1.000	1.000	.9750	.8500
	PA	1.000	1.000	.9965	.9974
3.0	AR	1.000	1.000	.9750	.9250
	PA	1.000	.9983	.9952	.9972
4.0	AR	1.000	.9500	.9750	.9250
	PA	1.000	.9979	.9988	.9957

feature, our technique can achieve accurate prediction APRC as high as 99.83%. As the number of k increases, the APRC slightly drops to 99.68% in padding 4.0 which is make sense since there are more dead code inserted. For entropy feature, it can achieve 100% APRC in all the padding ratios for $k = 1$ and the performance drop when the rank increases. These results suggested that entropy as feature is a strong feature compared to compression ratio for this malware family.

4.3.2 VCL32

We test VCL32 malware files with 39 windows Cygwin64 [9] utilities files. The compression ratio and entropy values distribution of this type of malware are shown in Figs. 8 and 9, respectively. From the scatterplots of the distributions, there are clear pattern difference between the malware and the benign files for both features. This implies the obfuscation technique applied in this malware family is not strong because each variant tend to have the same pattern.

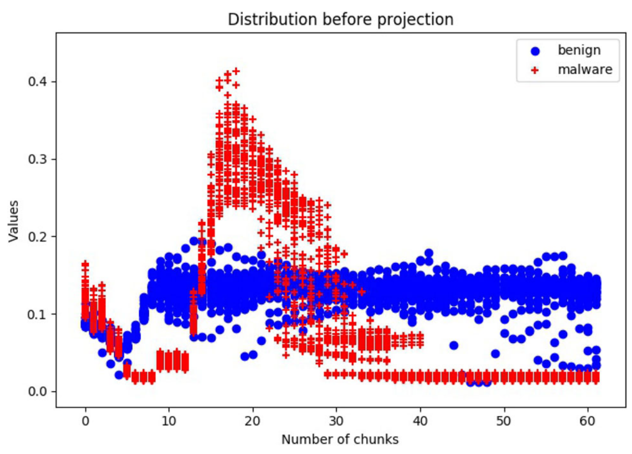


Fig. 8 Compression ratio distribution of VCL32 and benign files

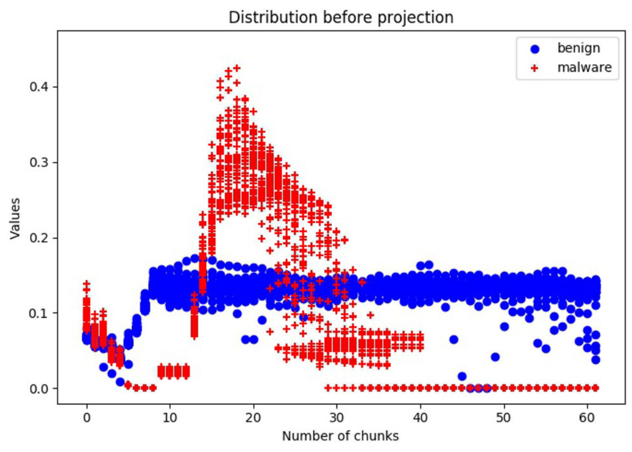


Fig. 9 Entropy distribution of VCL32 and benign files

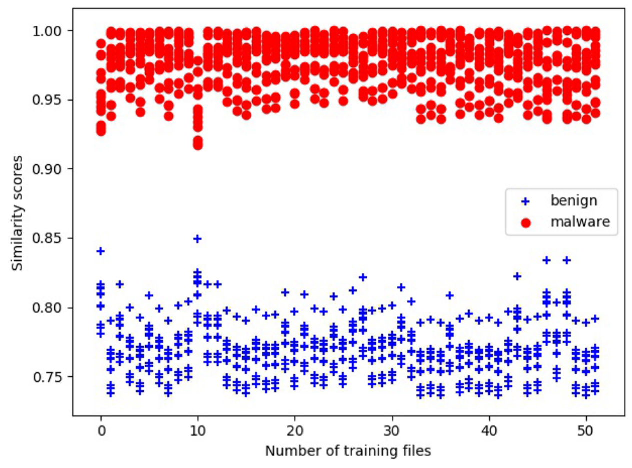


Fig. 10 VCL32 pairwise comparison on compression ratio feature

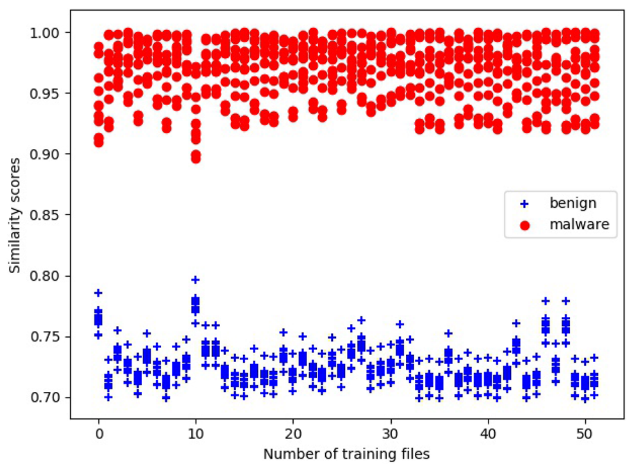


Fig. 11 VCL32 pairwise comparison on entropy feature

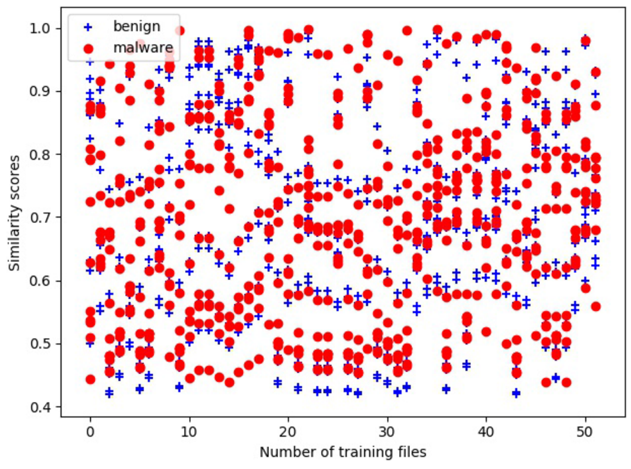


Fig. 12 VCL32 pairwise similarity score of compression ratio feature rank = 2

Both Figs. 10 and 11 show the results of projected distance between the coefficient H of training space and H_B of testing space with rank $k = 1$ for both feature extractions. Here, entropy feature also reveal more compact values than compression ratio values which imply the complexity characteristic of each chunk in entropy is less random than the way compression ratio computed. It can be seen that there is clear separation among the malware and benign files in the testing space with this low-rank, however, we noticed if we increase the rank more than one, the projected distance will be stretched out within range of 1.0–0.4, an example is shown in Fig. 12. With higher number of rank, the distance between a train file in H with respect to all the test files in H_B becomes wider, due to more dimension points are involved during the computation of distance.

The scatterplots of maximum similarity scores with rank 1 for this malware family are shown in Figs. 13 and 14.

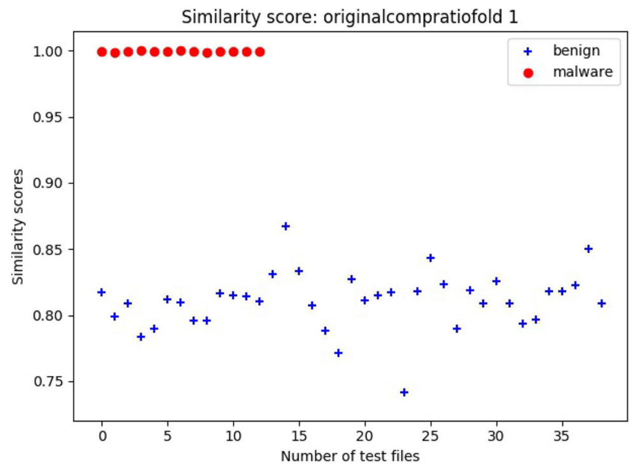


Fig. 13 VCL32 similarity score for compression ratio

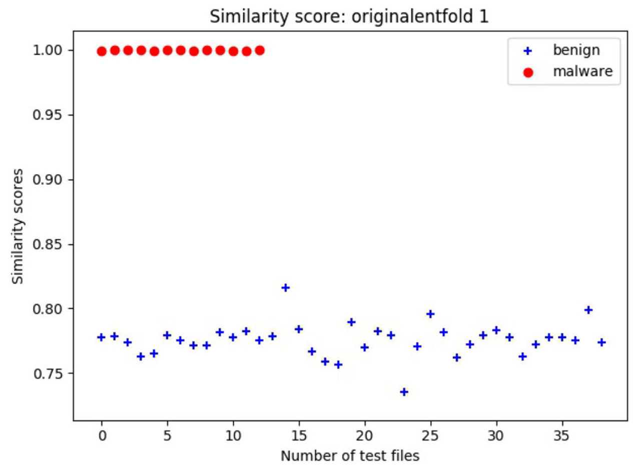


Fig. 14 VCL32 similarity score for entropy

From the figures, show a clear cut of similarity between the malware and benign files in testing set. We also observed that feature extraction using compression ratio has a wider dispersion than using entropy.

The AUC results for both compression ratio and entropy features are shown in Table 7 for different NMF ranks. As the low-rank dimension increases to 7, the accuracy prediction dropped slightly for both features. To further evaluate our technique, the average APRC results are shown in Tables 8 and 9 for both features. Surprising to see that compression ratio feature can still preform better than entropy feature with APRC 99.48 at $k = 7$. This entails VCL32 family may contains variants which are very similar to each other and simple compression method work well for this family (Table 9).

4.3.3 Vundo

For the third dataset, the chunks distribution extracted using both compression ratio and entropy features before project

Table 7 VCL32 mean AUC

Ranks	Compression ratio	Entropy
1	1.000	1.000
2	1.000	1.000
3	1.000	1.000
7	.9980	.9767

Table 8 VCL32 AUC of PRC for compression ratio feature

Evaluation metrics	$k = 1$	$k = 2$	$k = 3$	$k = 7$
Accuracy	1.000	1.000	1.000	.9846
APRC	1.000	1.000	1.000	.9948

Table 9 VCL32 AUC of PRC for entropy feature

Evaluation metrics	$k = 1$	$k = 2$	$k = 3$	$k = 7$
Accuracy	1.000	1.000	1.000	.9846
APRC	1.000	1.000	1.000	.9813

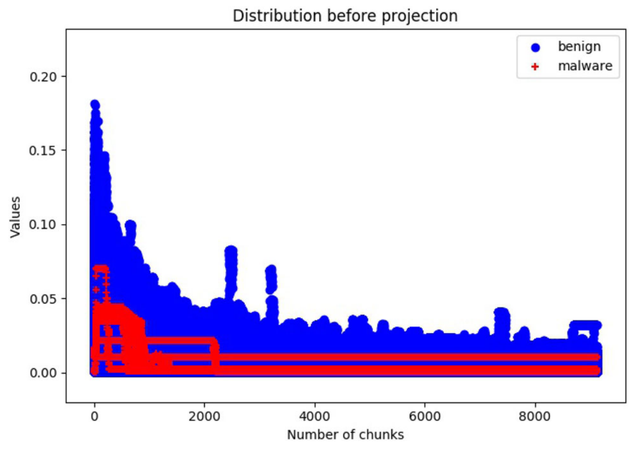


Fig. 15 Compression ratio distribution of malware and benign files before projection

onto NMF are shown in Figs. 15 and 16. We observed that in both feature extraction methods, benign files have higher series of normalized compression ratio or entropy values than malware files. This may due to the benign files we selected in this experiment are mostly setup or application files contain more compressed data than Vundo. From the figures, normalized compression ratio values are in the range 0–0.2, whereas, normalized entropy values span up to 1.0. However, if we look closely, most of the malware training files have relatively small entropy values. We suspect this is due to smaller mean file size that we took from the pool of malware and benign file which may not completely represent the entire file structure.

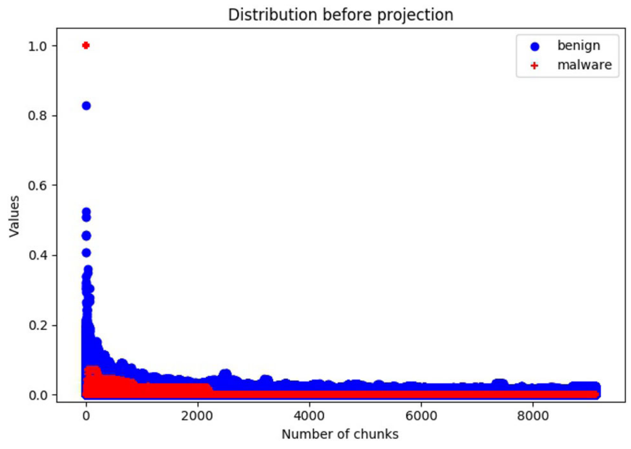


Fig. 16 Entropy distribution of malware and benign files before projection

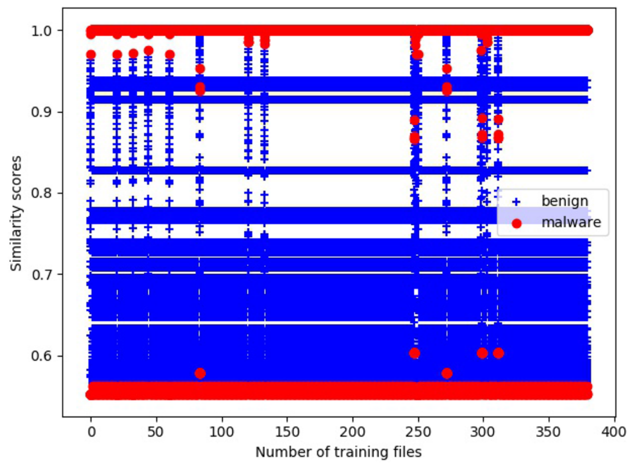


Fig. 17 Vundo pairwise comparison on compression ratio feature

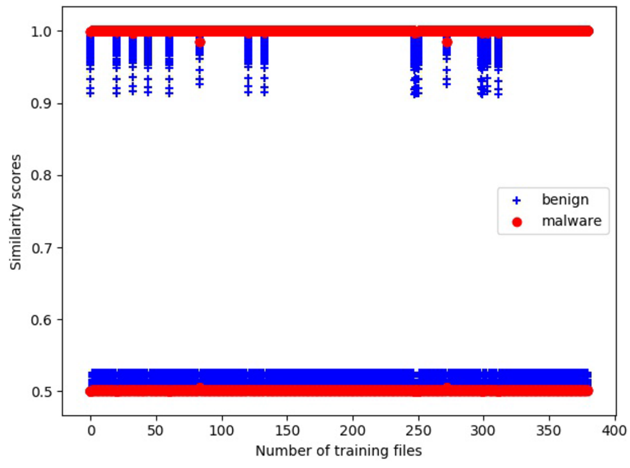


Fig. 18 Vundo pairwise comparison on entropy feature

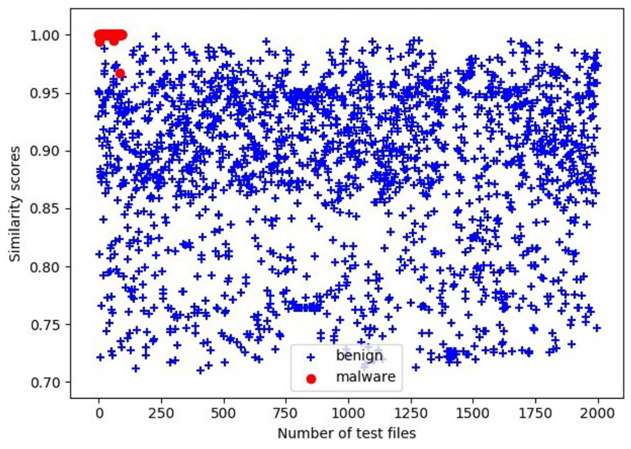


Fig. 19 Projected similarity scores for compression ratio feature with rank 2

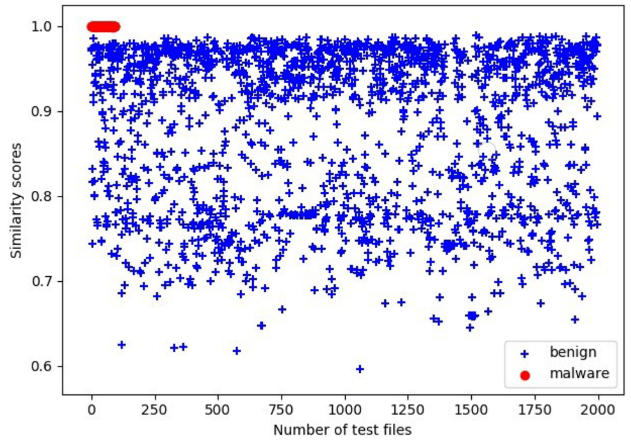


Fig. 20 Projected similarity scores for entropy feature with rank 2

The projected distance of 475 variants of malware with H in the training space and 2000 benign files with H_B in the testing space is shown in Figs. 17 and 18 for both features. The figures depicts with rank 1, the distance of coefficients between training space and testing space can be seen clearly in both features. We also notice in this type of family, with rank > 1, the distance will stretch out within a bound due to many dimension points considered. Figures 19 and 20 show the scatterplots of maximum coefficient similarity scores for compression ratio and entropy feature, respectively, with rank 1. We observed that scores using compression ratio have wider dispersion than using entropy feature, this may indicate entropy capture less randomness of chunks information in this family.

We then obtain the mean AUC after 5-folds in Table 10 for different ranks of NMF. This implies both features can be used to detect malware with rank 1–3 showing a satisfactory results of more than 99% accurate prediction. As for the average accuracy rate and APRC results are shown in Tables 11 and 12. The best rank to be used for both features is at

Table 10 Vundo average AUC

Rank	Compression ratio	Entropy
1	.9990	1.000
2	.9994	.9993
3	.9908	.9926
7	.9606	.7366

Table 11 Vundo compression ratio feature classification performance

Evaluation metrics	$k = 1$	$k = 2$	$k = 3$	$k = 7$
Accuracy	.9911	.9962	.9663	.9221
APRC	.9925	.9961	.9549	.4395

Table 12 Vundo entropy feature classification performance

Evaluation metrics	$k = 1$	$k = 2$	$k = 3$	$k = 7$
Accuracy	1.000	.9996	.9368	.8000
APRC	1.000	.9981	.9420	.2467

$k = 2$ with APRC 99.61% for compression ratio and 99.81% for entropy feature. Again, entropy feature demonstrated to be better feature than compression ratio because even with $k = 1$, it can achieve 100% accurate prediction for this malware family. However, its performance drop sharply when $k = 7$. This indicates for this metamorphic malware it has pattern similar with benign application files and with higher rank, there are redundant data points due to high obfuscation technique applied in this family.

5 Conclusion and future work

In this study we applied NMF technique on three challenging metamorphic malware datasets, in particular with real malware set and confirmed that this approach is as effective as that in [14]. We extracted and compared the file structural with two features, compression ratio using gzip and entropy. Our observations are summarized into the following:

1. Using entropy on series of byte chunks seems to have a smaller value than using compression ratio. According to [21], encrypted or compressed files have higher entropy values (< 6.8). Through out the experiments, we applied entropy on a series of byte chunks instead of the whole file itself and it is yet to proof that this caused the entropy values we got render less randomness among the chunks.
2. Due to the small variation among the byte chunks, entropy could be a better feature compared to compression ratio in our three types of datasets.
3. The files size under experiments are different across these datasets, average file size for MWOR is between 35–68 KB across different padding ratio, VSL32 has the smallest size with only 8 KB, whereas Vundo, after byte conversion, is 256 MB. We suspected the small file size in VCL32 produced such a high classification performance as in Table 8.
4. NMF with low-rank can produce local feature vectors W with better projection than with higher ranks. This denotes NMF can present significant patterns with low rank. However, with standard NMF algorithm, it could produce correlated basis functions as rank number increases and this will deteriorate the detection performance.

Overall, we are able to achieve satisfactory detection with accuracy rate more than 95% in all three datasets. A strength of our technique is that it is applied directly to binary file without the need for pre-processing which most of the time need special domain knowledge for further parsing. By applying NMF on the series of feature chunks, we noticed that NMF technique can be as effective as with [14] for classifying malware files. From the three types of datasets, we notice that entropy as feature achieved higher precision and detection rate at lower rank, 1 or 2 of NMF. This indicates with lower rank of NMF, it carries more significant feature than higher ranks, as higher ranks may contains redundant or "noice" which can make detection difficult.

Several limitations of our method include the direct comparison between each vector during similarity check and since no accelerate of convergence was considered, the computation here is intensive. Our current work only tested on dead code insertion and encryption types of obfuscation. Other types of obfuscation such as register substitution or control flow obfuscation may able to evade this method as the randomness of information in a series of byte chunk is not affected.

For future work, different sizes of sliding window can be studied when project onto NMF. So far, we only tested on 128-byte sliding window. We also need to categorize the file size in training and testing set, as there was no uniform size between these two groups at the moment and this may affect the files under investigated. More tests shall be conducted on a larger scale of real malware files to confirm NMF technique. Another further work, we plan to use NMF as clustering parameters in statistical model such as HMM to study the effectiveness of NMF in grouping malware family.

Acknowledgements This material is partly based upon work supported by the Universiti Putra Malaysia under Putra Grant No. GP/2018/9621600. We also want to thank Prof. Stamp Mark of SJSU, California for sharing the Linux metamorphic malware dataset used in this study.

References

1. 2018 Internet Security Threat Report: https://resource.elq.symantec.com/LP=5840?cid=70138000000rm1eAAA (2018)
2. Alam, S., Horspool, R.N., Traore, I., Sogukpinar, I.: A framework for metamorphic malware analysis and real-time detection. Comput. Secur. **48**, 212–233 (2015)
3. Alshahwan, N., Barr, E.T., Clark, D., Danezis, G.: Detecting malware with information complexity. arXiv:1502.07661 (2015)
4. Austin, T.H., Filiol, E., Josse, S., Stamp, M.: Exploring hidden markov models for virus analysis: a semantic approach. In: 2013 46th Hawaii International Conference on System Sciences (HICSS), pp. 5039–5048. IEEE (2013)
5. Baysa, D., Low, R.M., Stamp, M.: Structural entropy and metamorphic malware. J. Comput. Virol. Hacking Tech. **9**(4), 179–192 (2013)
6. Bhattacharya, S., Menéndez, H.D., Barr, E., Clark, D.: Itect: scalable information theoretic similarity for malware detection. arXiv:1609.02404 (2016)
7. Borello, J.M., Mé, L.: Code obfuscation techniques for metamorphic viruses. J. Comput. Virol. **4**(3), 211–220 (2008)
8. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, Hoboken (2006)
9. Cygwin: Cygwin get that linux feeling—on windows. http://www.cygwin.com/. Accessed 23 July 2018
10. Deshpande, S., Park, Y., Stamp, M.: Eigenvalue analysis for metamorphic detection. J. Comput. Virol. Hacking Tech. **10**(1), 53–65 (2014)
11. Ekhtoom, D., Al-Ayyoub, M., Al-Saleh, M., Alsmirat, M., Hmeidi, I.: A compression-based technique to classify metamorphic malware. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pp. 1–6 (2016). https://doi.org/10.1109/AICCSA.2016.7945801
12. Elad, M., Aharon, M.: Image denoising via sparse and redundant representations over learned dictionaries. IEEE Trans. Image Process. **15**(12), 3736–3745 (2006)
13. http://vxheaven.org/lib/vzo21.html (2001)
14. Jidigam, R.K., Austin, T.H., Stamp, M.: Singular value decomposition and metamorphic detection. J. Comput. Virol. Hacking Tech. **11**(4), 203–216 (2015)
15. Kaggle: Microsoft malware classification challenge (big 2015). http://arxiv.org/abs/1802.10135 (2016). Accessed 23 July 2018
16. Khammas, B.M., Monemi, A., Ismail, I., Nor, S.M., Marsono, M.: Metamorphic malware detection based on support vector machine classification of malware sub-signatures. TELKOMNIKA (Telecommun. Comput. Electron. Control) **14**(3), 1157–1165 (2016)
17. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755), 788 (1999)
18. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: Advances in Neural Information Processing Systems, pp. 556–562 (2001)
19. Lee, J., Austin, T.H., Stamp, M.: Compression-based analysis of metamorphic malware. Int. J. Secur. Netw. **10**(2), 124–136 (2015)
20. Li, Y., Ngom, A.: Non-negative matrix and tensor factorization based classification of clinical microarray gene expression data. In: 2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 438–443. IEEE (2010)
21. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Secur. Priv. **5**(2), 40–45 (2007)
22. Microsoft: Windows defender security intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FVundo (2012). Accessed 16 May 2016
23. Mohan, V., Hamlen, K.W.: Frankenstein: stitching malware from benign binaries. WOOT **12**, 77–84 (2012)
24. Paatero, P., Tapper, U.: Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values. Environmetrics **5**(2), 111–126 (1994)
25. Radkani, E., Hashemi, S., Keshavarz-Haddad, A., Haeri, M.A.: An entropy-based distance measure for analyzing and detecting metamorphic malware. Appl. Intell., 1–11 (2017)
26. Saito, T., Rehmsmeier, M.: The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PloS ONE **10**(3), e0118432 (2015)
27. Saleh, M.E., Mohamed, A.B., Nabi, A.A.: Eigenviruses for metamorphic virus recognition. IET Inf. Secur. **5**(4), 191–198 (2011)
28. Snort: Snort. https://www.snort.org/
29. Sridhara, S.M., Stamp, M.: Metamorphic worm that carries its own morphing engine. J. Comput. Virol. Hacking Tech. **9**(2), 49–58 (2013)
30. Support for Gzip Files: https://docs.python.org/2/library/gzip.html/ (2017). Accessed 28 Nov 2017
31. SysTutorials: xxd (1)—linux man pages. https://www.systutorials.com/docs/linux/man/1-xxd/
32. Szor, P., Ferrie, P.: Hunting for metamorphic, symantec security response. https://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf (2003)
33. Wojnowicz, M., Chisholm, G., Wolff, M., Zhao, X.: Wavelet decomposition of software entropy reveals symptoms of malicious code. J. Innov. Digit. Ecosyst. **3**(2), 130–140 (2016)
34. Wong, W.: Analysis and detection of metamorphic computer viruses. Ph.D. thesis, San Jose State University (2006)
35. You, I., Yim, K.: Malware obfuscation techniques: a brief survey. In: 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), pp. 297–300. IEEE (2010)