A Hypercuboid-Based Machine Learning Algorithm for Malware Classification

Thi Thu Trang Nguyen
VNU University of Engineering and Technology

Dai Tho Nguyen[1]
VNU University of Engineering and Technology

Duy Loi Vu
VNU University of Engineering and Technology

[1] Contact Author: nguyendaitho@vnu.edu.vn

***Abstract*— Malware attacks have been among the most serious threats to cyber security in the last decade. Anti-malware software can help safeguard information systems and minimize their exposure to the malware. Most of anti-malware programs detect malware instances based on signature or pattern matching. Data mining and machine learning techniques can be used to automatically detect models and patterns behind different types of malware variants. However, traditional machine-based learning techniques such as SVM, decision trees and naive Bayes seem to be only suitable for detecting malicious code, not effective enough for complex problems such as classification. In this article, we propose a new prototype extraction method for non-traditional prototype-based machine learning classification. The prototypes are extracted using hypercuboids. Each hypercuboid covers all training data points of a malware family. Then we choose the data points nearest to the hyperplanes as the prototypes. Malware samples will be classified based on the distances to the prototypes. Experiments results show that our proposition leads to F1 score of 96.5% for classification of known malware and 97.7% for classification of unknown malware, both better than the original prototype-based classification method.**

Keywords*— *Malware classification, machine learning, k-nearest neighbors algorithms, prototype-based learning, hypercuboids

I. INTRODUCTION

Malware, also known as malicious code, refers to a program covertly inserted into a system with the intent of compromising the security of the victim's data, applications, or operating systems. There are many types of malware such as Trojan, worm, spyware, rootkit... Along with the rapid development of the Internet, the number of malware is increasing day by day. Although significant improvements have been made in anti-malware solutions, more and more sophisticated propagation technology and detection evasion techniques have been used. Therefore, malware is still a significant threat to computer systems. Detection and analysis of the malware behaviors are critical to the minimization of the consequential damages.

Malware analysis refers to the process of determining their purposes, behaviors, payload methods, and propagation mechanisms. The analysis is usually performed manually by domain experts. Recently, the research community seeks to automate parts of the process to aid the domain experts in their work and reduce the time required. According to [1], an automatic malware analysis aims to achieve one of the following objectives: malware detection, similarity detection, and category detection. Some examples of malware detection include [2], [8], and [9]. Refer to [3], [5], and [10] for examples of similarity analysis. Malware classification is studied in [4], [6], [11], [14], and [16].

In this work, we focus on solving the malware classification problem by using specifically widely known supervised machine learning methods. According to [13], there exist two distinguished approaches to supervised learning: model-based learning and instance-based learning. The model-based learning algorithms such as SVM (support vector machine), Naive Bayes, and decision trees seek to construct a model that generalizes the training data. If the input data is too complex, it can become very difficult to find an appropriate model for representing the data instances. Meanwhile, the instance-based learning algorithms such as k-NN (k Nearest Neighbors), radial basis function networks, and kernel machines make the classification decisions by comparing the new malware instances with the representative instances extracted from the training dataset during the training phase. Each representative instance, also called prototypes, belongs to a malware family and is considered as a representative of that family. Each malware family can have more than one representative instance. As there can be more flexibility in the selection of individual representative instances compared to construction of a single model, we opt for instance-based learning for malware classification.

In particular, our work is inspired by the instance-based classification methods proposed in [4], [6], and [12]. The method in [6] uses information about the n-grams of system calls to embed the behaviors of malware samples into a vector space called feature space. Each component of a feature vector is a value 0 or 1 that denotes the presence or absence of the respective system call n-gram. The proposed algorithm includes a prototype extraction phase for finding the prototypes representing the clusters/classes of malware samples; a clustering phase that uses the prototypes to group unknown samples into clusters; and a classification phase that uses also the prototypes but this time to predict class labels for known malicious codes, and detect novel malicious codes. The method in [4], called Dendroid, classifies malware instances by using text mining and information retrieval techniques. Each malware family is represented by a family feature vector, a concept similar to prototypes but while each prototype must correspond to malware instance, it is not necessarily so for the family feature vectors. After the family feature vector extraction phase, a 1-NN (One Nearest Neighbors) algorithm is used for classification. In [12] the authors also construct family feature vectors for representing malware families. Then, unknown malware are classified based on the similarities with the family feature vectors.

In this paper, we propose a new prototype extraction method for the above process. During the training phase, we extract prototypes only from the labeled malware samples instead of all the samples in the dataset. Then a hypercuboid is constructed for each malware class, each hypercuboid surrounds all the samples of a malware class. We choose the

978-1-6654-0435-8/21/$31.00 ©2021 IEEE

data points on the hyperplanes as prototypes. We evaluate the performances of our proposed method for classifying known classes and rejecting unknown classes by using the F1-measure metric computed from precision and recall. The experimental results show F1-mesures of 96.5% for classification of known malware and 97.7% for classification of unknown malware, both better than Rieck et al.'s method.

II. Previous prototype-based malware classification methods

The work in [6] concerns open-world classification as the proposed prototype-based method can detect and classify instances of unknown malware classes. The compression of each malware family into a small set of prototypes representing the whole family allows to reduce the volume of data to be processed and the classification time. Another advantage is the possibility of incremental learning, where the the sets of prototypes can be easily updated when new training data is added, without the need for retraining from scratch.

The classification process goes through the following steps:

- Running in sandbox
- Embedding of behaviour
- Prototype extraction
- Classification using prototypes
- Clustering using prototypes

During the first step, the malware sample to be classified is run and monitored within a sandbox environment to collect the generated sequence of system calls. At the next step, the sequence of system calls is represented as a binary feature vector of n-grams. Each component of the feature vector is a value 0 or 1 that denotes the presence or absence of the respective system call n-gram. Then, the feature vector is normalized by dividing it by its Euclidean length to create a vector of length 1. During the classification using prototypes step, the current malware sample is classified based on the geometric distances between of its feature vector and the existing prototype vectors. The shortest Euclidean distance from the unknown malware sample to an existing prototype is computed. The sample is put into the corresponding malware class if this distance is less than a predetermined threshold d_r. Otherwise, it serves as an input to the next prototype extraction phase, the purpose of which is to find new prototypes for representing the remaining malware classes. The prototypes are extracted from the set of unclassified malware samples using an algorithm adapted from the linear-time algorithm proposed by Gonzalez in [15]. The first prototype can be predefined or randomly chosen. All data points near each prototype will be grouped together into a cluster. "Near" here means the distances from those data points to the given prototype must be smaller than a predetermined threshold d_p. Then, the farthest point from the current prototypes is chosen as a new prototype. The procedure is repeated until each data point belongs to a cluster. Following the prototype extraction phase is the clustering phase, during which the prototypes are regrouped together to create large enough clusters. The idea behind the merge procedure is that the smaller the distance between the two prototypes is, the more likely they belong to the same family. The extracted prototypes will be used for the classification of the next malware sample.

In [4] Suarez-Tangil et al. propose a malware classification method based on the so called family feature vectors, using text mining and information retrieval techniques for Android malware classification. The method contains three phases:

- Modeling phase
- Classification phase
- Analysis phase

During the modeling phase, for each malware family, a family feature vector is calculated from all its instances. Consequently, each family has only one representative feature vector. If there are for example 12 malware families, then 12 family feature vectors will be constructed and used. A family feature vector can correspond to a virtual data point or a real data point. They are used as references when classifying malware samples. At the classification phase, each malware sample is modeled as a feature vector, then it is assigned to a certain family if its feature vector is the nearest to that family's representative feature vector. In the final analysis phase, hierarchical clustering and linkage analysis techniques are used to recognize the relationships between the different types of malware instances. Dendroid method is not an open-world classification as every input malware sample is sure to be assigned to a known family, but the concept of representative feature vectors is quite similar to prototypes.

In [12], Shrestha et al. also use prototypes to represent malware families. Each family of malware is represented by only one feature vector, just like Suarez-Tangil et al.'s method. In order to build prototypes, all the files belonging to a same malware family are merged to form a new one. The authors compute the tf-idf value of each printable string in this file. Then, they construct a prototype vector from the printable strings and the corresponding tf-idf values for representing the malware family. The procedure is repeated for each of the malware families in the training dataset. The assignment of a family label to a malware sample is made based on the comparison results of the cosine similarities between its feature vector and the prototype vectors. The malware sample will get classified into the family that has the highest similarity score with it.

Dendroid [2] and the method proposed by P. Shrestha et al. [3] have quite different training methods compared to [6]. For each malware family, a single representative feature vector is built only from the training data belonging to that family instead of several instances being chosen from the whole training dataset.

III. Our hypercuboid-based proposition for prototype extraction

We find that the previous prototype extraction methods exhibit the following problems.

First, the prototypes in [6] are extracted from all malware samples in the training dataset. That can lead to mistakes such as grouping samples that are not in the same malware family into the same cluster or choosing as the representative prototype for a group a sample that is not in the same malware family as the majority of samples in the group. The methods in [4] and [12], which construct a prototype by using only the samples in the same malware family can overcome this problem of the first method. However, they only use one point

to represent a family, this might lead to loss of too much information.

The second problem is that prototypes in the distance–based prototype extraction method are selected purely based on distances, without using any information on directions, this might lead to directional bias in the multi-dimensional space. In case a sample to be classified should belong to a certain family, but in the direction to this sample there are not any prototypes, the malware may be classified wrongly to another family.

We propose a method for better representing the malware families with a novel concept of prototypes constructed based on hypercuboids. For each malware family, we build a hypercuboid that surrounds all its data points. The faces of the hypercuboid represent the directions in the feature vector space. We choose as prototypes the closest points to the hyperplanes of the hypercuboid. Using this technique we can obtain prototypes in all directions, thus break the directional bias that may occur with the previous prototype extraction methods. Additionally, as the number of prototypes extracted from a malware family somehow corresponds to its size and variety in the feature vector space, we can expect to have enough prototypes for representing it.

Figures 1, 2, and 3 illustrate the differences in prototype extraction between our proposition and the previous methods.

Note:

Δ: class 1, O: class 2, □: class 3

Bold points are prototypes (or family feature points) in the training data set

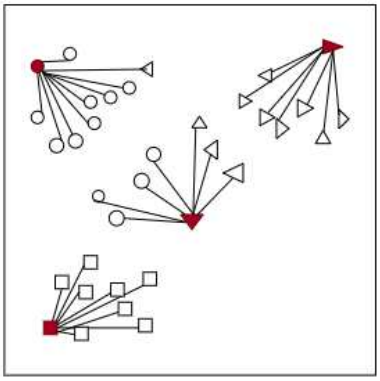


Fig. 1. Prototype extraction in [6]

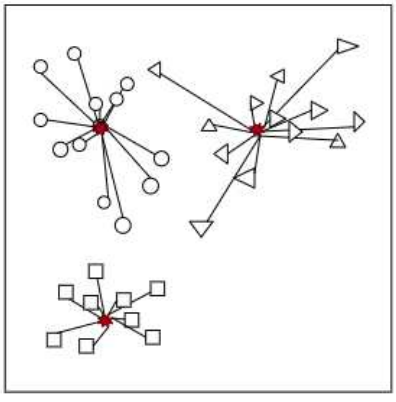


Fig. 2. Extraction of family feature points/prototypes in [4] and [12]

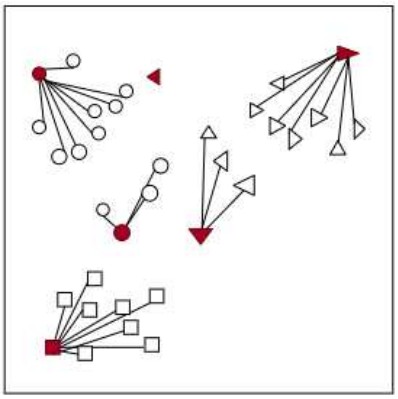
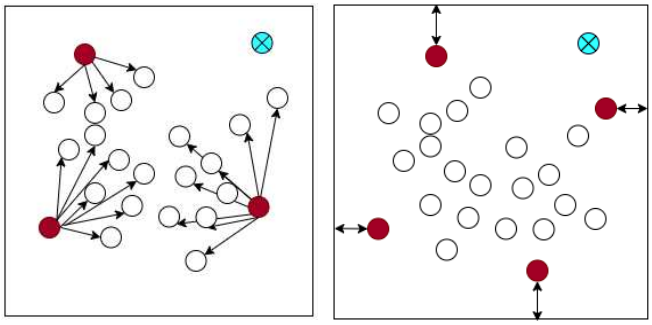


Fig. 3. Our proposition for prototype extraction

Figure 1 illustrates the probable situations that the points of class Δ are clustered in the same group as the prototype of class O, and that the prototype of class Δ represents many points of class O.

In Figure 2, each family feature point is aggregated from all data points in the same family. A family feature point can be a real data points or a virtual point that does not exist in the corresponding family. All data points of a family are represented by only one family feature point.

Figure 3 shows that all prototypes of the class □ are data points of this class, and the same goes for classes O and Δ. Although there is a point not belonging to any clusters, it represents itself, instead of falsely representing other classes. All points in each cluster are characterized by the prototypes belonging to the cluster itself. Moreover, each family is represented by many prototypes so we do not lose much information.



a) Prototypes are biased in direction

b) Our prototypes are not biased in direction

Fig. 4. Comparison of our prototypes and the prototypes in [6]

Figure 4.a illustrates a directional bias in prototype extraction. The sample data point to be classified (circle with cross symbol) is located in the direction where there are no prototypes, so it can be easily misclassified.

Figure 4.b illustrates the prototypes that are the closest to the faces of a hypercuboid (for the two-dimensional space, the hypercuboid is a rectangle). We can find prototypes in all directions. For any malware family, there is always a prototype near any sample data point belonging to it (circle with cross symbol) in any direction.

The idea of our algorithm is as follows. For each family, build a hypercuboid around all its data points. The hypercuboid is constructed from two initial points, one with the minimum coordinates in all dimensions and the other with

the maximum coordinates. Then, from these two outmost vertices, we draw the lines parallel to the coordinate axes, the lines intersect creating new vertices, from new vertices, we continue to draw new lines parallel to the coordinate axes, and so on. Finally, we obtain a hypercuboid surrounding all the given elements in the malware family. Once constructed, the data points on the hyperplanes of a hypercuboid will be chosen as prototypes for the corresponding malware family.

Figure 5 shows the pseudocode of our prototype extraction algorithm.

Algorithm for prototype extraction

prototypes ← Ø

for ***l*** ∈ families

prototypes[l] ← Ø

for ***i*** ∈ dimensions in the feature space

min[l] = the minimum value in dimension ***i*** of data points in family ***l***

max[l] = the maximum value in dimension ***i*** of data points in family ***l***

for ***x*** ∈ data points in family ***l***

if ***x[i]*** = ***min[l]*** or ***x[i]*** = ***max[l]***

prototypes[l] += ***x***

break

Fig. 5. Prototype extraction using hypercuboids

IV. Experiments and Evaluation

A. Dataset

We use the dataset referenced in Rieck et al.'s work [6] for our experiments. This dataset contains the malware samples extracted from the large malware database maintained at the CWSandbox website and labeled by 6 different well-known anti-virus products. After removing classes with too few (less than 20) samples and too many (more than 300) samples, 3133 behavioral samples were obtained. The malware samples are grouped into 24 classes. Some typical malware families in the dataset are Bancos with 48 samples, Podnuha and Rotator with 300 samples, Posion with 26 samples, Sality with 85 samples, and Virus with 202 samples. As we focus on malware classification rather than detection, the used dataset contains only malicious samples, no benign samples.

B. Feature extraction

From the dataset, we extract the level 1 system call sequences (only the names of the system calls, no argument information) and find that there are overall 85 system calls. With the sequence of system calls obtained from each malware sample, we proceed to the extraction of the corresponding binary vector of 2-grams (2-grams mean 2 consecutive system calls), each component of which is a value 0 or 1 denoting the presence or absence of the respective system call 2-gram.

For example, let $A = \{a_1, a_2\}$ be a set of all possible system calls, then $S = \{a_1a_1, a_1a_2, a_2a_1, a_2a_2\}$ is the set of all possible 2-grams. Suppose we have a system call sequence $a_1a_2a_1a_1a_1$. Table I shows the presence or absence of each 2-gram in this call sequence. Then, the corresponding feature vector is $x = (1, 1, 1, 0)$. We normalize the vector x so that it has unit length.

$$|x| = \sqrt{1^2 + 1^2 + 1^2 + 0^2} = \sqrt{3}$$

The normalized version of x is $x = (\frac{1}{|x|}, \frac{1}{|x|}, \frac{1}{|x|}, \frac{0}{|x|}) = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, 0)$

TABLE I. The presence or absence of 2-gram

2-gram	a_1a_1	a_1a_2	a_2a_1	a_2a_2
presence	1	1	1	0

The normalized feature vector is used to represent each malware sample.

The reason for the choice of binary vectors is that through experiments, we found that it allows to obtain higher efficiency than the use of frequency vectors. As there are totally 85 different system calls in the dataset, the size of the feature vector space is 85×85. But since there are many zero components, it is possible to extract and compare the feature vectors in linear time. Refer to [7] for a detailed discussion of the linear time algorithms for comparison of sequential data.

C. Evaluation

We evaluate our proposed hypercuboid-based prototype extraction method in terms of the abilities to classify known classes of malware and recognize new classes, using the F1 score metric, which is a mix of two standard measures Precision and Recall. The Precision indicates how many of the predicted positive cases are actually positive. The Recall shows how many of the actual positives are labeled as positives. F1 score balances Precision and Recall. For example, a classifier with Precision = Recall = 0.5 is better than another classifier with Precision = 0.2 and Recall = 0.8 according to the F1 score measure.

Following are the definitions of the metrics:

- Number of true positives for class *i*

  TP_i = The number of samples belonging to class *i* that are correctly assigned to class *i*

- Number of false positives for class *i*

  FP_i = The number of samples not belonging to class *i* that are incorrectly assigned to class *i*

- Number of true negatives for class *i*
  TN_i = The number of samples not belonging to class *i* that are correctly not assigned to class *i*

- Number of false positives for class *i*
  FN_i = The number of samples belonging to class *i* but are incorrectly not assigned to class *i*

- Average precision:

$$P = \frac{\sum TP_i}{\sum (TP_i + FP_i)} \quad (1)$$

- Average recall:

$$R = \frac{\sum TPi}{\sum(TP_i + FN_i)} \quad (2)$$

- F1 score:

$$F1 = \frac{2 * P * R}{P + R} \quad (3)$$

The F1 score varies in the range from 0 to 1. The higher the value is, the better the classification is. We evaluate the following F1 score metrics.

- F1 score for classification of known malware classes

  F_k = F1 score on a dataset with known labels

- F1 score for rejection of unknown malware classes (not appearing in the training phase)

  F_u = F1 score on an unlabeled dataset

Both the ability to identify novel malware (F_u) and the ability to classify known malware (F_k) depend on the choice of a distance threshold (d_r) for separating unknown malware samples from known malware classes. The bigger we choose d_r, the lower F_u will be and the higher F_k, because the less malware samples will be rejected. On contrary, the smaller d_r is, the higher F_u will be and the lower F_k. In the training phase, we determine the optimal value for d_r such that both F_k and F_u are the highest possible.

The overall 24 class dataset is divided in the ratio of 70% for training and 30% for testing. But only 18 of the 24 classes are used in the training phase and for evaluating F_k. The remaining 6 classes in the testing part are used for evaluating F_u. The training part of the dataset doesn't contain any instances of these 6 classes. We do not divide the total number of classes into two halves as in [6], as the number of new malware families is usually small in comparison with the number of old known malware families. We perform each experiment 10 times and take the average of the results for our proposed method and the distance–based prototype extraction method in [6]. Table II show the obtained average results for d_r varying from 0 to 1.

TABLE II. CLASSIFICATION PERFORMANCES WITH VARIED D_R VALUES

d_r	F_k-proposed	F_u-proposed	F_k-[6]	F_u-[6]
0	0	1	0	1
0.1	0.473	0.983	0.133	0.994
0.2	0.743	0.983	0.251	0.994
0.3	0.931	0.983	0.831	0.994
0.4	0.965	0.977	0.890	0.992
0.5	0.977	0.880	0.932	0.910
0.6	0.987	0.805	0.940	0.804
0.7	0.991	0.743	0.943	0.716
0.8	0.994	0.685	0.941	0.716
0.9	0.994	0.499	0.942	0.544
1	0.995	0.420	0.942	0.496

Our goal is to choose a threshold value d_r such that both F_k and F_u measures are the highest possible. The best d_r value for Rieck et al.'s method [6] is 0.5, which corresponds to F_k = 0.932 (93.2%) and F_u = 0.901 (90.1%). Meanwhile, the F1 score measures of our method are optimal at d_r = 0.4 with F_k increased to 0.965 (96.5%) and F_u increased to 0.977 (97.7%). Therefore, we can conclude that our method is more efficient than Rieck et al.'s method. Figure 6 shows the F1 score – distance threshold graphs of the two methods.

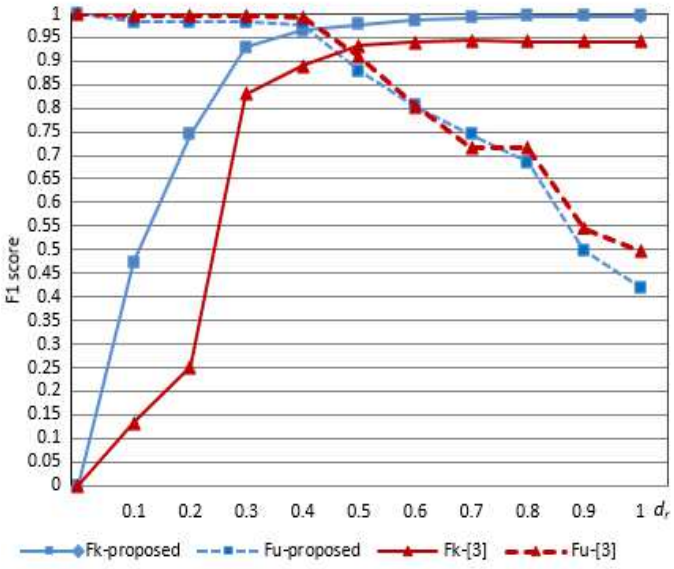


Fig. 6. Comparison of the F1 scores

V. CONCLUSION

We presented a simple novel method for prototype extraction in view of open-world malware classification. Experimental results showed that our proposed method is quite efficient, achieving F1-micro scores of 96.5% for classification of know malware and 97.7% for detection of new malware, overcoming the disadvantages of the original prototype-based method. But our method of extracting the prototypes using hypercuboids might be suitable only when data points are clustered in distinct small areas. In these cases, the data points nearest to the hyperplanes will properly represent each malware family because the all its data points are likely located in the vicinity of the prototypes. But in the cases where data points of a malware family are distributed over a large area, the data points in the center can be far from the prototypes. Therefore, more careful considerations should be carried out for our proposition to be effective for more diverse datasets.

ACKNOWLEDGMENT

Nguyen Thi Thu Trang was funded by Vingroup Joint Stock Company and supported by the Domestic Master/ PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), Vingroup Big Data Institute (VINBIGDATA), code VINIF.2020.ThS.62.

REFERENCES

[1] Daniele Ucci, Leonardo Aniello, Roberto Baldoni: Survey of machine learning techniques for malware analysis. Computers & Security 81: 123-147 (2019).

[2] F. Ahmed, H. Hameed, M. Z. Shafiq, M. Farooq, Using spatio-temporalinformation in api calls with machine learning algorithms for malwaredetection, in: Proceedings of the 2nd ACM workshop on Security andartificial intelligence, ACM, 2009, pp. 55–62.

[3] G. Liang, J. Pang, C. Dai, A behavior-based malware variant classificationtechnique, International Journal of Information and Education Technol-ogy 6 (4) (2016) 291.

[4] Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez, Jorge Blasco: Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. Expert Systems with Applications,Volume 41, Issue 4, Part 1, 2014, Pages 1104-1117, ISSN 0957-4174.

[5] J. Upchurch, X. Zhou, Variant: a malware similarity testing framework,in: 2015 10th International Conference on Malicious and Unwanted Soft-ware (MALWARE), IEEE, 2015, pp. 31–39.

[6] K.Rieck, P.Trinius, T.Holz: Automatic Analysis of Malware Behavior using Machine Learning. Journal of Computer Security (JCS), 19 (4) 639-668, 2011.

[7] K. Rieck and P. Laskov: Linear-time computation of similarity measures for sequential data. Journal of Machine Learning Research, 9(Jan):23–48, 2008.

[8] M. G. Schultz, E. Eskin, F. Zadok, S. J. Stolfo, Data mining methods fordetection of new malicious executables, in: Security and Privacy, 2001. SP 2001. Proceedings. 2001 IEEE Symposium on, 2001, pp. 38–49.

[9] M. Kruczkowski, E. N. Szynkiewicz, Support vector machine for malwareanalysis and classification, in: Web Intelligence (WI) and Intelligent AgentTechnologies (IAT), IEEE Computer Society, 2014, pp. 415–420.

[10] P. Khodamoradi, M. Fazlali, F. Mardukhi, M. Nosrati, Heuristic metamor-phic malware detection based on statistics of assembly instructions usingclassification algorithms, in: Computer Architecture and Digital Systems(CADS), 2015 18th CSI International Symposium on, IEEE, 2015, pp.1–6.

[11] P. M. Comar, L. Liu, S. Saha, P. N. Tan, A. Nucci, Combining supervisedand unsupervised learning for zero-day malware detection, in: INFOCOM,2013 Proceedings IEEE, 2013, pp. 2022–2030.

[12] Prasha Shrestha,Suraj Maharajan,Gabriela Ramirez de la Rosa, Alan Sprague, Thamar Solorio and Gracy Warner: Using String Information for Malware Family Identification. @Springer International Publishing Switzerland 2014, A.L.C.Bazzan and K.Pichara(Eds.): IBERAMIA 2014, LNAI 8864,pp.686- 697, 2014.DOI:10.1007/978-3-319-12027-0_55.

[13] Quinlan, J. Ross. “Combining Instance-Based and Model-Based Learning.” ICML (1993).

[14] S. Attaluri, S. McGhee, M. Stamp, Profile hidden markov models andmetamorphic virus detection, Journal in Computer Virology 5 (2) (2009)151–169,

[15] T. Gonzalez: Clustering to minimize the maximum inter cluster distance. Theoretical Computer Science 38, pages 293–306, 1985.

[16] Z. Chen, M. Roussopoulos, Z. Liang, Y. Zhang, Z. Chen, A. Delis, Malwarecharacteristics and threats on the internet ecosystem, Journal of Systemsand Software 85 (7) (2012) 1650–1672.