Efficient Incremental Instance-based Learning Algorithms for Open World Malware Classification

Kien Hoang Dang
University of Engineering and Technology
Vietnam National University, Hanoi

Dai Tho Nguyen *
University of Engineering and Technology
Vietnam National University, Hanoi
Email: nguyendaitho@vnu.edu.vn

Thu Trang Nguyen Thi
University of Engineering and Technology
Vietnam National University, Hanoi

***Abstract*—Malware is growing rapidly in number and become more and more sophisticated. To prevent them we always need to collect more malware samples and update them to the classifier or detection. In this paper, we will propose a method to update new labeled samples of malware to the classifier easily without re-train everything. The classifier can be updated both labeled malware of an existed class or a new class. Our method also has the ability to detect malware of unknown families. Experiments are performed over the traditional computer malware dataset and the IoT malware dataset. The results have shown that our method can reach the macro F1 score almost the same re-train everything but take significantly less time.**

***Index Terms*—IoT malware, Computer malware, n-gram system call, Open world classification**

I. Introduction

Malware is a critical threat to the security of both computers and IoT devices. The massive number of malware samples require us to have automatic methods to analyze efficiently and machine learning is one of them. Machine learning has been used widely in malware analysis and brought a lot of effects. An important task of machine learning is supervised learning which includes classification. In the classification task, we will learn from a given dataset of samples, each sample has features and its discrete label. After the learning process, we can assign a label for samples outside the dataset. The effectiveness of the classifier depends on the goodness of the training set. Over time, the training dataset needs to be replenished continuously to help the classifier enhance the performance (e.g detect malware of a family that was detected recently). As demonstrated in [16], malware samples from the same family can perform significantly different behavior over time because the attacker always wants to evade detection or change malware functions. So we not only need to update malware of a new family but also malware samples of existed families. The problem is how to update the classifier when the training set is replenished with minimal effort.

Instance-based or memory-based learning has been used widely in researches and kNN is one of the most popular algorithms of that class. It provides the ability to update the classifier easily when we have additional data. We just need to add them to the training set and the classifier will be updated automatically. The main problem of kNN is the high storage requirements in order to contain the training data and make classification decisions when a data sample arrives [1] and prototype selection is a solution to solve that problem. Select prototypes from a set help us to reduce the volume of data that needs to be stored in the memory and have been interested in many research like [4], [5], [6], [7]. In prototype selection, we will select a smaller set that represents the whole dataset called prototypes. In the malware analysis field, this method was also used by Rieck et al. in [3] to compress the training set and used for kNN classification. In the training phase, prototypes were selected from the training set by the algorithm that was introduced by Gonzalez in [2]. When predicting a test sample, the classification decision will be given based on only the prototypes, not the whole training set. Moreover, malware samples of unknown families can be detected by comparing the Euclidean distance to the nearest prototype with a threshold.

When we use the prototypes to represent the training set and use them for kNN, the problem is how to update the prototype list when new labeled data samples incoming. The easiest way is re-selecting prototypes from the union dataset of both old and new samples. However, it can take a long time to select prototypes when the dataset is big and moreover the update work may need to be performed many times.

In this paper, we will introduce a method to update the prototype list when a new dataset incoming without re-extract from the union dataset of both the old and new incoming datasets. From the prototype list that was extracted before we will consider the new dataset to decide to add prototypes to the list or not in two ways: batch processing and continuous processing. We will compare our proposed method with the re-extract from everything method in terms of time required, compression ratio, and classification performance when using kNN with k=1. Experiments were performed in both the computer malware dataset in [3] and the IoT malware dataset [8]. The results show that our method can handle the new dataset with significantly less time required and reach the same classification performance and compression ratio.

The rest of our paper is organized as follows. Section 2 includes relating works. In section 3 we will overview the feature extracting algorithm, the prototype extracting algorithm that was proposed by Gonzalez, and the classification

* Coresponding author

algorithm based on prototypes. Section 4 is our contribution to update classifiers when having additional samples arrive at the training set. Section 5 is the overview of the computer malware dataset and IoT malware dataset which we use to evaluate, measures to evaluate the performance, and experiment results.

II. Related Work

In recent years, open-world classification has been interested in many works like [9], [10], [11] to solve the disadvantage of the closed-world assumption of the classic classification task. In the classic classification task, the learning algorithm takes a fixed training data of many classes to build a classifier and assumes all classes in the testing set have existed in the training set. Unfortunately, in real life, an unknown class can appear anytime. The open-world approach assumes that samples of an unknown class can appear in the testing set. The classifiers need to detect and learn knowledge from data of that class. Fei et al. [9] presented 2 challenges to the classifier. First, the classifier must have the ability to detect samples of unknown classes. Second, the system needs to be able to selectively update its models whenever a new class of data arrives without re-training the whole system using the entire past and present training data. Our method can update the classifier easily when a new dataset incoming with a small amount of effort and has the ability to detect samples of unknown malware families by using Euclidean distance.

There are two main ways to update data to the classifier. In the first way, data samples of known classes can be updated to enhance the classifier's performance for known classes. That update method can be reviewed in [17], [18]. In the second way, data samples of unknown classes are updated to the classifier and after that, the classifier can classify data of that class. That method has been researched in [8], [9]. Our method can update the classifier with data samples of both known classes and unknown classes by adding prototypes of the new dataset to the old prototype list. There are some methods that also have the ability to update data samples of both known classes and unknown classes like [19] and [20] but they do not detect data samples of unknown classes.

III. Background

A. Feature extraction using systemcall and n-gram

Extracting the feature vector of malware using n-gram of systemcall has been used in many researches like [3], [8], [15]. In this paper, we will extract features that represent for malware samples by using n-gram of the system call sequence with $n = 2$. A 2-gram of the systemcall is a pair of system-call that appear side by side in the system call sequence. For example, the system call sequence is (*open read read close*) then (*open read*), (*read read*) and (*read close*) are all 2-grams of system call that appear in the system call sequence. Suppose $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ is the set of all possible 2-gram of system calls ($n = |\mathcal{S}|$), x is the systemcall sequence that was created when run malware in the sandbox environment and x will be represented by a point in a n-dimensional vector space. Suppose v is a point in n-dimensional vector space and the value of i^{th} dimension is defined as follow:

$$v_i = \begin{cases} 1 & \text{if } x \text{ contain the 2-gram } s_i \\ 0 & \text{otherwise} \end{cases}$$

The coordinate of the point that represent for x in n-dimensions vector space is $\frac{v}{||v||}$. In that way, we always have the maximum Euclidean distance between two points greater or equal to 0 and less than or equal $\sqrt{2}$. This features extraction method has brought efficiency with both IoT and computer malware in [3] and [8].

B. Extract prototype algorithm

Extracting prototypes of a dataset will give us a smaller set that represents the whole data. In this paper, we will use the method that was introduced by [2] which based on the clustering approach. That method will create clusters in the dataset and each cluster will have a header, each header is considered as a prototype and the set of all headers will represent the whole dataset.

We will keep a list of the distance of all data samples to their closest prototype. In this paper, we use the Euclidean distance which was used by Rieck et al. [3] to select prototypes from a malware set. In the first, we have no prototype and we will perform a loop to choose prototypes one-by-one from the dataset. The first prototype will be chosen randomly from the dataset with the same opportunity for all samples. Each time a prototype is created, the distance of all data sample to it closest prototype will be updated. The data sample that has the farthest distance to its closest prototype will be considered as a prototype candidate. If, the distance of the candidate to its closest prototype greater than a threshold d_p then the candidate will be added to the prototype list, if not the algorithm will stop.

C. Classification using prototypes

After extract prototypes from the training set, we can use the prototype list to represent all data samples. We can classify data samples based on the nearest prototypes using the 1NN rule. When a data sample incoming, we will find the closest prototype and assign a label to the sample based on it. To detect samples of unknown classes, we use a threshold d_r as used by Rieck et al. at [3] to reject a sample that far from all prototypes of the training set. Let x is the data sample that we need to classify, z is the closest prototype of x in the training set. The label of x will be decided as follow:

$$Label(x) = \begin{cases} label\ of\ z & \text{if } d(x,z) \leq d_r \\ unknown & \text{otherwise} \end{cases}$$

In which, $d(x, z)$ is the Euclidean distance between x and z.

IV. Update prototypes algorithms

In this section, we will introduce two algorithms to update the prototype list when a new dataset incoming and describe how they differ. Both 2 algorithms inherit prototypes of the

previous dataset and only update to the list of the prototype without re-extract from both the old and new datasets. The Algorithm 1 handles all dataset in one run time and the Algorithm 2 handles the new dataset sample-by-sample.

input : $old\_prototypes$: The prototype list of the old dataset
d_p: The distance threshold between a data point to the nearest prototype
$new\_dataset$: The new dataset of malware that needs to update
output: $new\_prototypes$: The prototype list represents both the old and new dataset

$new\_prototypes \leftarrow old\_prototypes$;
$distance[x] = \infty$ for all $x \in new\_dataset$;
for $x \in new\_dataset$ **do**
  for $z \in old\_prototypes$ **do**
    if $distance[x] > ||x - z||$ **then**
      $distance[x] = ||x - z||$;
    end
  end
end
while *True* **do**
  $max\_distance$ = max($distance$);
  if $max\_distance > d_p$ **then**
    break;
  end
  choose $z \in new\_dataset$ that $distance[z] = max\_dinstance$;
  $new\_prototypes \leftarrow new\_prototypes \cup \{z\}$;
  $distance[z] = 0.0$;
  for $x \in new\_dataset$ *and* $x \neq z$ **do**
    if $distance[x] > ||x - z||$ **then**
      $distance[x] = ||x - z||$;
    end
  end
end

Algorithm 1: Batch update prototype algorithm

With the Algorithm 1, when a new dataset incoming, we will assign $distance$ is the array that holds the distance of all data in the new dataset to the nearest prototype that was existed. If all elements of $distance$ less than or equal threshold d_p then we don't need to extract any prototype more. If not, we will choose the data that has the farthest distance to its nearest prototype to become a new prototype like the original algorithm. With a new dataset with m samples and the number of prototypes that existed is n. We will need a time complexity $O(n * m)$ to calculate the distance of m samples to its closest prototype. After that, each time create a prototype we must check to update all $prototypes$ list. We can generate maximum m prototypes in case all data samples are far from each other and far from all existed prototypes. So, The time complexity of the algorithm is $O(m^2 + n * m)$.

In the Algorithm 2, one data sample will be handle at one time. We also calculate the distance to the closest prototype. If

H **input :** $old\_prototypes$: The prototype list of the old dataset
d_p: Maximum distance between a data point to the nearest prototype
new_data: The new data point represent for a malware sample that needs to update
output: $new\_prototypes$: The prototype list represents both the old and new dataset

$new\_prototypes \leftarrow old\_prototypes$;
$distance = \infty$;
for $z \in old\_prototypes$ **do**
  if $distance > ||new\_data - z||$ **then**
    $distance = ||new\_data - z||$;
  end
end
if $distance > d_p$ **then**
  $new\_prototypes \leftarrow new\_prototypes \cup \{z\}$
end

Algorithm 2: Continuous update prototype algorithm

the distance to the closest prototype less than d_p then no more prototype is created. If not, it will become a new prototype and will be added to the current prototype list. With each data, we need to find the distance to all existed prototypes so the complex of the algorithm is $O(n)$ with n is the number of existed prototypes.

In the Algorithm 2 the creation of a prototype was decided when a data sample incoming so the order of data incoming will affect the final prototype list. In the Algorithm 1, all sample in the new dataset was considered one time and choose new prototype candidate based on the farthest distance to the current prototypes so the order of sample in the dataset is not mattered. When we run Algorithm 1 to update the list of the prototypes, there also have some differences compare with using the re-extract method in the union dataset from both the old and new dataset using the original algorithm. In the first case, the first prototypes are only created in the old data set, the creation of prototypes in the new dataset is depended on the coverage of prototypes of the old dataset. In the second case, the first prototype can be created in the old dataset or new dataset and decide the creation of prototypes overall the union dataset. However, in both two cases when we have run complete, the distance between each data to its nearest prototype is always less than d_p so a prototype will not represent data samples that too far from it and the distance of two prototypes always greater than d_p help us make sure the number of prototypes isn't too much.

V. Experiment

A. Datasets

To evaluate the efficiency of our proposed method we will perform experiments in the traditional computer malware dataset and IoT malware dataset. Both two datasets include system call sequences when running malware samples in the sandbox environment. The computer malware dataset was

taken from [3] with more than 3000 samples belong to 24 malware families. The IoT malware dataset was introduced in our previous work at [8] with nearly 1000 samples belong to 6 IoT malware families include Gafgyt (BASHLITE), Mirai, MrBlack, Downloader-Mirai, Tsunami (Kaiten), Hajime. All datasets include the system-call sequence when running malware samples in the sandbox environment.

B. Experimental Settings and Metrics

To evaluate the ability to detect unknown malware families, we will hold some families for only the testing purpose and do not provide any sample in the training set. For example, the number of classes for of training set will be chosen m% of all classes and the value of m is changed in many values. For the computer malware dataset, we will choose m = 25, 50, 75, 100 as performed by Shu et al. in [10]. For the IoT malware dataset, because the number of classes is very small(6 classes) so we will use the value of m bigger, we will choose m= 3/6*100%, 4/6*100%, 5/6*100%, 6/6*100% corresponding to drop 3,2,1,0 classes from the training set. When using 100% class for training that means all classes are seen and it is the same as tradition classification. The data will be split randomly to train and test 30 times and average the result. The value of dp and dr is inherited from [3] for the computer malware dataset and chosen by the method at [3] for the IoT malware dataset. To compare the classification performance between re-extract prototypes from all data and only update the new prototypes we will use F1-score with the macro average as used in [10], [9]. The F1 score with macro average is defined as follow:

$$PR_i = \frac{TP_i}{TP_i + FP_i}$$

$$RC_i = \frac{TP_i}{TP_i + FN_i}$$

$$F_i = \frac{2 * PR_i * RC_i}{PR_i + RC_i}$$

$$macro\ F_1 - score = \frac{\sum_{i=1}^{n} F_i}{n}$$

In which:
n: The number of class
TP_i: The number of samples belong to the i^{th} class and was labeled as i^{th} class
FP_i: The number of samples do not belong to the i^{th} class and was labeled as i^{th} class
FN_i: The number of samples belong to the i^{th} class and was labeled as j^{th} class ($j \neq i$)

In our experiment, The F1 score will be calculated over $m + 1$ classes, with m is the number of known classes in the training set and 1 class represent all unknown families. That means, when evaluating the performance we will consider all unknown families in the testing set is belong to the same class called "unknown". In the case use 100% of classes for training, that like the traditional closed-world classification task, the F1 score is calculated over m classes.

To evaluate the performance of the classifier when update samples of a new malware class, we will set up the experiment as the script which was introduced by Fei et al. [9]. Our method has the ability to update a new class to the classifier so we first will build the classifier from only 2 classes by extracting prototypes of that classes. We cumulative update other classes from the training set to the classifier by updating prototypes as Algorithm 1 and Algorithm 2. Note that Algorithm 2 will handle each sample of the new arrive class while Algorithm 1 handle all the samples at one time. With the original method that use re-extract strategy, we will extract prototypes from all data of the training set instead of simulating all the process (start with 2 classes and re-extract prototypes from all data when a new class arrives). That means we will compare the performance of methods when the last class arrives as performed in [9].

To evaluate the performance of the classifier when update samples of a new malware class, we will split the training set into 3 batches. First our algorithm will select prototypes from the first batch after that update prototypes from the second batch and third batch. With the re-extract strategy, we will extract prototype from all data samples when a new batch arrives.

C. Experimental Results

1) Classification Results: Update malware samples of new families

In this experiment we will compare the performance of two update methods and the re-extract strategy when update data samples of known families. The macro F1-score over 2 datasets are shown in Figure 1a and Figure 1b.

We can see that both two update methods have the result almost as same as performing re-extract prototypes from all data. Note that the first prototype was chosen randomly so sometimes the two update methods reach the higher macro F1-score than the re-extract strategy, however, re-extract prototypes usually bring a little higher performance than two update methods.

2) Classification Results: Update malware samples of known malware families

We will perform updated data samples for known malware families based on the setting we have introduced above. The results are shown in TABLE I and TABLE II. Based on the results we can see that updating samples of known classes will help us enhance the classifier's performance for all methods. In all cases with the same training set and the number of known classes, the three methods have almost the same macro F1-score. The continuous update method in the IoT malware dataset can reach a little high performance than the two other methods.

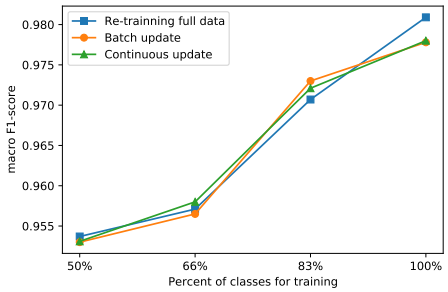
3) Compression ratio comparison: As we mentioned, one of the main challenges of kNN algorithm is the memory required. Extract prototypes from a set will help reduce the memory required by selecting a small set that represents the whole training set and so that speeds up the process to find the nearest data point. Compression ratio is defined as the ratio

	Training data	Re-extract prototype	Batch update	Continuous update
100% classes	Batch 1	**0.9818**	**0.9818**	**0.9818**
	Batch 1+2	0.9888	0.9887	**0.9895**
	Batch 1+2+3	**0.9905**	0.9895	0.9895
75% classes	Batch 1	**0.9189**	**0.9189**	**0.9189**
	Batch 1+2	0.9330	0.9350	**0.9409**
	Batch 1+2+3	**0.9466**	0.9409	0.9409
50% classes	Batch 1	**0.8771**	**0.8771**	**0.8771**
	Batch 1+2	0.9147	**0.9233**	0.9203
	Batch 1+2+3	0.9261	**0.9346**	0.9328
25% classes	Batch 1	**0.9118**	**0.9118**	**0.9118**
	Batch 1+2	**0.9308**	0.9253	0.9293
	Batch 1+2+3	0.9285	**0.9293**	**0.9293**

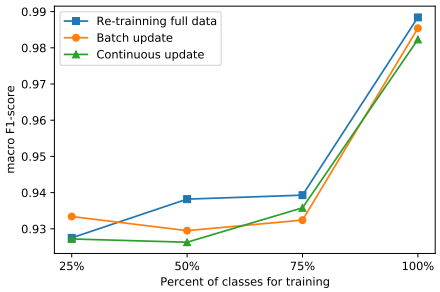
TABLE I: Update data of known families for the computer malware classifier

	Training data	Re-extract prototype	Batch update	Continuous update
100% classes	Batch 1	**0.9818**	**0.9818**	**0.9818**
	Batch 1+2	0.9773	0.9787	**0.9814**
	Batch 1+2+3	0.9791	**0.9814**	**0.9814**
83% classes	Batch 1	**0.9208**	**0.9208**	**0.9208**
	Batch 1+2	0.9658	0.9671	**0.9741**
	Batch 1+2+3	0.9723	**0.9741**	**0.9741**
66% classes	Batch 1	**0.9239**	**0.9239**	**0.9239**
	Batch 1+2	0.9442	0.9457	**0.9461**
	Batch 1+2+3	0.9452	**0.9461**	**0.9461**
50% classes	Batch 1	**0.9257**	**0.9257**	**0.9257**
	Batch 1+2	0.9445	0.9446	**0.9476**
	Batch 1+2+3	0.9472	**0.9476**	**0.9476**

TABLE II: Update data of known families for the IoT malware classifier



(a) IoT malware



(b) Computer malware

Fig. 1: Macro F1-score over the IoT malware dataset and the computer malware dataset

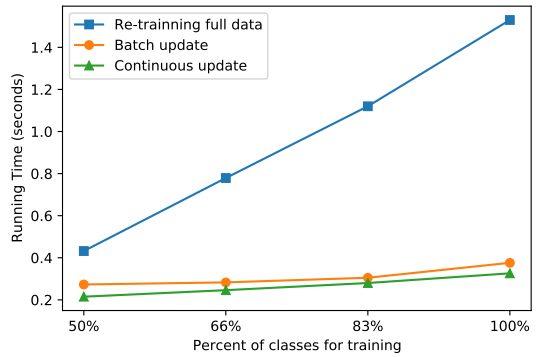
	Re-extract prototype	Batch update	Continuous update
IoT malware dataset	15.5%	15.6%	**15.3%**
Computer malware dataset	4.5%	4.5%	**4.4%**

TABLE III: Compresstion ratio in the IoT malware dataset and the computer malware dataset

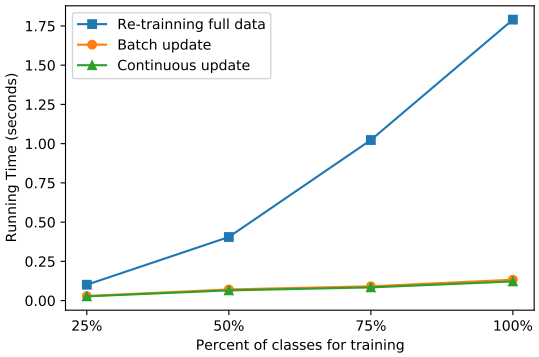
between the number of prototypes and the number of data in the training set. We will compare the compression ratio of our method with the original method which re-extract from all data which include both old data samples and new incoming data samples. The results are averaged from both update unknown and know families .The results are shown in TABLE III.

4) Running Time Comparison: We also interested in the run time of our update method and the original algorithm. We will calculate run time of all method when update data samples. With the update data of new families case, we will compare run time of all method when the last class arrive as performed in [10]. The results can be show in Figure 2a and Figure 2b. With the update data of new families case, we calculate the time to update the last batch (the 3rd batch in our experiment). The results can be show in Figure 3a and Figure 3b. The compression ratio of both method is almost the same and the continuous update method can reach a little good than the two other methods.

Based on the results, we can see that the batch update

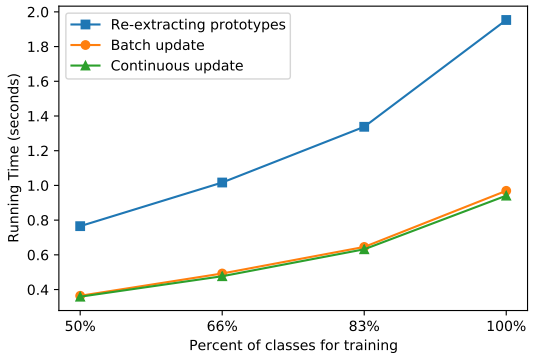


(a) IoT malware dataset

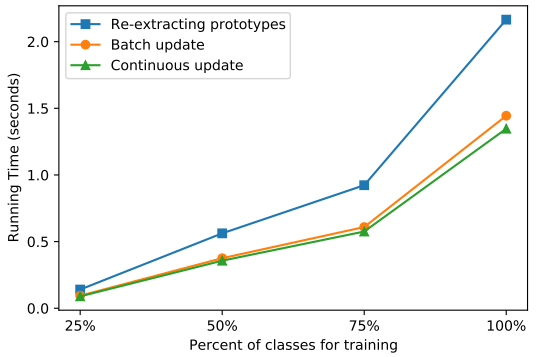


(b) Computer malware dataset

Fig. 2: Running time summary for the last class



(a) IoT malware dataset



(b) Computer malware dataset

Fig. 3: Running time summary for the last batch

method and continuous update method take a lower time than re-extract prototypes from all data when a class or a batch arrives. The reason is very simple, with the batch update and continuous update method we just need to handle samples of the new class or the new batch when it arrives instead of re-handle from the beginning.

VI. Conclusion and Future Work

In this paper we have introduced our method to update prototype list when a new dataset incoming in two ways: Batch processing and continuous processing.

Acknowledgement

Kien Hoang Dang 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.25

References

[1] Garcia, S., Derrac, J., Cano, J. R., & Herrera, F. (2012). *Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study*. In:*IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3), 417–435.

[2] Gonzalez, T., *Clustering to minimize the maximum intercluster distance*. In: *Theoretical Computer Science* , 38 , pages 293–306, 1985.

[3] Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). *Automatic analysis of malware behavior using machine learning*. In: *Journal of Computer Security*, 19(4), 639–668.

[4] Calvo-Zaragoza, J., Valero-Mas, J. J., & Rico-Juan, J. R. (2015). Improving kNN multi-label classification in Prototype Selection scenarios using class proposals. Pattern Recognition, 48(5), 1608–1622.

[5] Paredes R. & Vidal E. (2006), *Learning Prototypes and Distances: A Prototype Reduction Technique Based on Nearest Neighbor Error Minimization*, In :*Pattern Recognition*, vol. 39, no. 2, pp. 180-188, 2006.

[6] Kim S. W. & Oommen B. J. (2007). *On Using Prototype Reduction Schemes to Optimize Dissimilarity-Based Classification*, In: *Pattern Recognition*, vol. 40, no. 11, pp. 2946-2957, 2007.

[7] García-Pedrajas, N., Romero del Castillo, J. A., & Ortiz-Boyer, D. (2009). A cooperative coevolutionary algorithm for instance selection for instance-based learning. Machine Learning, 78(3), 381–420.

[8] Hoang, D. K., Tho Nguyen, D., & Vu, D. L. (2020). *IoT Malware Classification Based on System Calls*. In: *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*.

[9] Fei, G., Wang, S., & Liu, B. (2016). *Learning Cumulatively to Become More Knowledgeable*. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*

[10] Shu, L., Xu, H & Liu, B. (2017). *DOC: Deep Open Classification of Text Documents*. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2911–2916.

[11] Shu, L., Xu, H., and Liu, B., *Unseen Class Discovery in Open-world Classification*, 2018.

[12] Fink M., Shalev-Shwartz S., Singer Y., and Ullman S. (2006). *Online multiclass learning by interclass hypothesis sharing*. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 313–320, 2006.

[13] Kakade S. M., Shalev-Shwartz S.& Tewari A. (2008). *Efficient bandit*

algorithms for online multiclass prediction. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 440–447, 2008.

[14] Zhao P., Hoi S. C. H., & R. Jin (2011). In: *Double updating online learning. Journal of Machine Learning Research*, 12:1587–1615, 2011.

[15] Canzanese, R., Mancoridis, S., & Kam, M. (2015). *System Call-Based Detection of Malicious Processes*. In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. pages 119-124, 2015.

[16] Wadkar, M., Di Troia, F., & Stamp, M. (2019). *Detecting Malware Evolution Using Support Vector Machines*. In: *Expert Systems with Applications*

[17] Kakade, S. M., Shalev-Shwartz, S., & Tewari, A. (2008). *Efficient bandit algorithms for online multiclass prediction*. In : *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 440–447, 2008

[18] Zhao P., Hoi S. C. H., & Jin R. (2011) . *Double updating online learning*. In: *Journal of Machine Learning Research*, 12:1587–1615, 2011.

[19] Feng, L., Zhao, C., Chen, C. L. P., Li, Y., Zhou, M., Qiao, H., & Fu, C. (2020). *BNGBS: An efficient network boosting system with triple incremental learning capabilities for more nodes, samples, and classes*. In *Neurocomputing*, 412, 486–501.

[20] Chen, C., Min, W., Li, X., & Jiang, S. (2019). Hybrid incremental learning of new data and new classes for hand-held object recognition. In :Journal of Visual Communication and Image Representation