applied sciences

Article

Identifying the Author Group of Malwares through Graph Embedding and Human-in-the-Loop Classification

Dong-Kyu Chae , Sung-Jun Park, Eujeanne Kim, Jiwon Hong and Sang-Wook Kim *

Department of Computer and Software, Hanyang University, Seoul 04763, Korea;
dongkyu@hanyang.ac.kr (D.-K.C.); sjpark@agape.hanyang.ac.kr (S.-J.P.); eujeanne@agape.hanyang.ac.kr (E.K.);
nowiz@agape.hanyang.ac.kr (J.H.)
* Correspondence: wook@hanyang.ac.kr

Abstract: Malware are developed for various types of malicious attacks, e.g., to gain access to a user's private information or control of the computer system. The identification and classification of malware has been extensively studied in academic societies and many companies. Beyond the traditional research areas in this field, including malware detection, malware propagation analysis, and malware family clustering, this paper focuses on identifying the "author group" of a given malware as a means of effective detection and prevention of further malware threats, along with providing evidence for proper legal action. Our framework consists of a malware-feature bipartite graph construction, malware embedding based on DeepWalk, and classification of the target malware based on the k-nearest neighbors (KNN) classification. However, our KNN classifier often faced ambiguous cases, where it should say "I don't know" rather than attempting to predict something with a high risk of misclassification. Therefore, our framework allows human experts to intervene in the process of classification for the final decision. We also developed a graphical user interface that provides the points of ambiguity for helping human experts to effectively determine the author group of the target malware. We demonstrated the effectiveness of our human-in-the-loop classification framework via extensive experiments using real-world malware data.

Keywords: malware author groups; malware detection; classification; graph embedding; human-in-the-loop method

check for updates

Citation: Chae, D.-K.; Park, S.-J.; Kim, E.; Hong, J.; Kim, S.-W. Identifying the Author Group of Malwares through Graph Embedding and Human-in-the-Loop Classification. *Appl. Sci.* **2021**, *11*, 6640. https://doi.org/10.3390/app11146640

Academic Editors: Sławomir Nowaczyk, Mohamed-Rafik Bouguelia and Hadi Fanaee

Received: 19 May 2021
Accepted: 16 July 2021
Published: 20 July 2021

1. Introduction

Computer technology has become essential to the public more than ever. As the importance of computer systems increases, attempts to attack the system are increasing accordingly. Malwares are used for such attacks to gain access to the user's private information or to control the computer system itself [1,2]. Such heisted private information often leads to another heist of private information that is used for identity theft or is even sold illegally. In the case of computer system control, the damage becomes more complicated and severe. With the hijack, the attacker not only gains easy access to every private information stored and linked with the system but can also use the system for another hijacking. It can even be used to organize an attack that might cause serious damage to corporations or governments.

Various studies, such as malware detection, malware propagation analysis, or malware family analysis, have been conducted to prevent attacks. Malware detection [3,4] identifies whether a target program is malware or not. Malware propagation analysis [5] tracks the propagation process to prevent further spreading of the malware. Malware family clustering [6,7] and classification [8,9] groups the malware family as a cluster and extracts shared features within the family. However, we found that there were only a few attempts to identify the author groups of malware. These author groups create and release malware to attack requested governments or institutions from certain countries [10]. The source code of the malware is suspected to be shared effortlessly among malware authors within the same group. Such easy access to the source code enables malware authors at any level to duplicate, modify, or even combine it with another malware and release the derivative version rapidly.

To help security experts to efficiently cope with the malware attacks which may have similar behaviors from the previous attacks sent by certain author group, this study focuses on classifying the author group of a target malware. It may be easier to detect or predict the attack strategy in the future if the original source code of the author group of the target malware can be correctly defined. Therefore, we propose a graph-based and human-machine collaborative framework for identifying author groups effectively. In summary, our framework is composed of the following three tasks: (1) construction of a malware-feature graph, (2) embedding each malware, and (3) classification based on the k-nearest neighbor (KNN) approach aided by human experts. To construct the malware-feature graph, we carefully defined some distinctive features that are expected to be highly related to the shared characteristics of malware codes within an author group. Thus, we expect that the features extracted from the malware can be the key signatures that represent the author group. Then, with the extracted features, we construct a malware-feature graph, which is a bipartite graph comprising a malware part and a feature part, each of which is connected to its extracted feature nodes with edges. We then trained a graph-embedding model to project malware to a shared latent space. Finally, for the target malware, we selected its nearest malware in the latent space, and then classified the target malware's author group as the chosen neighbor's author group.

However, in this classification approach, there are some ambiguous cases that may be incorrectly classified. For example, there would be multiple malware from different author groups that are almost equidistant from the target malware, rather than just one malware that is apparently closest to the target malware. In addition, there would be no malware sufficiently close to be considered as a neighbor of the target malware. To handle such exceptions, we developed a "human-intervenable classification framework". Here, the standards of ambiguity must be defined. We propose two metrics measuring such ambiguity: the inter- and intra-class closeness.

The inter-class closeness evaluates whether the distances from each author group cluster to the target malware are considerably far, meaning that none of the author group clusters are sufficiently close to the target malware to be classified as one of them. For intra-class closeness, if nearby author groups have approximately the same distance from the target malware, the system is unable to decide to which author group a target malware should be assigned to with certainty. If both cases are applicable to the target malware, the system considers the malware to be ambiguous and postpones the classification to human experts. We also developed a graphical user interface that provides the points of ambiguity for helping human experts to effectively determine the author group of the target malware (i.e., ambiguous case).

We evaluated our approach through extensive experiments using a real-world dataset labeled by a group of domain experts. The results demonstrated that our proposed metrics, inter-class closeness, and intra-class closeness were effective in avoiding the risk of misclassification. The accuracy of the inter-class inliers and outliers showed significant differences, with accuracies of 96.6% and 76.0%, respectively. For intra-class closeness, we were able to find the optimal hyperparameter value that balances the accuracy and human engagement rate. After the engagement of human experts, we were able to increase the accuracy of the machine-only classification of 93.7% to 95.7% with a much smaller number of samples turned over for manual inspection.

The remainder of this paper is organized as follows: Section 2 introduces the dataset and details of its extracted features for this research and explains how to construct the malware-feature graph. With the graph constructed, Section 3 describes the graph embedding for the KNN-based classification and the mechanism of human expert intervention. Section 4 presents the experiments and evaluates the performance of the proposed framework. Section 5 presents the conclusions of this study.

2. Related Work

This section summarizes four research directions to fight against malware, including (1) malware detection, (2) malware clustering, (3) malware propagation analysis, and (4) malware family classification. To the best of our knowledge, our work is the first study dealing with the problem of author group classification on malware.

Malware detection aims at discovering the presence of malware on a system. It tries to determine whether a given program is malicious or not. The detection is often based on certain distinctive signatures. For example, Kreuk et al. [11] enhanced the effectiveness of malware detection by modification of malware binary files expecting significant changes in functionality which will make the malware distinctive from benign programs. Yan et al. [12] designed convolutional neural network for the malware detection considering the opcodes of the malware as grayscale images to perform the detection. Bhandai et al. [13] assesses semantic traits of malware for the detection to prevent the obfuscated malwares from disguising the malware detectors based on pattern-matching.

Malware clustering aims at gathering them into sets of malware that exhibit similar behavior for helping security experts to easily derive generalized signatures to each malware cluster. Bayer et al. [14] utilized the dynamic analysis to gather execution traces to cluster the malware by the behavioral profiles. Pitolli et al. [6] employed Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm to the malware clustering task and assessed the existing malware family ground truths generated by various vendors. Islam et al. [15] introduced FIRMA, a tool offering solutions for automated malware clustering and signature generation.

With the increased number of devices, the network has also become expanded and complicated. Hence, the spread of malware throughout the diverse connection, such as wireless internet or Bluetooth, has been posing a major concern in internet security. *Malware propagation analysis* aims at analyzing such spreads and prevents further infections. In Reference [16], multiple existing malware detection and analysis methodologies, such as Athdi model, ACT scheme, or SII model, were reviewed focusing on malware propagation through email and social networks. Cheng et al. [5] proposes a novel model that efficiently analyzes the threat of hybrid malware by promptly providing approximate knowledge of the malware's severity and propagation speed.

In general, malware can be categorized into one of well-known malware *families*, such as Ramnit, Vundo, Simda, and so on [9]. The *malware family classification* methods often extract structural information of malware and examines its various attributes to accurately classify the malware into its family class and enables the security experts to properly respond to encountered (unknown) malware. The authors of Reference [17] gathered a variety of malware datasets and provided comprehensive comparisons by categorizing into its family to suggest the direction of the future research of this field. To enhance the performance, Huang et al. [8] proposed MtNet, a classifier using multi-task deep learning architecture for the malware family classification. For the aspect of feature extraction, Ahmadi et al. [18] suggests a feature fusion approach which enables effective concatenation of features leading to optimal point between accuracy and running time.

3. Malware Graph Construction

3.1. Features

We gathered 1941 malware samples and labeled them with corresponding author groups, which were classified by cybersecurity experts using real-world forensic standards. Each sample was classified into five classes. For security purposes, we name each class as A, B, C, D, or E.

Our next step is to define malware features. To select meaningful malware features for author group classification, extracting discriminative signatures from malware is important [15]. In this regard, the binary file or used library of the target malware may include useful static features. For example, features, such as strings or functions, found in binary files may be informative. However, because these derivations are often modified or combined rather than reused line-by-line, the malware is often metamorphic and polymorphic, making such features less significant for the classification. As a countermeasure to such cases, dynamic features gathered while executing the binary file in a constrained environment were used for classification, along with the static features [15].

As a result, our static features extracted from the analysis of the static binary include:

- Function: We define several operation codes from the gathered functions while statically analyzing binary files as static features. If the derivations reuse the exact code previously observed, the functions will be useful for classifying the malware as the same group [19]. Because using the whole plain text of the function is inefficient in terms of memory usage, we convert each function into a shortened fixed length value using a cryptographic hash function;
- Basic Block: We define the basic blocks of the operation codes from functions as static features. In the operation codes, each basic block is divided by a jump command. To prevent subdivision of the functions that occurred owing to such minor changes, we used the basic block as a feature, as well;
- Strings: We defined strings found in binary files as static features. Most strings are often meaningless for malware analysis. However, because of the possibility that the malware author's habitual human behavior is reflected in the codes, strings can also be informative for the author group classification purpose [20];
- Imports and Exports: Some malware may also involve other files. During this process, the malware needs interfaces to call libraries and public methods. We define the processes as static features because they contain the names of the methods imported or exported. Because the names of the libraries are often consistent, this static feature is relatively effective against modification of the derivations [21].

Next, our dynamic features, which were extracted while executing the binary file in a virtual machine, include the following:

- Mutexes: Mutexes are used as locking mechanisms for memory location. If the author groups or the original of the modified malware is the same, the names of the mutants are likely to be reused, as well [22]. Therefore, we defined mutexes as dynamic features.
- Networks: Each malware with a purpose will likely send some sort of report to the author or designated location. Locations, such as DNS, URLs, or IP addresses, are likely to be reused if the malware is in the same author group [23].
- Files: During the execution of the malware, certain patterns of read or write can be observed in the filesystem. Some patterns mimic benign programs patterns, and some may occur while processing the attack. Either case often occurs in the filesystem, and if the malware were developed by the same author group, the locations would likely be shared.
- Keys: There is a location called a registry where Microsoft Windows stores settings for the OS and other program files. As a means of malware attack, the authors often target the settings in this registry for certain purposes. Therefore, we defined the registry keys accessed or modified as a dynamic feature.
- Drops: Malware, such as Trojan, is designed to download a drop from the web to disguise detection. As the dropper is undetected, it can keep downloading the drop and keep the malware attack. If the dropper is part of the malware built by the same author group, there is a high possibility of reuse of the drop file, which can be considered as a dynamic feature.

3.2. Graph Construction with Feature Refinement

We built a bipartite graph from malware and its features. Malware and distinct features have their own nodes. Each malware node in the malware part is connected to its extracted feature nodes in the feature part with edges.

Before applying graph embedding methods to this graph, we first refine some feature nodes from the graph to reduce their size. Among the approximately 524 K feature nodes, we note that some feature nodes are found unusable and only increase the computation overhead, which would make the prediction inaccurate and slow. More specifically, features with a low frequency are often meaningless and insignificant for the representation of an author group; however, some might be a detail for single malware samples. These features are a limited relationship between genetically nearby samples rather than a flow of malware mutations. By removing such meaningless feature nodes, the classification time can be significantly decreased with an improvement in the accuracy.

Unlike low frequency features, there are also some feature nodes that are commonly found throughout multiple author groups. These commonly found features may contribute little to the author group identification. Here, we used the well-known information gain (IG) and entropy (E) to evaluate the contribution of such features for classification [24]. These were calculated as follows:

$$IG_F = E(S) - E_F(S), \tag{1}$$

$$E(S) = -\sum_{i \in l} P_{i,S} \log P_{i,S}, \tag{2}$$

$$P_{i,S} = \frac{|C_{i,S}|}{|S|}, \tag{3}$$

$$E_F(S) = \sum_{j \in F} \frac{|S_j|}{|S|} \times E(S_j), \tag{4}$$

where S represents the complete set of malware samples, and $C_{(i,S)}$ represent the labeled samples in S with author group class i. Furthermore, l represents a set of all author groups found in the samples, and S_j represents sample S with a feature value j of a feature in question F. Then, IG_F represents the amount of entropy reduction with the feature F as a given. In Equation (1), entropy E is considered a common occurrence throughout the total author groups. This implies that the high entropy is less distinctive for the classification. In other words, features with a high information gain are informative features that should be kept. Both information gain and entropy can be used as metrics to determine the usefulness of a given feature A [24]. Between the two, we used information gain for this research because it provided better results in our evaluation.

4. Graph-Based Classification with Human Engagement

After building the malware-feature bipartite graph and removing the feature nodes with only one frequency and a low IG, new malware nodes (i.e., test data) must be added to the graph to be classified. Our next steps toward classifying the test malware nodes are (1) embedding all malware nodes and (2) performing KNN-based classification with hu-man expert intervention.

4.1. Graph Embedding

Graph embedding converts nodes to latent vectors on the latent feature space according to its graph topology. Thereby, relative distances between the nodes in the graph are also reflected to the latent space. We employed two well-known graph embedding methods: DeepWalk [25] and large-scale information network embedding (LINE) [26]. DeepWalk performs multiple random walks, say K, from each node to collect a set of K paths with sequences of nodes that has a predefined length (say L). Then, it learns the embeddings of the nodes by using the collected random walks as "context" data, based on the assumption that adjacent nodes are likely to have similar paths with similar embeddings. The objective function is:

$$\max(\log Pr(\{v_{i-w}, \ldots v_{i+w}\} \backslash v_i | \Phi(v_i)), \tag{5}$$

where $Pr(|)$ shows the co-occurrence probability between malware around to target malware v_i within the window size of w. Additionally, Φ is a mapping function that sets the target malware to the latent space, eventually setting topologically similar malware in the shared latent space after the training.

LINE is another embedding method based on edges rather than nodes, unlike DeepWalk. LINE uses two similarities for conversion: first order and second order. The first order proximity (O_1) between two nodes is computed through the weight of the edges linked to each other, while the second order proximity (O_2) uses similar neighbor nodes. LINE is then trained by concatenating these two proximities for each node. Each proximity objective function for edges (i, j) between nodes v_i and v_2 in the set of V nodes can be written as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}, \tag{6}$$

$$p_2(v_j|v_i) = \frac{exp(-\vec{u}'^T_i \cdot \vec{u}_j)}{\sum_{k=1}^{|V|} \exp(-\vec{u}'^T_i \cdot \vec{u}_j)}, \tag{7}$$

$$O_1 = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j), \tag{8}$$

$$O_2 = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_j|v_i), \tag{9}$$

where $\vec{u}_i \in R^z$ is the z-dimensional vector of node v_i. Further, p_1 is the joint probability of node v_i in the space $V \times V$, and p_2 is a conditional distribution of v_i in the entire set of nodes. With w_{ij}, which is the weight of the edge (i, j), the embeddings (i.e., $\{\vec{u}_i\}_{i=1...\{V\}}$) can be obtained by optimizing the objective functions O_1 and O_2.

Both DeepWalk and LINE can be used for embedding malware nodes, and we used DeepWalk for this task because it provided better results in our evaluation. With the embeddings, each malware is now converted to a feature vector, which is a proper format for classification, such as KNN classification. The KNN classifier labels the target sample by finding the majority of the top-k nearest neighbor labels. Our framework borrowed this classifier. In practice, we find the top-1 nearest neighbor for the target malware and then confirm the author group of the chosen neighbor as the target malware's author group. We also tried various k values, such as 2 or 3, but setting k as 1 provided the best results.

4.2. Human-in-the-Loop Classification

In the aforementioned classification process, there can be ambiguity for some test malware samples to be classified. Some may have approximately the same distance from the respective nearest author groups or may not have any close-enough author group at all. For these ambiguous cases, we designed our framework to postpone the classification and let human experts be involved in the final decision, rather than enforcing the final classification while taking the risk of being wrong. We argue that it is much more effective to let human experts make decisions rather than ignore ambiguity and suggest unreliable predictions.

As a standard for each type of uncertainty to deal with, we proposed the inter- and intra-class closeness, which will be explained in the following two subsections.

4.2.1. Inter-Class Closeness

Inter-class closeness is the measurement of the absolute distances among malware to verify whether the target malware is close enough to the nearest neighbor. The measurements were compared according to the distance distribution within the same author group chosen for comparison. Specifically, let v_t and v_1 denote the target malware and the chosen nearest neighbor of v_t, respectively. Let $d_{t,1}$ be the distance between v_t and v_1. Then, $d_{t,1}$ is checked to determine whether it is an outlier or not based on the distance distribution between malware belonging to v_1's author group. If the value of $d_{t,1}$ is outside of the interquartile range (IQR) obtained from the box plot analysis [27], we consider $d_{t,1}$

as the outlier, which means that classifying v_t would be risky because the distances from each author group to v_t are considerably far.

4.2.2. Intra-Class Closeness

Intra-class closeness is a measurement that compares the relative distance from multiple nearest neighbors to the target malware. Let $d_{(t,2)}$ be the distance between the target malware v_t and its second nearest malware, which must belong to a different author group from v_t. Then, the relative distance can be examined by comparing the difference between $d_{t,1}$ and $d_{t,2}$ as follows: If $\frac{d_{t,2}}{d_{t,1}} > \theta$, then $d_{t,1}$ satisfies the intra-class closeness, which means that it would be safe to classify v_t because $d_{t,1}$ is considerably short compared with $d_{t,2}$. Otherwise, it is considered as an ambiguous case. Note that is a tunable hyper-parameter.

To summarize, our framework first examines that whether v_t satisfies the inter-class closeness standard. If it fails, then our framework checks whether it also fails in the second standard, intra-class closeness. If it does not satisfy this either, we let the "machine" postpone the classification, and let human experts determine it.

4.2.3. Intervention of Human Experts

We now explain our human-engaged classification for identifying malware author groups more accurately. If the machine cannot classified it with both inter- and intra-class closeness confirmation, human experts take the role of classification by following two stages. First, they simply refer to visualized box plots, which will be introduced later. Second, if they are not able to judge based on visualization, they manually inspect the given malware in detail. The second step will be a time-consuming job but will provide significantly better accuracy for the ambiguous cases compared to the machine learning-based classification. Hence, our goal is to provide reasonable classification accuracy while controlling the number of malwares to be inspected manually as much as possible, so that only the most necessary malware samples will be inspected by human experts.

Figure 1 shows some examples of visualization that we designed to provide human experts for the first step mentioned above. The x and y axes indicate the author group and the distance distribution among all the malware and their nearest neighbors in the corresponding author group. In the figures, outliers are plotted as black points, and some nearest neighbors are plotted as colored points, along with the distance distribution illustrated with a box-and-whisker plot [27]. With the visual explanation, the human experts were able to recognize the ambiguity that the machine has experienced and efficiently assess the unidentified malware that was postponed.

4.2.4. Implementation Details

For training the DeepWalk model with our malware-feature graph, we set the walk length (L), the number of paths per each node (P), the window size (w), and the dimensionality of the latent space (z) as 80, 10, 10, and 32, respectively. These values have been obtained via an extensive grid search. Stochastic gradient descent with a learning rate of 0.025 was used for model training. Hierarchical softmax [9], which was used in other embedding models, such as word2vec [28], was also employed. We tried several values for θ within $\{1.2, 1.5, 2, 3\}$.

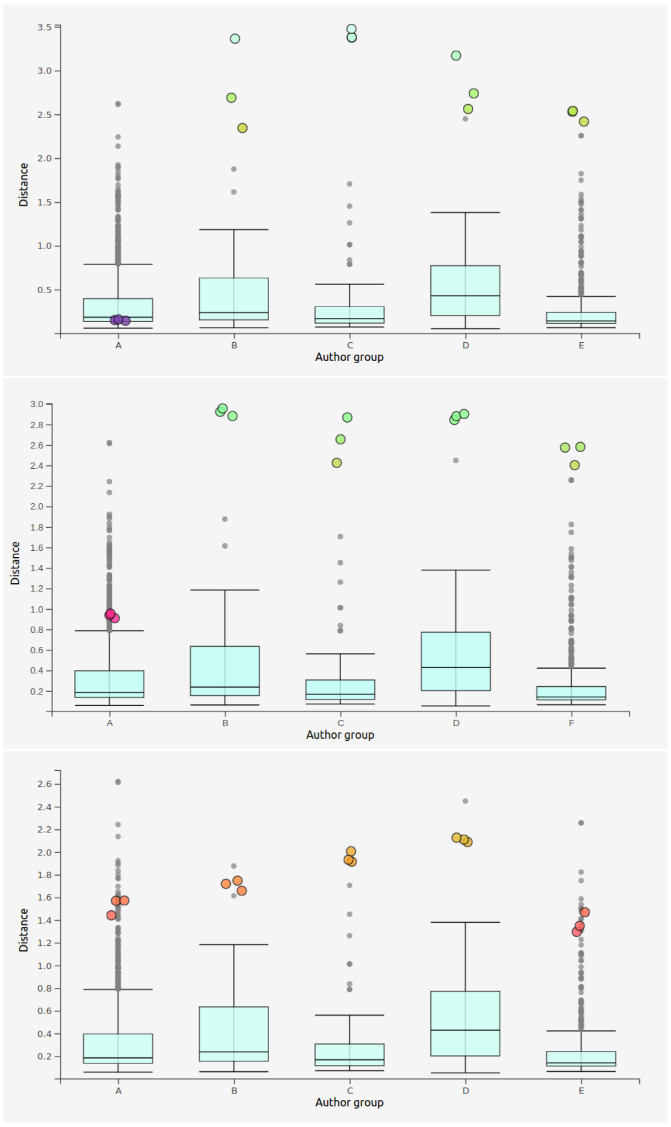

Figure 1. Three examples of data visualization. for human experts.

5. Evaluation

5.1. Experimental Setting

We used the malware dataset introduced in Section 2. For all the experiments, we followed the leave-one-out cross-validation (LOOCV) protocol for accuracy evaluation [29,30]. For each validation, LOOCV uses $N - 1$ subsets as the training set and a single left sample as the test set ($N = 1941$ in our case). In our evaluation, we first constructed a malware-feature graph by using all the training and test malware and embedded them. We then predicted the author group of the test malware and checked whether the prediction was correct. After N validations, the $F1$ score was computed. For the exclusive examination, we applied a macro-average, a micro-average, and a weighted approach to collect $F1$ scores [31]. The macro-average $F1$ score ($F1_M$) is a simple average for each class. To consider the imbalance among the classes, we used the micro-average F1 score ($F1_\mu$), which emphasizes common classes and understates uncommon classes. We also used the weighted F1 score ($F1_w$), which can be calculated as follows:

$$Precision_w = \sum_{i \in l} \frac{tp_i}{tp_i + fp_i} \times W_i, \tag{10}$$

$$Recall_w = \sum_{i \in l} \frac{tn_i}{tn_i + fn_i} \times W_i, \tag{11}$$

$$W_i = \frac{tp_i + fn_i}{tp_i + fn_i + tn_i + fp_i}, \tag{12}$$

$$F1_w = \frac{2 \times Precision_W \times Recall_W}{Precision_W + Recall_W}, \tag{13}$$

where l is a set of all class labels, which are the author groups in this experiment. Furthermore, tp_i, fp_i, tn_i, and fn_i are the number of samples in each class as true positives, false positives, true negatives, and false negatives, respectively. The weight W_i was used to prevent sample imbalances among classes.

5.2. Experimental Results

5.2.1. Effectiveness of Inter-Class Closeness

In this experiment, we evaluated the effectiveness of inter-class closeness as a metric for assessing ambiguity. Therefore, we compared the KNN classifier's accuracy on malware sets satisfying the inter-class closeness (i.e., inliers) and not satisfying them (i.e., outliers).

As shown in Table 1, the classifier was able to classify 1819 out of 1941. Specifically, for each group A, B, C, and D, the classifier was able to classify with an accuracy of 96.4%, 74.1%, 82.4%, 80.6%, and 94.4%, respectively. Groups B, C, and D showed a relatively low accuracy, presumably due to the low number of samples. Importantly, every accuracy with the inliers from each group outperformed those of the outliers by a large portion, which demonstrates the effectiveness of our inter-class closeness metric.

Table 1. Effectiveness of inter-class closeness.

		Total	**# Inliers**	**# Outliers**
Total	# malwares	1941	1670	271
	# correct/incorrect	1819(93.7%)/122	1613(96.6%)/57	206(76.0%)/70
Group A	# malwares	1220	1074	146
	# correct/incorrect	1176(96.4%)/44	1074(98.2%)/19	121(82.9%)/25
Group B	# malwares	85	71	14
	# correct/incorrect	63(74.1%)/22	61(85.9%)/10	2(14.3%)/12
Group C	# malwares	85	69	16
	# correct/incorrect	70(82.4%)/15	61(88.4%)/8	9(56.3%)/7
Group D	# malwares	72	65	7
	# correct/incorrect	58(80.6%)/14	57(87.7%)/8	1(14.3%)/6
Group E	# malwares	479	391	88
	# correct/incorrect	452(94.4%)/27	379(96.9%)/12	73(14.3%)/21
$F1_M$ **(total)**		85.1%	90.0%	51.1%
$F1_\mu$ **(total)**		85.1%	96.6%	76.0%
$F1_w$ **(total)**		85.1%	96.7%	74.8%

We were also able to confirm that postponing the classification of outliers for further inspections is effective. As shown in the accuracy results for the inliers, the machine can perform the classification with a significant accuracy of 96.6% by classifying only with the inliers and postponing the outliers.

5.2.2. Effectiveness of Intra-Class Closeness

Next, we evaluated the effectiveness of the intra-class closeness based on the 271 inter-class outliers determined in the previous experiment. We selected each outlier and examined whether $\frac{d_{t,1}}{d_{t,2}} > \theta$: if so, our KNN classifier classifies this malware; otherwise, it postpones the classification and transfers the malware to human experts. The experiment was conducted under various values for θ.

In Table 2, "Y/correct" indicates the number of malware samples satisfying the intra-class closeness, and the number of correctly classified samples. 'N' indicates the number of samples that do not satisfy the intra-class closeness. We observed that with a higher θ value, the number of samples satisfying the intra-class closeness decreases and the number of samples transferred to human experts increases. Conversely, with a lower θ value, the proportion of the samples that the machine classifies with the KNN classifier will increase, but it may have the risk of a lower accuracy. For example, with $\theta = 3$, there were 45 out of 271 inter-class outliers that were additionally classified with the 1-NN classifier because it satisfied the intra-class closeness, while the remaining 226 samples were transferred to human experts. Of the additional 45 samples, 44 samples were classified correctly with an accuracy of 97.8%. However, with $\theta = 1.2$, only 186 samples satisfied the intra-class closeness, and, among them, only 161 were classified correctly with an accuracy of 86.6%, and the remaining 85 samples were transferred to human experts. To summarize, a higher θ provided higher accuracy in general. However, a higher θ also caused several target malwares not to satisfy the relative closeness, which might result in time and other expenses for the classification transferred to human experts.

Table 2. Effectiveness of intra-class closeness.

	# of	$\theta = 3$		$\theta = 2$		$\theta = 1.5$		$\theta = 1.2$	
	Outliers	**Y/Correct**	**N**	**Y/Correct**	**N**	**Y/Correct**	**N**	**Y/Correct**	**N**
Total	271	45/44	226	93/88	178	141/129	130	186/161	85
Group A	146	11/11	135	49/48	97	84/81	62	105/98	41
Group B	14	1/0	13	3/0	11	5/0	9	6/0	8
Group C	16	2/2	14	2/2	14	4/3	12	8/5	8
Group D	7	0/0	7	0/0	7	0/0	7	3/0	4
Group E	88	31/31	57	39/38	49	48/45	40	64/58	24
$F1_M$	-	70.0%	-	68.3%	-	62.1%	-	47.2%	-
$F1_\mu$	-	97.%	-	94.6%	-	91.5%	-	86.6%	-
$F1_w$	-	96.9%	-	93.2%	-	90.1%	-	84.6%	-

Table 3 summarizes the classification accuracy and proportion of classified samples after applying the two metrics, intra- and inter-class closeness. First, 1819 samples were classified accurately with an accuracy of 93.7% using the KNN classifier. After applying the inter-class closeness, 1613 out of 1670 samples could be classified accurately with an accuracy of 96.6%. This significant accuracy could be achieved by postponing the 271 inter-class outliers. Then, with the intra-class closeness applied to the 271 inter-class outliers, additional samples can be classified with the KNN classifier depending on the value of θ: When θ decreases, we can observe the increasing ratio of classified samples and a slight decline in the overall accuracy.

We found that $\theta = 1.5$ shows the most balance between the accuracy and postponing rate. With this parameter value, we were able to confirm that the classifier showed reasonably high accuracy, along with a low postponing ratio.

Table 3. Effectiveness of inter-close and intra-close closeness. The accuracy and postponing rate are balanced when $\boldsymbol{\theta = 1.5}$.

	k-NN (w/o Give Up)	**Inter-Class**	**Intra-Class ($\theta = 3$)**	**Intra-Class ($\theta = 2$)**	**Intra-Class ($\theta = 1.5$)**	**Intra-Class ($\theta = 1.2$)**
# classified	1941	1670	1715	1763	**1811**	1856
# correct	1819	1613	1657 (1613 + 44)	1701 (1613 + 88)	**1742 (1613 + 129)**	1774 (1613 + 161)
$F1_M$	85.1%	90.0%	89.8%	89.5%	**88.9%**	87.8%
$F1_\mu$	93.7%	96.6%	96.6%	96.5%	**96.2%**	95.6%
$F1_w$	93.7%	96.7%	96.7%	96.5%	**96.2%**	95.6%
# give up	0	271	226	178	**130**	85
% of classified	100%	86.0%	88.4%	90.8%	**93.3%**	95.6%

5.2.3. Effectiveness of Man-Machine Collaboration

In this experiment, for the 130 malware postponed according to inter-class closeness and intra-class closeness with $\theta = 1.5$, two cybersecurity experts examined the sorted-out malware with ambiguity for this experiment. Here, the experts classified the malware through the following two stages: First, they simply referred to the visualized box plots introduced in Section 4.2.3, in order to reduce time and effort. Second, if they were not able to judge based on the visualization results, they manually inspected the given malware in detail. The second step will be a time-consuming job but will provide 100% accuracy of classification.

Table 4 shows the results. For comparison, we included the results of the KNN classification with and without applying inter- and intra-class closeness with $\theta = 1.5$. Two human experts classified (with the visualized interface and without manual inspection) 49 and 73 samples out of postponed 130 samples, respectively. Then, they selected the remaining 81 and 57 samples for manual inspection, respectively. After manual inspection, all the selected postponed samples were classified correctly and were able to classify 1858 and 1846 samples appropriately, with 95.7% and 95.1% accuracy, respectively.

Table 4. Effectiveness of man-machine collaboration.

	k-NN (w/o Give Up)	**Intra-Class ($\theta = 1.5$)**	**Expert 1**	**Expert 2**	**Average**
# classified	1941	1811	1860	1884	1872
# correct	1819	1742	1777	1789	1783
$F1_\mu$	93.7%	96.2%	95.5%	95.0%	95.25%
# give up	0	130	81	57	69
# correct after manual inspection	-	-	1858 (1777 + 81)	1846 (1789 + 57)	1852 (1783 + 69)
$F1_\mu$ after manual inspection	-	-	95.7%	95.1%	95.4%

6. Conclusions

This paper proposes a novel framework for malware author group classification based on human-intervenable classification and graph embedding. We extracted features from malware using both static and dynamic analyses. With the selectively extracted features, we built a malware-feature bipartite graph and performed a KNN classification model based on graph embedding. We also removed the less significant features of efficiency. To make the framework human-intervenable, we suggested intra- and inter-class closeness as metrics for determining the ambiguity of classification. We also developed visualized box plot results for human experts. We conducted experiments using a real-world malware dataset labeled by cybersecurity experts. The results confirmed the effectiveness of intra- and inter-class closeness as metrics and the effectiveness and efficiency of the human-intervenable framework.

In our future work, we plan to additionally use the control flow and the call graph information as our malware features. Since the control flow and the call graph information have been recognized as unique characteristics of a program, we believe incorporating such graph information into our malware-feature bipartite graph will be a promising research direction to more accurate author group classification on malware. We are also planning to try various existing visualization techniques for malware. Based on an excellent survey [32], there have been malware visualization methods which seem to be applicable to our work (e.g., 2D/3D displays or geometrically-transformed displays). We plan to apply them to the graphical user interface for helping our human experts determine the author group of a target malware. We will also try to evaluate which visualization makes our human experts classify given malware the most accurately.

Author Contributions: Conceptualization, S.-J.P. and E.K.; methodology, D.-K.C. and E.K.; software, J.H.; validation, D.-K.C.; formal analysis, S.-J.P.; investigation, D.-K.C.; resources, J.H.; data curation, S.-J.P.; writing—original draft preparation, D.-K.C.; writing—review and editing, D.-K.C. and S.-W.K.; visualization, E.K.; supervision, S.-W.K.; project administration, S.-W.K.; funding acquisition, S.-W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (*MSIT) ((1) No. NRF-2020R1A2B5B03001960 and (2) No. 2018R1A5 A7059549). This work was also supported by (3) the Brain Convergence Research Program of the National Research Foundation (NRF) funded by the Korea government (MSIT) (No. NRF-2021M3E5D2A01021156) and (4) Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-01373, Artificial Intelligence Graduate School Program (Hanyang University)). * Ministry of Science and ICT.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alabdulmohsin, I.; Han, Y.; Shen, Y.; Zhang, X. Content-agnostic malware detection in heterogeneous malicious distribution graph. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, Indianapolis, IN, USA, 24–28 October 2016; pp. 2395–2400.
2. Hong, J.; Park, S.; Kim, S.W.; Kim, D.; Kim, W. Classifying malwares for identification of author groups. *Concurr. Comput. Pract. Exp.* **2018**, *30*, e4197. [CrossRef]
3. Souri, A.; Hosseini, R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Hum. Centric Comput. Inf. Sci.* **2018**, *8*, 1–22. [CrossRef]
4. Christodorescu, M.; Jha, S.; Seshia, S.A.; Song, D.; Bryant, R.E. Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 8–11 May 2005; pp. 32–46.
5. Cheng, S.M.; Ao, W.C.; Chen, P.Y.; Chen, K.C. On modeling malware propagation in generalized social networks. *IEEE Commun. Lett.* **2010**, *15*, 25–27. [CrossRef]
6. Pitolli, G.; Aniello, L.; Laurenza, G.; Querzoni, L.; Baldoni, R. Malware family identification with birch clustering. In Proceedings of the 2017 International Carnahan Conference on Security Technology, Madrid, Spain, 23–26 October 2017; pp. 1–6.
7. Rafique, M.Z.; Caballero, J. Firma: Malware clustering and network signature generation with mixed network behaviors. In *International Workshop on Recent Advances in Intrusion Detection*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 144–163.
8. Huang, W.; Stokes, J.W. MtNet: A multi-task neural network for dynamic malware classification. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, San Sebastián, Spain, 7–8 July 2016; pp. 399–418.
9. Kong, D.; Yan, G. Discriminant malware distance learning on structural information for automated malware classification. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 11–14 August 2013; pp. 1357–1365.
10. Plohmann, D.; Clauss, M.; Enders, S.; Padilla, E. Malpedia: A collaborative effort to inventorize the malware landscape. In Proceedings of the Botconf, Montpellier, France, 6–8 December 2017.

11. Kreuk, F.; Barak, A.; Aviv-Reuven, S.; Baruch, M.; Pinkas, B.; Keshet, J. Deceiving end-to-end deep learning malware detectors using adversarial examples. *arXiv* **2018**, arXiv:1802.04528.
12. Yan, J.; Qi, Y.; Rao, Q. *Detecting Malware with an Ensemble Method Based on Deep Neural Network*; Security and Communication Networks: London, UK, 2018.
13. Bhandai, S.; Panihar, R.; Naval, S.; Laxmi, V.; Zemmari, A.; Gaur, M.S. Sword: Semantic aware android malware detector. *J. Inf. Secur. Appl.* **2018**, *42*, 46–56.
14. Bayer, U.; Comparetti, P.M.; Hlauschek, C.; Kruegel, C.; Kirda, E. Scalable, behavior-based malware clustering. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2009; Volume 9, pp. 8–11.
15. Islam, R.; Tian, R.; Batten, L.M.; Versteeg, S. Classification of malware based on integrated static and dynamic features. *J. Netw. Comput. Appl.* **2013**, *36*, 646–656. [CrossRef]
16. Sneha, S.; Malathi, L.; Saranya, R. A survey on malware propagation analysis and prevention model. *Int. J. Adv. Technol.* **2015**, *6*, 1–4.
17. Ronen, R.; Radu, M.; Feuerstein, C.; Yom-Tov, E.; Ahmadi, M. Microsoft malware classification challenge. *arXiv* **2018**, arXiv:1802.10135.
18. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; pp. 183–194.
19. Bilar, D. Opcodes as predictor for malware. *Int. J. Electron. Secur. Digit. Forensics* **2007**, *1*, 156–168. [CrossRef]
20. Costantini, G.; Ferrara, P.; Cortesi, A. Static analysis of string values. In Proceedings of the International Conference on Formal Engineering Methods, Durham, UK, 26–28 October 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 505–521.
21. Sikorski, M.; Honig, A. Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software. No Starch Press. Available online: https://nostarch.com/malware (accessed on 1 February 2012).
22. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv. CSUR* **2008**, *44*, 1–42. [CrossRef]
23. Grégio, A.R.; Fernandes Filho, D.S.; Afonso, V.M.; Santos, R.D.; Jino, M.; de Geus, P.L. Behavioral analysis of malicious code through network traffic and system call monitoring. In *Evolutionary and Bio-Inspired Computation: Theory and Applications V*; International Society for Optics and Photonics: Bellingham, WA, USA, 2011; Volume 8059, p. 80590O.
24. Han, J.; Pei, J.; Kamber, M. *Data Mining: Concepts and Techniques*; Elsevier: Amsterdam, The Netherlands, 2011.
25. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
26. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1067–1077.
27. Dekking, F.M.; Kraaikamp, C.; Lopuhaä, H.P.; Meester, L.E. *A Modern Introduction to Probability and Statistics: Understanding Why and How*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005.
28. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
29. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the International Joint Conference on Artificial Intelligence, Montreal, QC, Canada, 20–25 August 1995; Volume 14, pp. 1137–1145.
30. Berrar, D. Cross-Validation. 2019. Available online: https://www.sciencedirect.com/science/article/pii/B978012809633820349X?via%3Dihub (accessed on 20 May 2021).
31. Sokolova, M.; Lapalme, G. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manag.* **2009**, *45*, 427–437. [CrossRef]
32. Wagner, M.; Fischer, F.; Luh, R.; Haberson, A.; Rind, A.; Keim, D.A.; Aigner, W. A survey of visualization systems for malware analysis. In Proceedings of the Eurographics Conference on Visualization, Cagliari, Italy, 25–29 May 2015; pp. 105–125.