Large-Scale Malicious Software Classification with Fuzzified Features and Boosted Fuzzy Random Forest

Fang-Qi Li, Shi-Lin Wang*, *Senior Member, IEEE,* Alan Wee-Chung Liew, *Senior Member, IEEE,* Weiping Ding, *Senior Member, IEEE,* Gong-Shen Liu

***Abstract*—Classification of malicious software, especially in a very large dataset, is a challenging task for machine intelligence. Malware can have highly diversified features, each of which has highly heterogeneous distributions. These factors increase the difficulties for traditional data analytic approaches to deal with them. Although deep learning-based methods have reported good classification performance, the deep models usually lack interpretability and are fragile under adversarial attacks. To solve these problems, fuzzy systems have become a competitive candidate in malware analysis. In this paper, a new fuzzy-based approach is proposed for malware classification. We focused on portable executable files in the Windows platform and analyzed the distributions of static features and content-oriented features. Fuzzification was used to reduce the ubiquitous impact of noise and outliers in a very large dataset. Finally, a novel boosted classifier consisted of fuzzy decision trees and support vector machine is proposed to perform the malware classification. By using fuzzy decision trees, the inner structure of the classifier can be readily interpreted as discriminative rules, while the novel boosting strategy provides state-of-the-art classification performance. Extensive experimental results showed that our method significantly outperformed several state-of-the-art classifiers.**

***Index Terms*—Malware classification, fuzzy decision tree, boosted random forest, computer security.**

I. Introduction

MALICIOUS software is a concomitant threat of computer systems. Although tremendous effort has been devoted to developing security tools that protect electronic devices, the malware industry still takes the prevenient position and causes continual damage in commerce, privacy, and even physical security. Nowadays, the diversity of malwares and their speed of propagation grow even faster with the emergence of modern communication technologies like smartphones, the internet of things and cloud service. Variants of malwares have infiltrated into almost all available platforms to peek, steal or destroy valuable information. As of February 2020, AV-Test Gmbh [1] has collected over 1,024 million malwares, of which 19 million were exposed within just the first two months of 2020. The analysis of such diversified malwares is beyond the capability of current intelligent systems, and their robustness, efficiency, and scalability to large datasets are some of the major challenges. So far, techniques based on machine learning has been recognized as a promising approach to analyze and prevent malwares [1]–[3]. But when a vast number of malware samples are provided online, whether current paradigms scale well to very large collections of malwares remains a questionable issue.

Machine learning models are mostly applied to two aspects of malware analysis: (i) malware detection, which classifies between malicious and benign software, and (ii) malware classification, which classifies between different families or categories of malwares. The problem of malware detection has attracted much research [4]–[6]. Reports on a series of benchmark datasets have verified the applicability of many machine learning approaches such as deep learning [2], random forest [5] and boosting [7]. However, the problem of malware classification turns out to be much more challenging. Different categories of malwares have different functionalities and invoke different level of alert from users, and they should not be responded identically. To name a few, ransomware that kidnaps computers and demands ransom occupies a tiny portion of all malwares, yet it brings significant loss for large companies that have important data stored in their disks. The attack of WannaCry (ransomware that robbed users for bitcoin) that erupted since May 2017 has evaporated billions of dollars [8]. Password stealer (PWS) is particularly harmful for individual users since most users tend to use identical or homogeneous password for their e-mail, social network and banking accounts. The attack of PWS on one single electronic device could bring a potentially much larger threat to an individual's possessions, privacy, and security. Virus and worms have recently aimed at high-performance computers in colleges, universities, and companies to steal computing resources for bitcoin mining. Therefore, malware classification can help to deploy different pertinent reaction strategies to better protect different computer systems. Moreover, the classifier itself might provide information about the differences between different categories of malwares as well as their forensics information.

(Corrseponding author: Shi-Lin Wang.)

Fang-Qi Li, Shi-Lin Wang and Gong-Shen Liu are with School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. (e-mail: solour_lfq@sjtu.edu.cn; wsl@sjtu.edu.cn; lgshen@sjtu.edu.cn)

Alan Wee-Chung Liew is with School of Information and Communication Technology, Griffith University, Gold Coast Campus, QLD 4222, Australia. (e-mail: a.liew@griffith.edu.au)

Weiping Ding is with School of Information Science and Technology, Nantong University, Nantong 226019, China. (e-mail: ding.wp@ntu.edu.cn)

The work described in this paper was supported by National Natural Science Foundation of China (61771310, 61976120).

[1] https://www.av-test.org/en/statistics/malware/

Hitherto, both traditional machine learning and deep learning models have been applied in malware analysis. Both methods have yielded high accuracy in malware detection, while the results for malware classification are relatively scarce. Many of the studies on malware classification stem from the Microsoft 2015 Kaggle competition [9], in which deep learning methods based on the visual feature of malware obtained an accuracy of 99.8%. Yet deep learning-based methods suffer from several problems including (i) the complexity of the model structure prevents interpretability and explainability, (ii) the features extracted (especially visual feature) consume too much memory and their validity remains vague, (iii) the model can be attacked by gradient based adversarial methods as in image classification tasks [10]–[12]. Recently, fuzzy theory has been incorporated into deep learning system to provide interpretability and robustness [13]–[16]. A large family of fuzzy decision-makers such as fuzzy clustering [17], [18], fuzzy support vector machine [19] and fuzzy decision tree [5], [20], have been proved to be effective in a range of data analysis challenges. There have been some applications of fuzzy systems to malware detection and classification [4], [5], [21], [22]. The fuzzy philosophy can be reflected in the feature extraction module and the classification module, both of which are indispensable elements in a malware analysis system, since the raw input is usually a binary file that cannot be processed directly. For the feature extraction module, the extracted numerical features can be fuzzified into linguistic terms, such fuzzification provides an intuitive but computable description of features. Meanwhile, the influence of noise in the data can be minimized. For the classification module, the parameters can be fuzzified into linguistically meaningful outputs from which fuzzy rules can be formulated as decision boundaries that make the decision-making more intuitive and robust. By incorporating fuzzy sets, fuzzy rules and corresponding modules, a system is able to yield comprehensible decision boundaries while preserving its accuracy and robustness. Both aspects are crucial for software security and information forensics.

In this paper, we adopt fuzzy theory to extract and analyze features, as well as to classify malwares. To evaluate the scalability of a malware classification method to big data, the VirusShare dataset, which continuously collects malwares from the web is used in this study. We firstly extract features from this very large collection of malwares by using both analytic tools and linguistic statistics. By examining the characteristics of the feature distribution, both the static and content-oriented features are fuzzified accordingly. Then a fuzzy random forest and a support vector machine (SVM) are designed to classify the static and n-gram features, respectively. Finally, a set of extra fuzzy decision trees are employed following the adaptive boosting strategy to further boost the classification accuracy and the generalization ability. The main contributions of our work are three-fold:

1) A new static and content-oriented feature representation are proposed for large-scale (over 200k samples) malicious software classification based on fuzzy partition and fuzzy set theory.
2) A new malware classification framework is proposed, which consists of a fuzzy random forest and a SVM to perform malware classification based on the fuzzified static features and the fuzzified n-gram features, respectively, followed by an adaptive boosting module using fuzzy decision tree to generate the final classification result. Moreover, the fuzzy rules in the proposed classification approach are explainable.
3) Empirical results obtained using a very large dataset indicated that fuzzy-based systems have comparable accuracy and a number of advantages such as interpretability and robustness.

This paper proceeds as follows. In Section II, the background and related works are reviewed. In Section III, the proposed method is elaborated in detail with the explanation of the motivations behind. Experimental results and discussions are presented in Section IV, and Section V draws the conclusion.

II. Related Works

A. Malware Classification

Malware detection has long been a task of concern for researchers. To detect whether a given executable is safe or not is a relatively easy task for machine intelligence since there are many available datasets of malwares such as VX-Heaven, IoT, Ransomware [5], while the benign samples such as DLLs are readily obtainable. In contrast, the reported works on malware classification are relatively fewer, because very few datasets of malware are properly labeled for researchers to perform classification, let alone very large datasets. The most widely referred to labeled dataset is Kaggle 2015 [9] with 10K samples containing nine families and six categories (worm, adware, backdoor, Trojan, TrojanDownloader and obfuscated malware). Models with optimal performance in Kaggle almost uniformly adopted deep learning [2], [23]. Microsoft sterilized the dataset by erasing the PEheader of all the provided binary files. This disables the application of some static feature extraction tools that parse the PEheader, and emphasizes features that rely on statistics from the file content. However, this dataset can no longer be called comprehensive from today's point of view. Online malware dataset such as the VirusShare dataset provides over one million malware samples each year, which includes over 10K families and thirty categories. To research the classification of a large volume of up-to-date malwares, one has to fetch data from websites such as the VirusShare dataset and label them manually.

Malware on different platforms or operating systems cannot be handled under a unified framework due to the following reasons: (i) Anti-malware engines for different platforms maintain different collections of malware categories; (ii) Software on different platforms have different code structure, leading to distinctive features and statistics. So far, Windows and Android have been the platforms that the majority of hackers, security staff and researchers focused on.

In this paper, we focus on portable executable (PE) files on Windows operating system since their occurrence or diversity is still dominating compared with other platforms such as

Android and macOS. According to the AV-TEST statistics, the ratio between the numbers of new malwares in Windows, Android, and macOS in 2019 is roughly $1 : 0.03 : 0.0007$.

B. Feature Extraction for Malware Analysis

Feature extraction for malware analysis has been studied by the computer security community as well as the machine learning community. Malware classification generally uses the same set of features as in malware detection. Features from malware can be grouped into three categories: dynamic, static, and content-oriented features [1].

1) Dynamic Features: Dynamic features are obtained by executing a program in a sandbox and recording its suspicious behaviors. Some of the widely used dynamic features are the sequence of API calls, that of the socket connections, the list of interactive files and the network traffic statistics. Several detection methods based on graph embedding or graph similarity of dynamic features have been demonstrated to be very effective [24]. However, the extraction of dynamic features suffers from arbitrariness (testers have to manually define the set of malicious behaviors) and high cost (need to build a sandbox that keeps the malware unaware and safely executes hundreds of thousands of executables) [1].

2) Static Features: Static features are obtained by parsing an executable file instead of executing it. Representative parsing tools include `pefile`, `Peframe`, disassembler software, VirusTotal report [2], shell command lines, etc [1], [2]. For example, `pefile` parses a file with the correct PE format into an object from which informative features can be read: the size of imported/exported resources, the number and size of sections, the list of linked DLLs and used APIs, information about debugging and address, etc. A disassembler compiles a binary file to its assembly code that contains the sequence or graph of operation which carries much information. Static features have served as the input of backend analyzers since the beginning of malware analysis owing to their easy accessibility. According to [2], although researchers have claimed to come up with more complex and intelligent ways of feature extraction, static features remains to be the most discriminative candidate. Even if deep learning models come into malware analysis, the static features still play a central part in malware analysis.

3) Content-Oriented Features: Apart from building a sandbox or applying rule-based analysis, some models digest the entire binary file as input to analyze its content without parsing it. By using an analogy between a binary file and a multimedia object such as a piece of text or image, deep learning models can be straightforwardly applied. For example, by observing the similarity between the classification of malwares from its binary representation and the classification of an article from its text, some linguistic models such as n-gram or latent semantics analysis that are used in Natural Language Processing (NLP) can be applied to malware classification. The efficacy of n-gram features in malware detection has been extensively studied [25] and empirically verified [2]. On the other hand, great advancement in computer vision suggests that compiling a binary file into a grayscale image can preserve the information and allow the use of computer vision algorithms. Visualization of binary file combined with a convolution neural network yields good performance in Kaggle and larger datasets [2], [23]. However, content-oriented features sharply increase the size of the feature vector, where some of them are unstable, fragile against adversarial attacks [10], and are impossible to be understood or summarized into formal knowledge.

[2]https://www.virustotal.com/

C. Classifiers for Malware Classification

Many classifiers have been applied to malware detection or classification, depending on the statistics of the extracted features and the size of the dataset. Pai et al. [26] used clustering, expectation maximization and hidden Markov models for six-class classification on a dataset with 11,000 samples. Souri and Hosseini [1] provided a comprehensive review of the application of many state-of-the-art classifiers such as support vector machine, naive Bayes classifier, random forest, and neural network in malware classification.

With the recent success of deep learning models in many applications, convolutional neural network (CNN) has also been successfully applied to malware detection by interpreting the binary file of software as an image [27]–[30], while [23] covered its performance in malware classification. Although the accuracy has turned out to be high according to [23], this approach has not been widely adopted within the computer security community for the following reasons: (i). To use CNN, a software has to be transformed into an image, during which process a lot of codes as well as information are lost. The size of the digital image provides a trade-off between accuracy and efficiency, whose optimal value is hard to define. (ii). CNN involves much more parameters than traditional machine learning methods, making it harder to train or to be deployed onto terminal devices. (iii). Traditional machine learning models usually yield decision boundary or classification rules that reflects the relationship between the statistical features of software. Traditionally, software engineers would observe the API call and the dynamic behavior of a software to determine whether it is benign or malicious. In contrast, CNN models usually learn features that are hard to interpret or to reason on.

Among various classifiers, the decision tree is of special interest by being simple to implement and straightforward in interpretation. In 2012, Adobe released a Python script that detects malware with only several hundreds of lines that obtained 94% accuracy on 130,000 malwares and 16,000 benign files. The model is simply an ensemble of four decision trees(J48, J48Graft [31], PART, Ridor [32]) based on static features extracted by `pefile`. Since most static features consist of a continual spectrum of values, the node in the decision tree is a crisp partition.

Besides traditional classifiers, fuzzy systems have also shown their advantages in malware analysis. Zhang et al. [22] built a dynamic fuzzy system on traditional static features for malware detection on a dataset with 604 samples. Dovom et al. [5] used a fuzzy pattern tree with the graphical operational

code features to classify 22,000 malwares. Both datasets in [5], [22] are relatively small in scale. [4], [21], [33] explored the application of the neuro-fuzzy approach with static features to detect Android malware and to classify Windows malware, during which a variational procedure is adopted to better fuzzify the features. Although [33] reported the application of the fuzzy system to a large dataset (ten repositories from Virus Share, over 130K malware samples), the overall classification accuracy is relatively low compared with non-fuzzy methods such as random forest. In summary, the previous works were more about reporting the efficacy of applying fuzzy systems to malware analysis rather than justifying the superiority of such approach. In addition, the features extracted in the fuzzy systems were confined to the static features.

Fuzzy rules, which has been widely applied in cybernetics [34]–[36], malware detection [21] as well as classification [33], [37], [38], is one of the most representative fuzzy based classifiers. For a feature vector $\mathbf{x} = (x_1, x_2, \cdots)^{\mathrm{T}}$, a fuzzy rule R_l indexed by l classifies $\mathbf{x}$ into a class C_l using:

$$\begin{aligned} R_l : \textbf{IF } (x_{I_{l,1}} \text{ is } \mathcal{F}_{I_{l,1}}) \textbf{ and } & (x_{I_{l,2}} \text{ is } \mathcal{F}_{I_{l,2}}) \cdots \\ \textbf{and } & (x_{I_{l,L_l}} \text{ is } \mathcal{F}_{I_{l,L_l}}) \\ \textbf{THEN } & \mathbf{x} \text{ belongs to class } C_l \\ & \text{with confidence } \mathrm{CF}_l, \end{aligned}$$

where L_l is the number of antecedents in R_l, each member in $\{I_{l,i}\}_{i=1}^{L_l}$ identifies a component of $\mathbf{x}$, $x_{I_{l,i}}$ and a fuzzy set $\mathcal{F}_{I_{l,i}}$ defined in the domain of $x_{I_{l,i}}$. Formally, $\mathcal{F}_{l,i}$ can be embedded by a functional $\mu_{I_{l,i}}$ that maps the domain of $x_{I_{l,i}}$ to $[0, 1]$, and thus the fuzzy logic clause $(x_{I_{l,i}} \text{ is } \mathcal{F}_{I_{l,i}})$ takes value $\mu_{I_{l,i}}(x_{I_{l,i}})$ rather than zero or unity as in Boolean logic. Finally, the evidence provided by R_l is a combination of all L_l antecedents, which is usually derived by a reduction operator $\mathcal{O}$:

$$\mathrm{EV}(R_l, \mathbf{x}) = \mathrm{CF}_l \cdot \mathcal{O}(\{\mu_{I_{l,i}}(x_{I_{l,i}})\}_{i=1}^{L_l}),$$

where $\mathcal{O}$ can be product, min, etc. Finally, a fuzzy rule-based classifier considers the results of all J fuzzy rules $\{\mathrm{EV}(R_l, \mathbf{x}), C_l\}_{l=1}^{J}$ and yields a classification result.

The problem with fuzzy rules is that they cannot grow automatically. Usually, a basic set of rules is constructed first and then interpolation is performed to better fit the observed data [39]. On the other hand, the fuzzy version of a decision tree can also use fuzzy partitions as input, yet they can grow just as normal decision trees. This gives rise to a branch of classifiers known as a fuzzy decision tree [20], [40]–[42] that incorporates fuzzy partition in the input.

III. The Proposed Method

The proposed malware classification system consists of a feature extraction module and a classification module. The first module extracts both the static and the n-gram features from an executable in binary form, and the latter module fuzzifies these attributes and classifies the input into a malware category.

A. Feature Selection

When selecting the optimal combination of features for malware classification, convenience, efficacy, interpretability, and robustness are the four key issues that need to be considered comprehensively. Dynamic features are excluded due to their high cost in operating time and ambiguity in extraction. Static features are adopted for their accessibility and high discriminative power [1].

To deal with some cases where the static features are unavailable due to the source of the data, content-oriented features are also used. Specifically, the n-gram features are adopted as the content-oriented features because they are concise, discriminative, readily applied in almost all classifiers and can be easily fuzzified based on Hamming distance. Features that can be fuzzified, known as fuzzy features, are of special interest due to their robustness against noise and lower training variances [13].

For the above reasons, the static features and the content-oriented n-gram features are adopted to classify different various kinds of malwares. The detailed feature extraction approaches are elaborated as follows.

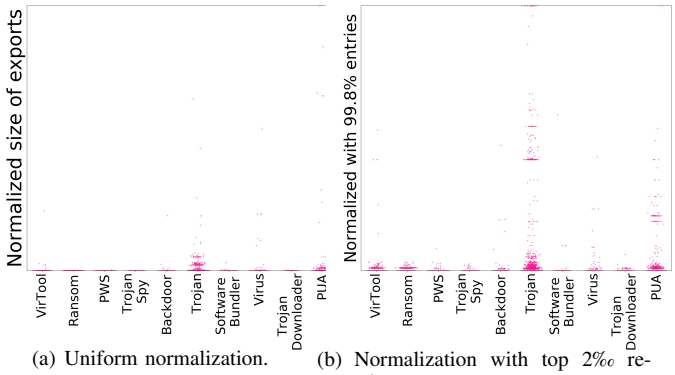
1) Static Features: To extract static features, we follow the work of [2], [33] and select a collection of attributes readily provided by the Python library `pefile` and Linux commands `size` and `file`. Features from over 200K samples downloaded from the VirusShare dataset [3] are collected and are denoted as raw features. Their statistics are listed as in Table. I.

TABLE I
Raw static features

Feature	Type	Range
`Number_Of_Sections`	Integer	[0,40]
`Size_Of_Export`	Integer	[0,10000+]
`Size_Of_Resources`	Integer	[0,2000000+]
`Size_Of_Debug`	Integer	{0,28,56,84,112$\cdots$728}
`Import_Address`	Integer	[0,14000000+]
`.text`	Integer	[0,10000000+]
`.data`	Integer	[0,2300000+]
`.bss`	Integer	[0,1100000+]
`.size`	Integer	[0,20000000+]
`Image_Version`	Integer	[0,8000+]
`Virtual_Size`	Integer	[0,300000+]
`.text/.size`	Float	[0,1)
`.data/.size`	Float	[0,1)
`.bss/.size`	Float	[0,1)

From Table. I, it is observed that for a large dataset, the values of some raw static features (or attributes) fluctuate with a large variance. Hence, when applying classification algorithms such as the decision tree, it would be extremely time-consuming to examine the exact partitions for the above attributes. Moreover, the raw static features also suffer from heavy noise and outliers that hinder normalization. This fact can be observed in Fig. 1. When applying a linear, uniform normalization onto the range of $[0, 1]$, (as shown in Fig. 1a), almost all entries will fall into a small interval due to outliers. By removing the top 2‰ entries with the largest values, such severe data aggregation phenomenon still appears as shown in Fig. 1b. This abnormality makes further partitions difficult for the decision tree classifier, while the weight-based classifiers are going to suffer from their normalizers.

[3] https://virusshare.com/



(a) Uniform normalization. (b) Normalization with top 2‰ removed.

Fig. 1. Data distribution of `Size_Of_Export`.

In [4], [21], the fuzzy features rather than the raw features were adopted to represent the static attributes to deal with the above data distribution problem. The self-organizing mapping (SOM) algorithm was adopted to cluster the static attributes into five linguistic fuzzy sets: *very small*, *small*, *middle*, *big*, *very big*. Such treatment is an orthodox preprocessing step for fuzzy systems. Although it helps to reduce the impact of noise and outliers, the clustering algorithm cannot sufficiently explore the complicated underlying pattern behind the static attributes from malwares. As shown in Fig. 2, the data distributions of a static attribute vary greatly among different malware classes and a fixed number of clusters is obviously inappropriate to handle the data distributions of all the malware classes.

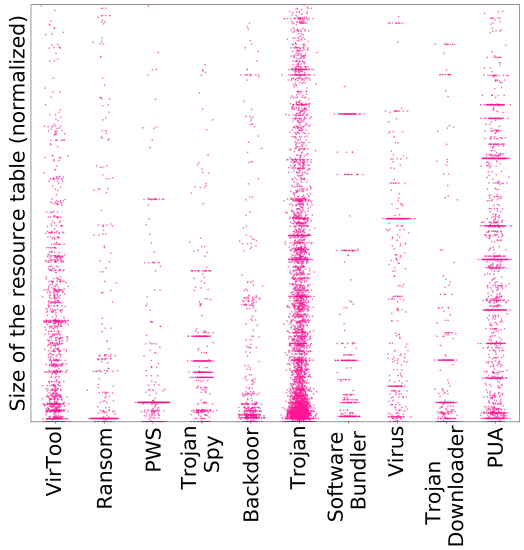


Fig. 2. The distribution of a normalized attribute, `Size_Of_Resources`, after removing top 1‰ entries. Each sample is added with a white Gaussian noise so the number of samples of a cluster can be observed.

Moreover, it can be observed that many attribute values aggregate around some specific values (the bars in Fig. 2). Taking data noise into consideration, the fuzzy partition is adopted to describe the data distribution of the raw static features. A fuzzy partition $\mathcal{F}$ of an attribute with domain $\mathcal{X}$ is characterized by a fuzzy membership function $\mu_{\mathcal{F}} : \mathcal{X} \to [0, 1]$. For $x \in \mathcal{X}$, x is said to belong to one side of the partition $\mathcal{F}$ with degree $\mu_{\mathcal{F}}(x)$ and to the other side with degree $(1 - \mu_{\mathcal{F}}(x))$. Since both fuzzy rules [33] and decision tree [43], [44] yield good performance in malware analysis, we hybridize the idea behind these two methods by allowing fuzzy partitions of both types: (i) *less than* partition $x < c$, which is the fuzzy version of the ordinary partition for a continuous attribute:

$$\mu_{c,a}^{\text{Halfspace}}(x) = \begin{cases} 1, & x \le c \\ \dfrac{a + c - x}{a}, & c < x < c + a \\ 0, & x \ge c + a \end{cases}, \tag{1}$$

(ii) *equal to* partition $x = c$, which is similar to a fuzzy logic assertation for a discrete attribute:

$$\mu_{c,a}^{\text{Interval}}(x) = \begin{cases} 0, & x \le c - a \\ \dfrac{a - c + x}{a}, & c - a < x \le c \\ \dfrac{a + c - x}{a}, & c < x < c + a \\ 0, & x \ge c + a \end{cases}. \tag{2}$$

The intuitions behind (1) and (2) are better understood by visualizing them as Fig. 3. $\mu^{\text{Halfspace}}$ fuzzifies the idea of partitioning the domain of a specific attribute in a decision tree, while μ^{Interval} fuzzifies the idea of selecting a value from a discrete domain in splitting a node in a decision tree. Fuzzy partitions also have a prominent advantage during searching the threshold c because they only require an approximate rather than an exact value of the threshold and the ambiguity can be handled by the fuzziness. Note that when $a \to 0$, (1) and (2) degenerates gracefully into their crisp counterparts, a crisp partition and a Dirac indicator.

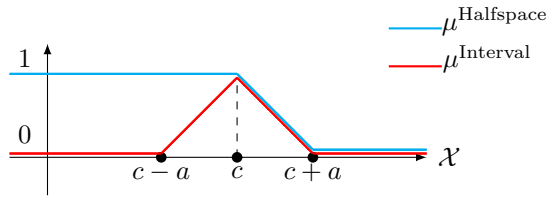


Fig. 3. $\mu^{\text{Halfspace}}$ and μ^{Interval}.

Remark: Only two parameters c and a are chosen to parameterize the fuzzy partition in our method. Formally, the shape of a fuzzy partition can be arbitrarily complex and contain many more parameters. Some works focus on decoupling the correlation between features, adopting a variational method or interpolation [21] to optimize the shape of the fuzzy membership function. However, these are not the most crucial variables on which this paper concentrates. So we ignore them from comparison and interested readers can refer to the references within to seek improvement from this perspective.

2) n-gram Features: The idea of using n-gram to classify malware stems from topic classification or analysis of natural language texts [45]. In a raw binary file, a character is defined as one byte and an n-gram word is a string consists of n successive bytes. We choose $n = 4$ empirically in our work, which balances the classification performance and the computational complexity.

To extract 4-gram features from the dataset, we first browse the entire training set of binary files and collect all appearing words to form a dictionary. The top K_1 most frequent words are kept. They are then sorted by their discriminative power and the top K_2 words are adopted as keywords. The 4-gram feature vector of an executable is a vector with K_2 components, each of which denotes whether or not one keyword appears in it.

To be illustrative, the top 1,000 most frequent words are treated as stop words and removed from the constructed dictionary and K_1 is set to $100,000$. The occurrence of this set of frequent words extracted from a set of malwares downloaded from the VirusShare dataset is shown in Fig. 4, whose detail is presented in Section IV.

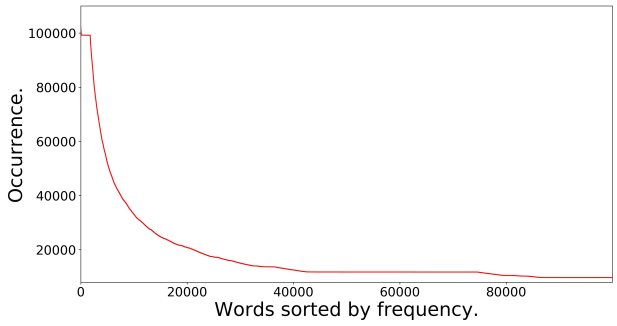


Fig. 4. The occurrence of top 100,000 frequent words.

Fig. 4 shows that the distribution of the frequency of 4-gram words follows the pattern of the frequency of words in natural language, i.e. following Zipf's law [46], which is another piece of evidence supporting using NLP models in malware analysis.

The discriminative power of a keyword w is measured by the mutual information [47] with respect to all categories $\mathcal{C}$:

$$\begin{aligned} I(w;\mathcal{C}) =& H_{|\mathcal{C}|}(\Pr(\mathcal{C})) - \Pr(w) H_{|\mathcal{C}|}\left(\Pr(\mathcal{C}|w)\right) \\ & - \left(1-\Pr(w)\right) H_{|\mathcal{C}|}\left(\Pr(\mathcal{C}|\neg w)\right), \end{aligned} \tag{3}$$

where $H_{|\mathcal{C}|}$ is the entropy of an information source with $|\mathcal{C}|$ types of signals, $\Pr(\mathcal{C})$ is the probability distribution of all categories, $\Pr(\mathcal{C}|(\neg)w)$ is the conditional distribution of all categories in the subset of malware samples with or without w, and $\Pr(w)$ is the proportion of executables containing w, whose values are approximated from the empirical distribution in the training set.

In NLP, the semantic similarity between words is used in many linguistic tasks [48], but such similarity between words is usually hard to capture due to the complexity of natural languages. Meanwhile, bitwise Hamming distance serves as a natural metric of the morphological similarity between words. Such similarity can be adopted to fuzzify the keywords for two reasons: (i) if two words have a shorter Hamming distance then they could be the same command with different addresses and thus they might have the identical utility as signatures. In fact, models that assume such similarities (e.g. the convolutional neural network in which the pixelwise similarity is transformed into contextual similarity) have achieved state-of-the-art performance [2], [23]; (ii) unlike static features, content-oriented features are known to be vulnerable facing adversarial samples. Bitwise modification can help a malware escape from the detectoror classifier [10]. Hence, a fuzzification in word matching can increase the cost of the attack and improve the robustness of the model. To examine the degree of similarity between informative words, we define the average bitwise distance within a collection of words $\mathcal{W}$ as (recall that each word contains 32 bits):

$$d(\mathcal{W}) = \frac{1}{\binom{|\mathcal{W}|}{2}} \sum_{w_1, w_2 \in \mathcal{W}} \frac{\mathbf{1}^{\mathrm{T}} w_1 \oplus w_2}{32}, \tag{4}$$

where $\oplus$ is bitwise *exclusive or*. For the collection of top $K_1 = 100,000$ frequent words, the average bitwise distance given by (4) is $d = 0.48$. For the 10%, i.e. $\frac{K_1}{10} = 10,000$, most informative words, the distance is reduced to $d = 0.45$, which demonstrates that the more informative words are more similar to each other. A statistical hypothesis testing is conducted to statistically validate this observation.

Hypothesis test: *The average bitwise distance between more informative words is shortened.* We assume the event that two bits are identical is subject to a Bernouli distribution $\mathrm{Ber}(\cdot)$, whose mean is given by the empirical mean on K_1, $d = 0.48$. The mean of the Bernouli distribution underlying the more informative words is d'. The null hypothesis and the alternative hypothesis are stated as follows:

$$\mathbf{H}_0 : d' \geq d,$$

$$\mathbf{H}_1 : d' < d.$$

To reject $\mathbf{H}_0$, we construct an event with minuscule probability assuming $\mathbf{H}_0$ holds. The empirical mean of the bitwise distance among the most informative words, $\bar{d}$, is the average of $N = \binom{10000}{2} \times 32$ pairs. We drop the dependency between them and approximating the distribution of $\bar{d}$ using central limit theorem, so roughly:

$$\bar{d} \sim \mathcal{N}\left(d', \frac{d'(1-d')}{N}\right).$$

Assuming $\mathbf{H}_0$ then the p-value of $\bar{d}$ taking value $d^* = 0.45$ is at most:

$$\begin{aligned} \int_{-\infty}^{d^*} \mathcal{N}\left(x \middle| d, \frac{d(1-d)}{N}\right) \mathrm{d}x &= \Phi\left(\frac{d^* - d}{\sqrt{\frac{d(1-d)}{N}}}\right) \\ &\approx \Phi(-2400) \ll 0.01. \end{aligned} \tag{5}$$

So it is safe to reject $\mathbf{H}_0$ and conclude that within the set of most informative words, the average Hamming distance is shorter so we assume that applying clustering or fuzzification among the keywords can delete repeated patterns and incorporate more discriminative patterns.

Formally, we fuzzify the event *word w belongs to an executable e* from the crisp indicator (whose value is e's component corresponding to w in its n-gram feature vector):

$$f(w, e) = \max_{w' \text{ appears in } e} \{\mathbb{I}[w' = w]\}, \tag{6}$$

where $\mathbb{I}[\cdots]$ is an indicator function whose value is one iff the input statement is true, into:

$$f_{\text{fuzzy}}(w,e) = \max_{w' \text{ appears in } e} \left\{ \mathbb{I}[\mathbf{1}^{\mathrm{T}}(w' \oplus w) \leq 2] \right\}. \quad (7)$$

To interprete from a fuzzy perspective, the generalization from (6) to (7) is tantamount to generalizing the indicator of the set containing only a single point w in the hyperspace of all possible words to a sphere with Hamming radius ≤ 2.

In practice, we browse the list of K_1 frequent words and delete a word if its Hamming distance to a previous one is less than three. Then the top K_2 informative words are selected from this filtered list. For an executable, its component corresponding to keyword w in 4-gram feature vector is set to one if there exists a word in this file with no more than two bits different from w. This process is stated in Algo. 1.

Algorithm 1 n-gram feature extractor.

Input: Training set in binary form $\mathcal{D}$, thresholds K_1, K_2, an executable in binary form e;
Initialize: Dictionary = $\{\}$, $\text{vec}_e = \mathbf{0}$;
for an executable e' in $\mathcal{D}$ **do**
  for a word w in e' **do**
    ++Dictionary[w];
  end for
end for
//Delete stop words.
Delete the top 1,000 words with the highest occurrence;
Save only the top K_1 most frequent words;
//Fuzzification.
Delete similar words;
Sort Dictionary using mutual information, i.e.(3).
Save only top K_2 words with the highest mutual information in Dictionary;
for a word w' in e **do**
  for a word w in Dictionary **do**
    if $\mathbf{1}^{\mathrm{T}}(w' \oplus w) \leq 2$ **then**
      $\text{vec}_e(w) = 1$;
    end if
  end for
end for
Output: vec_e.

B. The Classification Method

Considering the characteristics of the static and n-gram features of the malware, different classifiers are designed for each kind of features. For the static features, a fuzzy random forest is proposed to exploit the fuzzy partitions of the static features. For the n-gram features with a very high dimensionality, a multi-class support vector machine (SVM) with the radical basis function (RBF) kernel is employed as the classifier. Then the baseline classification result is obtained by a weighted voting over the results of each fuzzy decision tree and that of the SVM. Finally, a set of extra fuzzy decision trees (referred to as Ada-Trees in Fig. 5) are constructed following the adaptive boosting strategy to improve the classification accuracy and generalization ability. Combining the baseline classification results with those of the Ada-Trees, the final classification result can be derived. The structure of the proposed classification method is given in Fig. 5.

In the following formulation, we denote the collection of categories as $\mathcal{C} = \{C_l\}_{l=1}^{L}$, the collection of all attributes as $\mathcal{A} = \{A_m\}_{m=1}^{M}$, each A_m has its possible values distribute in domain $\mathcal{X}_m$. The collection of all training samples is $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^{N}$, where each $\mathbf{x}_n$ is a vector of length M with its m-th component $x_{n,m} \in \mathcal{X}_m$.

1) Fuzzy Random Forest: Based on the properties of the static features discussed in the previous subsection, a fuzzy random forest algorithm is proposed. The fuzzy decision tree is selected as the unit classifier, in which fuzzy partitions are adopted for node splitting. The fuzzy random forest, which is composed of a collection of fuzzy decision trees, has the following advantages and thus is suitable for our large-scale malware classification task: (i) the decision tree, or random forest were reported to have outstanding performance and scale well with big data [49]; and (ii) decision trees explore the feature space by conducting partitions, and thus the previous observations about the fuzziness of data distributions in the raw static features can be readily exploited.

To classify malwares using a fuzzy random forest based on the static features, the following issues in defining the model are required to be specified: the fuzzy partitions, the structure of the forest, the growing and pruning conditions, and the setting of hyperparameters.

The fuzzy partition has been elaborated in the previous subsection. In order to distinguish more than two classes and to increase the classifiers robustness against noise or missing attributes, a set of decision trees with various sets of training samples and candidate attributes are trained independently.

For each class $C \in \mathcal{C}$, a set of T fuzzy decision trees $\{\text{Tree}_{C,t}\}_{t=1}^{T}$ are trained to determine whether an input $\mathbf{x}$ belongs to C or not. To train each binary decision tree $\text{Tree}_{C,t}$, a subset $\mathbf{X}_{C,t}$ of $\mathbf{X}$ is randomly sampled such that instances from C occupies approximately fifty percent in $\mathbf{X}_{C,t}$. To achieve the greedy optimum in classifying $\mathbf{X}_{C,t}$, $\text{Tree}_{C,t}$ grows as an ordinary decision tree with two modifications: (i) the set of candidate attributes is a randomly sampled subset of $\mathcal{A}$ for each node in $\text{Tree}_{C,t}$, (ii) the ordinary crisp partition is replaced by the fuzzy partition.

To seek the optimal fuzzy partition at node v of $\text{Tree}_{C,t}$ is to find the best configuration of c and a in (1) and (2) which minimizes the classification loss. Given c and a, the dataset at node v, denoted by $\mathbf{X}_{C,t,v}$, is fuzzily partitioned into two splits P_1 and P_2. Each $\mathbf{x} \in \mathbf{X}_{C,t,v}$ belongs to P_1 with a membership value of $\mu_{c,a}(\mathbf{x})$ and to P_2 with a membership value of $1-\mu_{c,a}(\mathbf{x})$, where μ can be either $\mu^{\text{Halfspace}}$ or μ^{Interval}. The fuzzy totality of P_1 or P_2 is formulated as

$$N^1_{C,t,v} = \sum_{\mathbf{x} \in \mathbf{X}_{C,t,v}} \mu_{c,a}(\mathbf{x}), \quad (8)$$

$$N^2_{C,t,v} = \sum_{\mathbf{x} \in \mathbf{X}_{C,t,v}} \left(1 - \mu_{c,a}(\mathbf{x})\right). \quad (9)$$

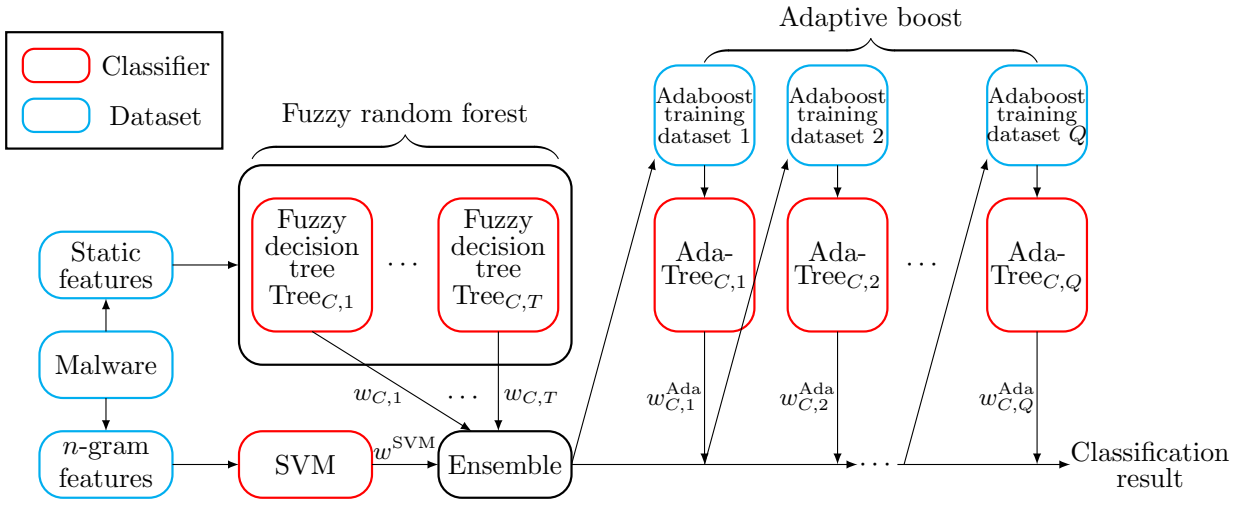


Fig. 5. The structure of the classifier for class C.

While the dominating class of P_1 or P_2 is given as:

$$C^1_{C,t,v} = \arg\max_{C'=C,\neg C} \left\{ \sum_{\mathbf{x}\in\mathbf{X}_{C,t,v}} \mu_{c,a}(\mathbf{x}) \cdot \mathbb{I}[\mathbf{x} \in C'] \right\}, \quad (10)$$

$$C^2_{C,t,v} = \arg\max_{C'=C,\neg C} \left\{ \sum_{\mathbf{x}\in\mathbf{X}_{C,t,v}} (1-\mu_{c,a}(\mathbf{x})) \cdot \mathbb{I}[\mathbf{x} \in C'] \right\}. \quad (11)$$

Hence the fuzzy number of mistaken samples in two partitions are:

$$E^1_{C,t,v} = \sum_{\mathbf{x}\in\mathbf{X}_{C,t,v}} \mu_{c,a}(\mathbf{x}) \cdot \mathbb{I}[\mathbf{x} \notin C^1_{C,t,v}], \quad (12)$$

$$E^2_{C,t,v} = \sum_{\mathbf{x}\in\mathbf{X}_{C,t,v}} (1-\mu_{c,a}(\mathbf{x})) \cdot \mathbb{I}[\mathbf{x} \notin C^2_{C,t,v}]. \quad (13)$$

The efficacy of partition c, a is measured by a fuzzy version of entropy following [50]:

$$\begin{aligned} \text{Entropy}_{C,t,v}(c,a) &= H_2\left(\frac{E^1_{C,t,v}+E^2_{C,t,v}}{N^1_{C,t,v}+N^2_{C,t,v}}\right) \\ &= H_2\left(\frac{E^1_{C,t,v}+E^2_{C,t,v}}{|\mathbf{X}_{C,t,v}|}\right). \end{aligned} \quad (14)$$

Fixing c and a, the node v is splitted into two nodes v_1, v_2 with:

$$\mathbf{X}_{C,t,v_1} = \{\mathbf{x} \in \mathbf{X}_{C,t,v} | \mu_{c,a}(\mathbf{x}) > 0\}, \quad (15)$$

$$\mathbf{X}_{C,t,v_2} = \{\mathbf{x} \in \mathbf{X}_{C,t,v} | \mu_{c,a}(\mathbf{x}) < 1\}. \quad (16)$$

If v is a leaf node, then its inference result is

$$C_v = \arg\max_{C'=C,\neg C} \left\{ \sum_{\mathbf{x}\in\mathbf{X}_{C,t,v}} \mathbb{I}[\mathbf{x} \in C'] \right\}. \quad (17)$$

To prevent overfitting, the depth of each fuzzy decision tree is restricted by a threshold $\mathcal{T}$. The algorithms for constructing a fuzzy decision tree is given in Algo. 2. Note that the fifth line in Algo. 2 is done by a grid search, the space where no less than 99.5% data lie is cut uniformly into 1000 parts, and a takes values in $\{0, 0.02\times c, 0.04\times c, \cdots, 0.2\times c\}$.

Algorithm 2 Fuzzy decision tree.

1: **Input:** A malware category C, the tree index t, the set of attributes $\mathcal{A}$, the dataset for this tree $\bar{\mathbf{X}}$, the maximal depth $\mathcal{T}$;
2: **Initialize:** root=$\{\bar{\mathbf{X}}, 0\}$, storage structure: Tree$_{C,t}$;
3: **Define:** split($v = \{\mathbf{X}', \tau\}$):
4: Sample $\mathcal{A}'$ from $\mathcal{A}$;
5: $c_v, a_v = \arg\max_{c\in A\in\mathcal{A}', a} \{\text{Entropy}_{C,t}(c,a)\}$ following (8)-(14) using $C, t, \mathbf{X}'$;
6: Save c_v, a_v into Tree$_{C,t}$;
7: Compute $\mathbf{X}_1$ following (15);
8: Compute $\mathbf{X}_2$ following (16);
9: **return** $v_1 = \{\mathbf{X}_1, \tau+1\}$, $v_2 = \{\mathbf{X}_2, \tau+1\}$;
10: **end def**
11: node_list = [root];
12: next_list = [];
13: **for** $i = 1$ to $\mathcal{T}$ **do**
14: next_list = [];
15: **for** node in node_list **do**
16: next_list.append(split(node));
17: **end for**
18: node_list = next_list;
19: **end for**
20: **Output:** Tree$_{C,t}$.

To classify an instance $\mathbf{x}$ using one fuzzy decision tree Tree$_{C,t}$, $\mathbf{x}$ is passed through all paths in Tree$_{C,t}$. This is the fundamental difference between a fuzzy decision tree and an ordinary decision tree. In an ordinary decision tree, a node is passed to the leaf through one and only one path. However, in a fuzzy decision tree, a node may go through multiple paths from the definition in (15) and (16). The weight of one path

$v_1, v_2, \cdots, v_H$ is calculated as:

$$\text{weight}(v_H, \mathbf{x}) = \prod_{h=1}^{H-1} (\mu_{c_h,a_h}(\mathbf{x}))^{\mathbb{I}[v_{h+1} \text{ corrseponds to } P_1 \text{ of } v_h]} (1 - \mu_{c_h,a_h}(\mathbf{x}))^{\mathbb{I}[v_{h+1} \text{ corrseponds to } P_2 \text{ of } v_h]}, \tag{18}$$

where v_H is a leaf of $\text{Tree}_{C,t}$. The inference result is derived by a vote of all leaves of $\text{Tree}_{C,t}$, i.e., $\mathbf{x}$ belongs to C with a confidence using (17), (18):

$$\text{Tree}_{C,t}(\mathbf{x}) = \frac{\sum_{v \text{ is a leaf of Tree}_{C,t}, C_v = C} \text{weight}(v, \mathbf{x})}{\sum_{v \text{ is a leaf of Tree}_{C,t}} \text{weight}(v, \mathbf{x})}. \tag{19}$$

And $\mathbf{x}$ belongs to $\neg C$ with a confidence $(1 - \text{Tree}_{C,t}(\mathbf{x}, C))$.

2) The Ensemble Module: Ensembling a collection of heterogeneous classifiers is a challenging research problem in itself [51]–[54]. Some of the ensembling schemes are majority voting, weighted voting as out-of-bag (OOB) vote (the weight of a classifier depends on its performance on an unobserved subset of training set) and dynamic voting (the weight of a classifier explicitly depends on $\mathbf{x}$). In our approach, the dynamic voting scheme, MWLFUS2 [20], is adopted for ensembling the fuzzy decision trees. For each $\text{Tree}_{C,t}$ in a binary classification task (referred to as the classification tree), a more shallow error tree is trained to predict whether an instance would be mistaken by $\text{Tree}_{C,t}$. The training label of a sample x for an error tree corresponding to $\text{Tree}_{C,t}$ is set to one if $\text{Tree}_{C,t}$ correctly classifies it and is zero otherwise. Then the output of the error tree, $w_{C,t}(\mathbf{x})$, serves as the dynamic weight for inference. For the SVM, a weight similar to the form given by Adaboost is adopted as $w^{\text{SVM}} = \frac{1}{2} \log\left(\frac{r}{1-r}\right)$, where r is the classification accuracy of SVM in the training set. The baseline ensemble deduces that $\mathbf{x}$ belongs to C with a confidence with (19):

$$\text{Base}(C, \mathbf{x}) = \sum_{t=1}^{T} w_{C,t}(\mathbf{x}) \times \text{Tree}_{C,t}(\mathbf{x}) + w^{\text{SVM}} \times \text{SVM}_C(\mathbf{x}), \tag{20}$$

where $\text{SVM}_C(\mathbf{x})$ is the output of SVM using the one-versus-all strategy.

3) The Adaptive Boost Module: Having equipped with a binary classifier for class C with T fuzzy decision trees and a one-versus-all SVM, we continue to boost its accuracy by adaptive boosting [55], [56] with Q extra trees $\{\text{AdaTree}_{C,q}\}_{q=1}^{Q}$.

The weight of training samples is uniform before boosting. To modify the weight of training samples for the q-th AdaTree, we first compute the classification accuracy of $\text{AdaTree}_{C,q-1}$ w.r.t. the weighted samples as $r_{C,q-1}$ (the baseline classifier can be regarded as $\text{AdaTree}_{C,0}$), let

$$\alpha_{C,q-1} = \frac{1}{2} \log \frac{r_{C,q-1}}{1 - r_{C,q-1}}. \tag{21}$$

The weight of an example $\mathbf{x}$ is firstly multipled by a factor $\exp\{\alpha_{C,q-1}\}$ if the previous classifier fails to correctly classify it and is shrinked by the same factor otherwise. These weights are then normalized as a probability distribution and a sampling procedure generates the training set for $\text{AdaTree}_{C,q}$. This process iterates for Q rounds.

Finally, all the $|\mathcal{C}|$ boosted binary classifiers select the classification result of $\mathbf{x}$ by incorporating (20) and extra adaboosting trees:

$$\arg\max_{C \in \mathcal{C}} \left\{ \text{Base}(C, \mathbf{x}) + \sum_{q=1}^{Q} \frac{1}{2} \log\left(\frac{r_{C,q}}{1 - r_{C,q}}\right) \text{AdaTree}_{C,q}(\mathbf{x}) \right\}.$$

IV. Experimental Results and Analysis

A. Dataset and Preprocessing

Many established models were built and verified on assorted datasets which are manually collected and maintained. But to actually examine a models ability to scale to actual big data requires a much larger dataset consists of samples collected on the fly.

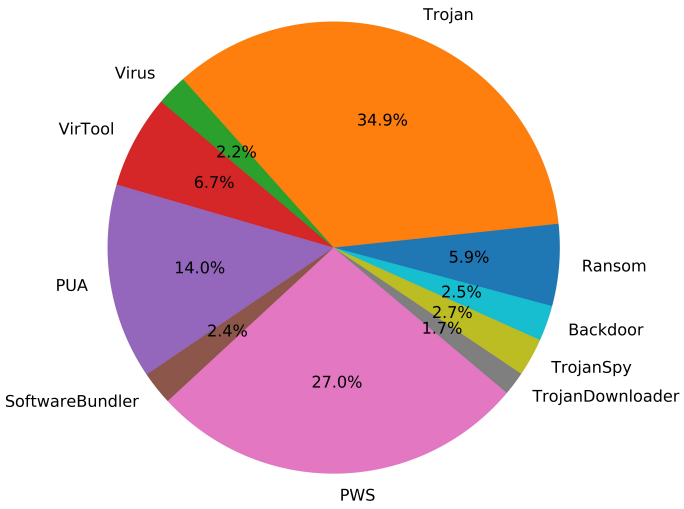


Fig. 6. The ratio of ten classes in the filtered Virus Share dataset.

We evaluate our model on a large-scale dataset, the VirusShare dataset. The VirusShare dataset collects a set of over 34 million assorted malwares across platforms like Android, macOS and various Windows versions. Updated malwares is uploaded to the VirusShare website, so the distribution of different malware categories is timely and realistic. We downloaded malwares from the 340th to the 373rd repository, and the dataset contained two million malwares and consumed 0.97TB. Files other than PE were removed. To obtain the ground truth of the classification, we resorted to Virus Total, which is an online malware analyzer. Virus Total takes the uploaded file or the unique MD5 code as inputs and returns the classification result of 72 anti-malware engines. Many of the engines have different ways of analysis and return diverse results. To simplify, we only parsed the outcome of the Microsoft anti-malware engine as in [33], and the PE files that escaped from the detection of Microsoft engine were removed. Finally, those classes with too few instances are deleted and the following ten representative classes are examined, including Virus, Backdoor, SoftwareBundler, PWS, Ransom, VirTool, Trojan, TrojanSpy, PUA and TrojanDownloader. The resulting 220K malware samples constituted the labeled dataset and the proportion of the ten classes is shown in Fig. 6.

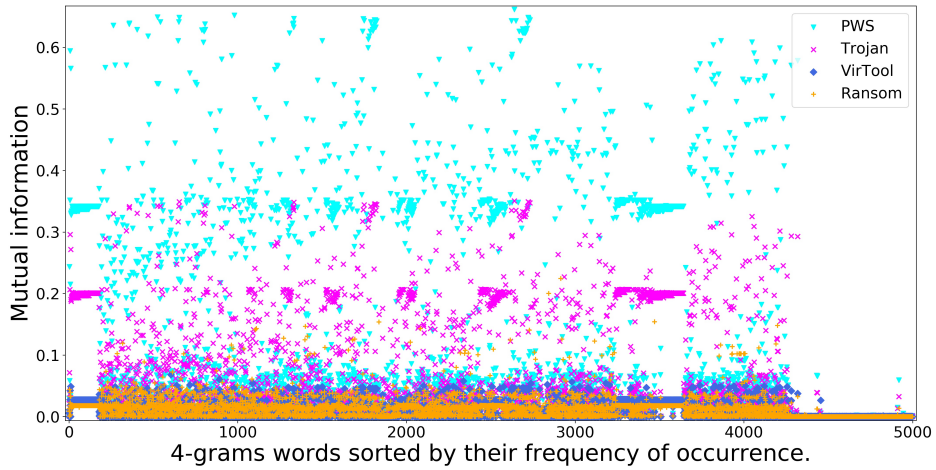


Fig. 7. Information gain of the most frequent words with respect to some malware categories.

The static features were collected using `pefile` and linux command line. The 4-gram features were extracted using a Python script.

B. Discriminative Power of the Fuzzy n-gram Features

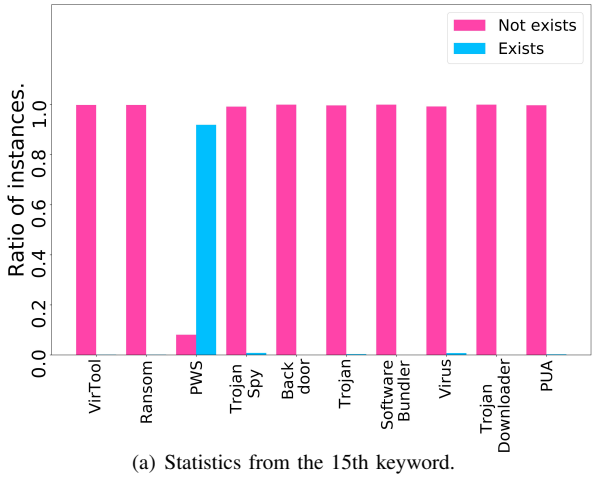
Before presenting the classification results and the comparisons, we briefly illustrate the discriminative power of the n-gram features. To the best of our knowledge, the n-gram features have not been formally analyzed in the malware classification scenario.

Given the 4-gram words with the highest frequency of occurrence, their information gain w.r.t. the four major malware categories, Trojan, PWS, VirTool, and Ransom, i.e. the mutual information between the existence of the word and the binary classification task between a category and the others, e.g., Ransom vs. the other were computed and shown in Fig. 7, where the 4-gram words are sorted according to their frequency. We chose the top 50,000 most frequent words and subsampled into 5,000 ones.

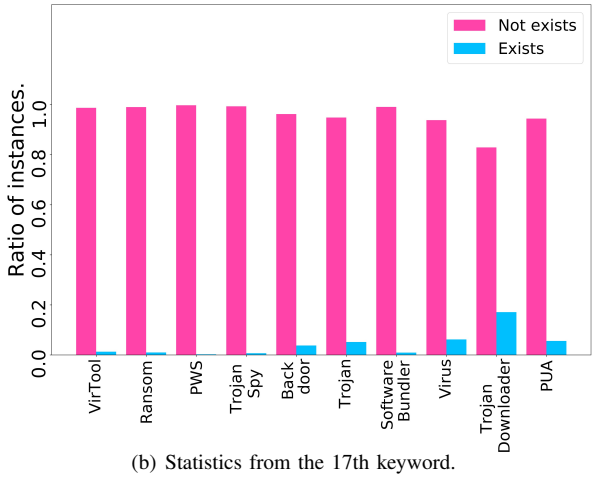
Some interesting conclusions can be drawn from Fig. 7:

1) The mutual information does not vary smoothly with the frequency, instead, there are many clusters and sharp cliffs in the graph.
2) The information gain of a word is different for different classes. This reflects a fact that some class is naturally easier to be distinguished from the others (in Fig. 7, PWS) using the 4-gram features.
3) The discriminative powers of a word w.r.t. different classes are positively correlated. For example, if a word is unhelpful in distinguishing between Ransom and the others then it is unlikely to provide information for discriminating other categories as well. Therefore using (3) instead of computing mutual information between each category and its complementary categories is sufficient.

As a result, we adopt (3) as the metric and select top $K_2 = 10,000$ keywords.



(a) Statistics from the 15th keyword.



(b) Statistics from the 17th keyword.

Fig. 8. The discriminative power of 4-gram features.

To delve into the discriminative power of a specific word, we visualize the statistics of two words in Fig. 8. If a file contains the word corresponding to Fig. 8a then we can readily conclude that it is a PWS. While if a file contains the word as in Fig. 8b then we suspect that it is very likely to be a TrojanDownloader and less likely to be a Trojan, virus, backdoor or a PUA and is unlikely to be of any other category.

Finally, to justify the efficacy of fuzzification of words using (7), we evaluate the classification accuracies using SVM based on the crisp word matching and the fuzzy word matching:

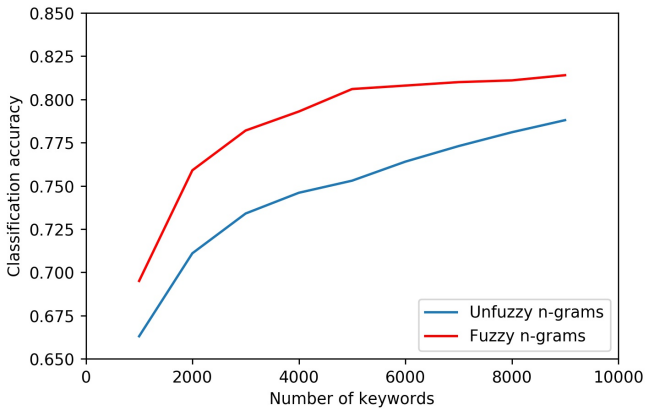


Fig. 9. Classification accuracy of SVM with original and fuzzy n-gram features.

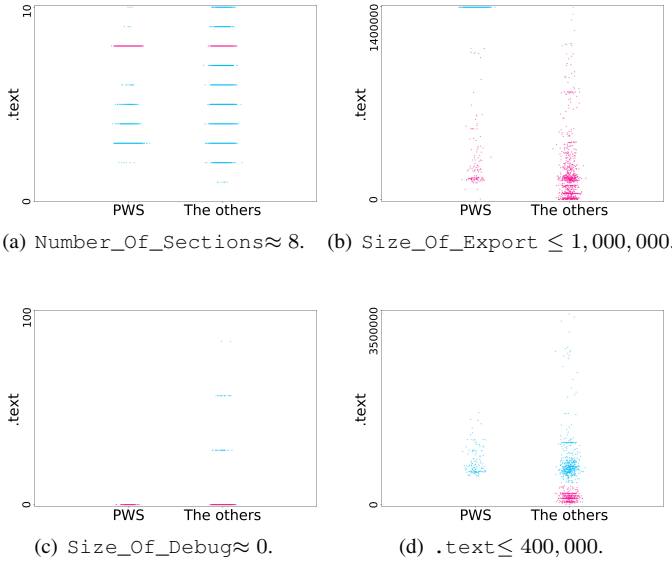
It can be observed from Fig. 9 that fuzzy n-gram features result in better classification accuracy. This is since fuzzy words grasp more discriminative patterns than original keywords. The fuzzification of words performs an aggregation (here aggregation is done based on Hamming distance), such aggregation, which akin to kernel smoothing, has the effect of smoothing noise and statistical variation, thus improving classification performance.

C. Interpretability of the Proposed Fuzzy Random Forest

To examine the interpretability of the proposed classifier, we illustrate the functionality of one extracted rule from the fuzzy random forest:

IF `Number_Of_Sections` ≈ 8 **and**
`Size_Of_Export` $\leq 1,000,000$ approximately **and**
`Size_Of_Debug` ≈ 0 **and**
`.text` $\leq 400,000$ approximately
THEN The input is not PWS.

The opeartion of the above rule above on a test set is visualized in Fig. 10, the subset that satisfies the antecedents is marked in red. It is clear that this rule correctly identifies the properties of PWS. Compared with the decision boundary of traditional machine learning models, this sort of fuzzy rules captures the statistical properties of different malware categories correctly without going too deeply into trivial numerical differences which potentially leads to overfitting. For example, a J48 decision tree for malware detection begins with one antecedent `Size_of_Export` < 211, while its fuzzy counterpart reads `Size_of_Export` < 200 with confidence score a, which enables malwares with `Size_of_Export` slightly higher than 211 to be further examined.



(a) `Number_Of_Sections`≈ 8. (b) `Size_Of_Export` $\leq 1,000,000$.

(c) `Size_Of_Debug`≈ 0. (d) `.text`$\leq 400,000$.

Fig. 10. Fuzzy partitions of two representative nodes in the fuzzy forest. Each sample is added with a white Gaussian noise so the number of samples of a cluster can be observed.

Fig. 10a and Fig. 10c are instances of fuzzy logic node where the partition of type (2) is adopted, while Fig. 10b and Fig. 10d visualize instances of type (1).

D. Malware Classification Results

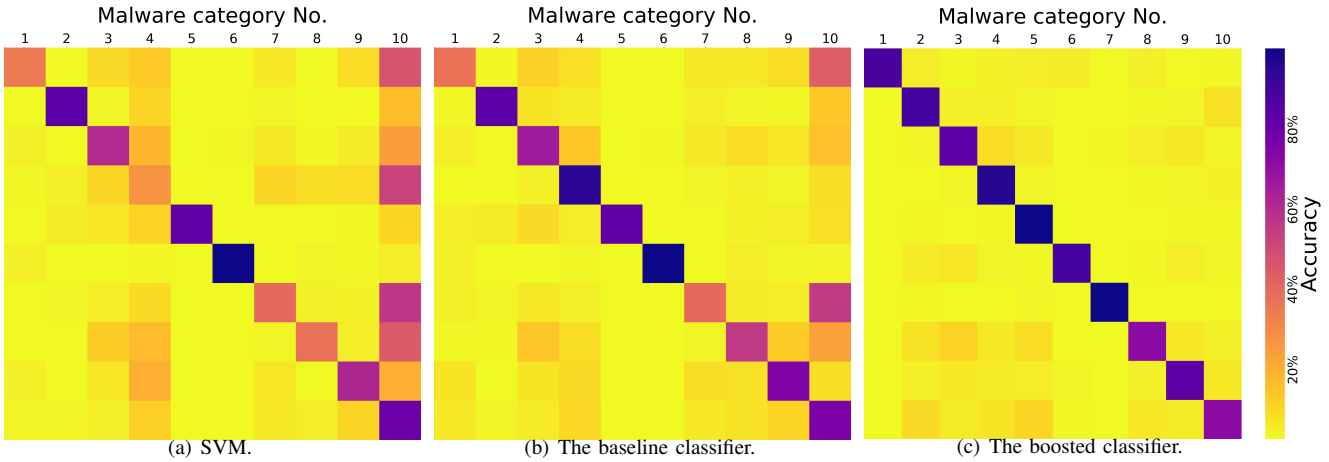
For each of the ten classes, we trained $T = 11$ independent fuzzy decision trees. Each fuzzy decision tree grew until the height of the tree exceeds $\mathcal{T} = 9$. Each binary classification was then adaptively boosted with $Q = 4$ extra trees. There are altogether 29,710 nodes in the forest.

We present the classification accuracy obtained by the SVM over n-gram features, the baseline classifier which combines the SVM with the fuzzy random forest, and the boosted classifier in Table. II as an ablation study. The corresponding confusion matrices are shown in Fig.11.

TABLE II
CLASSIFICATION ACCURACY AT DIFFERENT STAGES OF THE MODEL.

Stage	Accuracy	Number of parameters
SVM(n-gram)	81.4%	0.12M
SVM+fuzzy random forest	85.8%	0.17M
SVM+fuzzy random+adaptive boosting	90.8%	0.19M

It can be observed from Fig. 11a that some classes can be classified well with solely n-gram features. After incorporating the static features (Fig. 11b) and boosting classifiers (Fig. 11c), the overall accuracy is significantly improved. Unlike the decision boundary given by SVM, the fuzzy decision trees



(a) SVM. (b) The baseline classifier. (c) The boosted classifier.

Fig. 11. The confusion matrices obtained by different stages in the classifier. The malware categories are indexed by 1:TrojanDownloader, 2:SoftwareBundler, 3:Virus, 4:Trojan, 5:Ransom, 6:PWS, 7:TrojanSpy, 8:VirTool, 9:PUA, 10:Backdoor.

TABLE III
COMPARISION OF CLASSIFICATION ACCURACY, COMPLEXITY, AND VARIATION.

Model	Ours	Neuro-fuzzy [33]	Artificial neural network	Naive Bayes classifier
Accuracy	**90.8%**	26.5%	80.2%	59.1%
Number of parameters	0.19M	0.078M	1.1M	0.11M
Standard deviation	2.56%	3.68%	3.32%	1.49%

Model	Logistic regression	CNN [23]	Random forest
Accuracy	62.2%	90.7%	73.3%
Number of parameters	0.14M	2.53M	0.15M
Standard deviation	3.02%	1.54%	1.67%

yield a quantitative insight into the data that carries accessible knowledge.

To comprehensively evaluate the proposed approach in malware classification, a number of well-known classifiers that can be easily applied to the extracted features have been investigated, which include non-fuzzy random forest, naive Bayes classifier (using only n-gram features), artificial neural network, logistic regression, and SVM. In addition, we also compared with the state-of-the-art CNN-based malware classifier proposed in [23]. The CNN consists of 9 convolutional layers, 3 pooling layers and 2 densely connected layers. The image size of the visual feature is 448×448. Finally, the neuro-fuzzy approach [33] as an application of fuzzy system for large scale malware classification, is also compared. The results are shown in Table. III.

It can be concluded that our approach significantly outperformed the traditional methods. Compared with traditional machine learning models such as logistic regression, support vector machine or deep learning models, our method based on decision trees can be easily interpreted by human. The decision boundary on the fuzzy features from the SVM, together with the fuzzy rules provided by the fuzzy decision trees yields an accessible set of malware discrimination knowledge (such as the exemplary rule given before,) which is highly interpretable for human.

The CNN model achieved similar accuracy as our method, but with much more parameters. Moreover, considering that the visual representation of malware lacks many properties of real images such as spatial continuity between pixels, rescaling the visual representation to reduce computational complexity becomes infeasible.

As reported in the original paper, the neuro-fuzzy method [33], which is a combination of simple neural network and fuzzy system, performs poorly with this multi-category classification task. Although it decouples the correlation between properties by optimizing the elliptic fuzzy patch [21] and trains 27 rules, it does not consider content-oriented information and fails to finely partition the feature space, therefore its accuracy is relatively low. By properly accounting for statistical variation and noise and linguistic impreciseness of the extracted features using fuzzy theory, our method obtains trees with fewer nodes compared with random forest. Finally, by adopting a novel ensemble framework, we achieved state-of-the-art results for malware classification.

V. CONCLUSION

This paper studied the problem of large-scale malicious software classification. To achieve robust classification performance on large collections of malwares, fuzzy partition is applied onto the extracted static features and fuzzy matching is adopted for the linguistic features. We proposed a composite classifier constructed from an ensemble of support vector machine and fuzzy random forest, adaptively boosted by fuzzy decision trees. Our proposal is efficient, robust, and capable

of yielding interpretable rules for identifying the malware category. Comparison between other well-known classifiers on a dataset with over 200k samples from the the VirusShare dataset showed the significantly superior performance of the proposed framework.

The experimental results shed light on the efficacy of applying fuzzy systems to large-scale malware classification. Our algorithm can achieve good classification with interpretability, an essential element of explainable AI. We are looking forward to the applications of more complex and intelligent fuzzy systems in malware analysis, particularly fuzzy systems that can classify the interaction graph and other dynamic features. In our future work, we will also explore ways to make our classifier more robust against subtle adversarial attacks such as obfuscation and mutation attacks. The eminent threat of cyber-attack means that it is necessary and urgent to study the attack and defense mechanism in AI-based malware detection/classification systems.

REFERENCES

[1] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 3, 2018.

[2] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "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*, 2016, pp. 183–194.

[3] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *computers & security*, vol. 73, pp. 399–410, 2018.

[4] A. Shalaginov and K. Franke, "Automatic rule-mining for malware detection employing neuro-fuzzy approach," *Norsk informasjonssikkerhetskonferanse (NISK)*, vol. 2013, 2013.

[5] E. M. Dovom, A. Azmoodeh, A. Dehghantanha, D. E. Newton, R. M. Parizi, and H. Karimipour, "Fuzzy pattern tree for edge malware detection and categorization in iot," *Journal of Systems Architecture*, vol. 97, pp. 1–7, 2019.

[6] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 134–142.

[7] R. Perdisci, A. Lanzi, and W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," in *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2008, pp. 301–310.

[8] S. Mohurle and M. Patil, "A brief study of wannacry threat: Ransomware attack 2017," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.

[9] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[10] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[11] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[12] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. OReilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[13] C. P. Chen, C.-Y. Zhang, L. Chen, and M. Gan, "Fuzzy restricted boltzmann machine for the enhancement of deep learning," *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 6, pp. 2163–2173, 2015.

[14] K. Shihabudheen and G. N. Pillai, "Recent advances in neuro-fuzzy system: A survey," *Knowledge-Based Systems*, vol. 152, pp. 136–162, 2018.

[15] I. Couso, C. Borgelt, E. Hullermeier, and R. Kruse, "Fuzzy sets in data analysis: from statistical foundations to machine learning," *IEEE Computational Intelligence Magazine*, vol. 14, no. 1, pp. 31–44, 2019.

[16] L. Fang, X. Yun, C. Yin, W. Ding, L. Zhou, Z. Liu, and C. Su, "Ancs: Automatic nxdomain classification system based on incremental fuzzy rough sets machine learning," *IEEE Transactions on Fuzzy Systems*, 2020.

[17] A. S. Shirkhorshidi, T. Y. Wah, S. M. R. Shirkhorshidi, and S. Aghabozorgi, "Evolving fuzzy clustering approach (efca): An epoch clustering that enables heuristic post pruning," *IEEE Transactions on Fuzzy Systems*, 2019.

[18] M.-S. Yang and Y. Nataliani, "A feature-reduction fuzzy clustering algorithm based on feature-weighted entropy," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 2, pp. 817–835, 2017.

[19] Q. Fan, Z. Wang, D. Li, D. Gao, and H. Zha, "Entropy-based fuzzy support vector machine for imbalanced datasets," *Knowledge-Based Systems*, vol. 115, pp. 87–99, 2017.

[20] P. Bonissone, J. M. Cadenas, M. C. Garrido, and R. A. Díaz-Valladares, "A fuzzy random forest," *International Journal of Approximate Reasoning*, vol. 51, no. 7, pp. 729–747, 2010.

[21] A. Shalaginov and K. Franke, "A new method of fuzzy patches construction in neuro-fuzzy for malware detection," in *2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (IFSA-EUSFLAT-15)*. Atlantis Press, 2015.

[22] Y. Zhang, J. Pang, F. Yue, and J. Cui, "Fuzzy neural network for malware detect," in *2010 International Conference on Intelligent System Design and Engineering Application*, vol. 1. IEEE, 2010, pp. 780–783.

[23] Q. Chu, G. Liu, and X. Zhu, "Visualization feature and cnn based homology classification of malicious code," *Chinese Journal of Electronics*, vol. 29, no. 1, pp. 154–160, 2020.

[24] I. Abdessadki and S. Lazaar, "A new classification based model for malicious pe files detection," *International Journal of Computer Network and Information Security*, vol. 11, no. 6, p. 1, 2019.

[25] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, 2018.

[26] S. Pai, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "Clustering for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 2, pp. 95–107, 2017.

[27] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2482–2486.

[28] P. Mishra, K. Khurana, S. Gupta, and M. K. Sharma, "Vmanalyzer: Malware semantic analysis using integrated cnn and bi-directional lstm for detecting vm-level attacks in cloud," in *2019 Twelfth International Conference on Contemporary Computing (IC3)*. IEEE, 2019, pp. 1–6.

[29] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–5.

[30] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé *et al.*, "Deep android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 301–308.

[31] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision tree analysis on j48 algorithm for data mining," *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 6, 2013.

[32] S. Kalmegh and S. Deshmukh, "Categorical identification of indian news using j48 and ridor algorithm," *International Refereed Journal of Engineering and Science (IRJES)*, vol. 3, no. 6, pp. 79–84, 2014.

[33] A. Shalaginov, L. S. Grini, and K. Franke, "Understanding neuro-fuzzy on a class of multinomial malware detection problems," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 684–691.

[34] Y. Liu, J. Zhao, D. Wang, and W. Pedrycz, "Prediction intervals for granular data streams based on evolving type-2 fuzzy granular neural network dynamic ensemble," *IEEE Transactions on Fuzzy Systems*, 2020.

[35] T. Nguyen, I. Hettiarachchi, A. Khosravi, S. M. Salaken, A. Bhatti, and S. Nahavandi, "Multiclass eeg data classification using fuzzy systems," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.

[36] Z. Zhong, C.-M. Lin, Z. Shao, and M. Xu, "Decentralized event-triggered control for large-scale networked fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 1, pp. 29–45, 2016.

[37] Q. Huang, J. Yang, X. Feng, A. W.-C. Liew, and X. Li, "Automated trading point forecasting based on bicluster mining and fuzzy inference," *IEEE Transactions on Fuzzy Systems*, 2019.

[38] T. T. Nguyen, M. P. Nguyen, X. C. Pham, and A. W.-C. Liew, "Heterogeneous classifier ensemble with fuzzy rule-based meta learner," *Information Sciences*, vol. 422, pp. 144–160, 2018.

[39] N. Naik, R. Diao, and Q. Shen, "Dynamic fuzzy rule interpolation and its application to intrusion detection," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 1878–1892, 2017.

[40] C. Z. Janikow, "Fuzzy decision forest," in *22nd International Conference of the North American Fuzzy Information Processing Society, NAFIPS 2003*. IEEE, 2003, pp. 480–483.

[41] C. Marsala and B. Bouchon-Meunier, "Fuzzy partitioning using mathematical morphology in a learning scheme," in *Proceedings of IEEE 5th International Fuzzy Systems*, vol. 2. IEEE, 1996, pp. 1512–1517.

[42] C. Marsala, "Data mining with ensembles of fuzzy decision trees," in *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2009, pp. 348–354.

[43] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android malware detection based on network traffic using decision tree algorithm," in *International Conference on Soft Computing and Data Mining*. Springer, 2018, pp. 485–494.

[44] D. Moon, H. Im, I. Kim, and J. H. Park, "Dtb-ids: an intrusion detection system based on decision tree using behavior analysis for preventing apt attacks," *The Journal of supercomputing*, vol. 73, no. 7, pp. 2881–2895, 2017.

[45] A. Tripathy, A. Agrawal, and S. K. Rath, "Classification of sentiment reviews using n-gram machine learning approach," *Expert Systems with Applications*, vol. 57, pp. 117–126, 2016.

[46] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary physics*, vol. 46, no. 5, pp. 323–351, 2005.

[47] F. Li, D. Miao, and W. Pedrycz, "Granular multi-label feature selection based on mutual information," *Pattern recognition*, vol. 67, pp. 410–423, 2017.

[48] M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer, "Problems with evaluation of word embeddings using word similarity tasks," *arXiv preprint arXiv:1605.02276*, 2016.

[49] C. Olaru and L. Wehenkel, "A complete fuzzy decision tree technique," *Fuzzy sets and systems*, vol. 138, no. 2, pp. 221–254, 2003.

[50] M. Ramdani, "Système d'induction formelle à base de connaissances imprécises," Ph.D. dissertation, Paris 6, 1994.

[51] T. T. Nguyen, T. T. T. Nguyen, X. C. Pham, and A. W.-C. Liew, "A novel combining classifier method based on variational inference," *Pattern Recognition*, vol. 49, pp. 198–212, 2016.

[52] T. T. Nguyen, X. C. Pham, A. W.-C. Liew, and W. Pedrycz, "Aggregation of classifiers: a justifiable information granularity approach," *IEEE transactions on cybernetics*, vol. 49, no. 6, pp. 2168–2177, 2018.

[53] T. T. Nguyen, M. P. Nguyen, X. C. Pham, A. W.-C. Liew, and W. Pedrycz, "Combining heterogeneous classifiers via granular prototypes," *Applied Soft Computing*, vol. 73, pp. 795–815, 2018.

[54] T. T. Nguyen, M. T. Dang, A. W. Liew, and J. C. Bezdek, "A weighted multiple classifier framework based on random projection," *Information Sciences*, vol. 490, pp. 36–58, 2019.

[55] A. J. Wyner, M. Olson, J. Bleich, and D. Mease, "Explaining the success of adaboost and random forests as interpolating classifiers," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1558–1590, 2017.

[56] J. Wang, P. Li, R. Ran, Y. Che, and Y. Zhou, "A short-term photovoltaic power prediction model based on the gradient boost decision tree," *Applied Sciences*, vol. 8, no. 5, p. 689, 2018.