Hindawi
Scientific Programming
Volume 2021, Article ID 2666153, 9 pages
https://doi.org/10.1155/2021/2666153

Research Article

Compressed Firmware Classification Based on Extra Trees and Doc2Vec

Jing Qiu,[1,2] **Xiaoxu Geng**,[1] **and Guanglu Sun**[1]

[1]*Harbin University of Science and Technology, Harbin 150080, China*
[2]*Zhejiang A&F University, Hangzhou 311300, China*

Correspondence should be addressed to Jing Qiu; topmint@gmail.com

Received 1 September 2021; Revised 29 November 2021; Accepted 6 December 2021; Published 22 December 2021

Academic Editor: Shah Nazir

Firmware formats vary from vendor to vendor, making it difficult to track which vendor or device the firmware belongs to, or to identify the firmware used in an embedded device. Current firmware analysis tools mainly distinguish firmware by static signatures in the firmware binary code. However, the extraction of a signature often requires careful analysis by professionals to obtain it and requires a significant investment of time and effort. In this paper, we use Doc2Vec to extract and process the character information in firmware, combine the file size, file entropy, and the arithmetic mean of bytes as firmware features, and implement the firmware classifier by combining the Extra Trees model. The evaluation is performed on 1,190 firmware files from 5 router vendors. The accuracy of the classifier is 97.18%, which is higher than that of current approaches. The results show that the proposed approach is feasible and effective.

1. Introduction

With the continuous development of network infrastructure in recent years, the Internet of Things (IoT) has attracted much attention and the number of IoT devices has increased dramatically. People are gradually aware of the problems in IoT while enjoying the convenience brought by IoT devices. The security of IoT applications is one of the major concerns, and the security of firmware is one of the most important issues. However, it is hard to analyze and detect vulnerabilities in firmware. Different firmware has different instruction sets and corresponding hardware environments. Thus, the hardware type of the target firmware needs to be known before analyzing the firmware. Without knowing the hardware type of the target firmware, there is no way to build an emulation environment for the target firmware. Also, since routers are mostly small devices with limited memory capacity, the firmware usually consists of some files and needs to be compressed to reduce the memory footprint of the device. But this also means that it is much more difficult to extract semantic information from the unpacked firmware through static analysis, making it almost impossible to analyze the firmware statically. Therefore, accurate access to the hardware information of the firmware is a necessary prerequisite for further analysis of the firmware.

Before training the firmware classification model, the acquisition of the dataset (i.e., firmware) is the most important prerequisite work. There are two main ways to obtain the firmware: one is from the router vendor's website and the other is to extract the firmware directly from the hardware device. The latter requires disassembling and further analysis of the hardware to obtain the firmware corresponding to the device, which is less efficient. Therefore, the sources of firmware in this paper are obtained from the web.

After the firmware acquisition, we need to determine the virtual execution environment of the current firmware to perform the dynamic analysis of the firmware. Thus, configuring the execution environment of the firmware is a crucial part, and the most critical information is the architecture information of the firmware. Generally, the architecture information contains the manufacturer information and the operating device information of the current hardware device. The "type information" in this paper refers to the firmware vendor information of the device. Once the above type information is obtained, it is

possible to configure the corresponding virtual execution environment and proceed to the next stage of firmware analysis. The most representative and widely used firmware analysis tool to solve this problem is Binwalk [1], a firmware type analysis tool that analyzes compressed firmware through a signature database. Typically, Binwalk traverses all bytes in a binary file to extract and match signatures, while Binwalk can also split or extract the entire file.

Binwalk analyzes the firmware by matching signatures to the target firmware. Once a signature is found in the signature library, the corresponding analysis rules are applied to analyze the target firmware and obtain the unpacked file for further analysis of the firmware. However, Binwalk is not always able to display the firmware information well and may sometimes fail to obtain the structure information corresponding to the firmware file. For example, in the example in Figure 1, when using Binwalk to analyze Tenda's firmware named 4G300_V1.01.30, the structure information corresponding to the firmware file cannot be obtained.

The reason for the above problem is that Binwalk uses signatures to match firmware. If the signature of the target firmware does not exist in the signature database, then Binwalk does not work well. Also, Binwalk cannot update its signature database timely, which makes it difficult to analyze some new firmware.

This paper addresses the above issue by using machine learning to classify unknown firmware, compensating for the limitation of matching and analyzing firmware by signature only. A firmware classification model combining Extra Trees and Doc2Vec is proposed to improve the accuracy of firmware classification. Also, it can have a high classification accuracy for unknown firmware.

The main contributions are as follows:

(1) We propose a firmware classification model combining Extra Trees and natural language processing technology to improve the accuracy of firmware classification. It can also have high classification accuracy for firmware that is not recorded in the firmware signature database.

(2) We implement the prototype and evaluate it on 1,190 router firmware. The accuracy of the proposed model is 97.18%. We open-source our implementation at https://github.com/qiujing/firmware.

The rest of this paper is arranged as follows. Section 2 gives the details of the proposed solution. Section 3 shows the evaluation result and discussion. Section 4 introduces the related work. Section 5 concludes the paper.

2. Solution

In this paper, ten candidate models are mainly selected to classify firmware. They are Decision Tree, Random Forest, XGBoost, Support Vector Machine (SVM), Extra Trees, Decision Tree + Doc2Vec, Random Forest + Doc2Vec, XGBoost + Doc2Vec, SVM + Doc2Vec, and Extra Trees + Doc2Vec.

```
qiujing: ~/firmware/MERCURY$ binwalk –A isp1_2. bin
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
qiujing: ~/firmware/MERCURY$ |
```

Figure 1: Binwalk fails to identify Mercury isp1_2.bin.

2.1. Model Selection. The main reasons for choosing tree classifier as firmware classification in this paper are shown below.

From the perspective of dataset, there are few large public datasets available due to the small number of publicly available firmware on the market. Current deep learning methods are widely used for their excellent accuracy, and their classification results are generally better than those of tree classifiers. However, this phenomenon is predicated on having a large dataset to train the model. Deep learning can only achieve better results when it is sufficiently trained, and the amount of firmware that can be collected at present clearly does not meet this requirement. Most tree classifiers, on the other hand, require only a small amount of training data to achieve good results. Thus, tree classifiers are more suitable for firmware classification.

In terms of the classification methods, a tree classifier is more like a stepwise filtering of features. This makes a tree classifier relatively less computationally intensive and easy to translate into classification rules. As long as the tree roots go down to the leaves, the splitting conditions along the way can uniquely determine a classification result. Secondly, the classification rules mined by a tree classifier are highly accurate and easy to understand. A decision tree can clearly show which fields are more important, and it is easy to understand the classification basis of the model intuitively; that is, it can generate rules that can be understood. Thirdly, a tree classifier does not require any domain knowledge and parameter assumptions, which can be more easily brought into various application scenarios. Furthermore, the structure of a tree classifier itself is more suitable for multiple classifications. SVM can only achieve multiclassification by combining multiple binary classifications. Extra Trees/Random Forest, a forest classifier, is a good remedy for the limitations of a single decision tree. Each tree of the forest class has its own category judgment rule. Thus, it can analyze a set of data from multiple perspectives and finally decide the final category by the voting method, which greatly improves the accuracy.

In terms of the training/recognition time, the training time of a tree classifier is short. This can also be seen by the experimental results, where the training time of each classifier is within a good range. Since a tree classifier determines the classification rules after the training is completed, only the input feature values need to be brought into the classification rules layer by layer during prediction. This structure greatly improves the recognition speed.

2.2. Model Introduction. The decision tree algorithm is a relatively common machine learning method. A decision tree is a tree structure in which each nonleaf node within the tree represents a judgment of an attribute, and the leaf nodes

represent the category of its final classification. Such a tree structure as decision tree is extremely suitable for firmware vendor classification. Compared to deep learning, using decision tree only requires less training data to achieve a better classification effect, which also solves the problem that firmware datasets are less resourceful and difficult to obtain.

XGBoost, which stands for Extreme Gradient Boosting, is an improvement and extension of the gradient-boosted decision tree algorithm, which has a faster computing speed and higher accuracy [2]. XGBoost draws on the advantages of random forests and supports feature sampling to prevent overfitting of the final classification results, while also reducing computational effort. XGBoost also supports parallelization, which greatly reduces the time of model operations.

Random Forest is a modification of decision tree [3]. Its main idea is to integrate multiple decision trees through the idea of integrated learning. Each tree in a random forest is a classifier. The final classification result is determined by the vote of all the decision trees. Each tree in the random forest is randomly sampled from the training set, which ensures that each tree is a nonrepeating decision tree. If random sampling is not used, the entire random forest will have the same result as one decision tree. Similarly, random forests can achieve better classification results with fewer datasets.

Extra Trees [4] is very similar to Random Forest, which is composed of many decision trees. The differences between them are as follows.

(1) For the training set of each decision tree, Random Forest uses a random sampling bootstrap to select the sampling set as the training set of each decision tree, while Extra Trees generally does not use random sampling; that is, each decision tree uses the original training set. Random Forest applies the bagging model, and Extra Trees uses all the samples; only the features are randomly selected. Because the splitting is random, the results are somehow better than those obtained by Random Forest.

(2) After selecting the splitting features, the Random Forest decision tree will select an optimal feature value based on the principles of information gain, Gini coefficient, mean squared deviation, and so on to divide the points, which is the same as the traditional decision tree. However, extra tree is more aggressive and randomly selects a feature value to divide the tree. This results in a decision tree that is generally larger than the one generated by Random Forest because the random selection of the eigenvalue division point is not the optimal point. That is, the variance of the model is further reduced relative to Random Forest, but the bias is further increased relative to Random Forest. In some cases, the generalization ability of Extra Trees is better than Random Forest.

The basic idea of a support vector machine (SVM) is to find the separating hyperplane with the largest geometric separation from each class, while ensuring the correct classification of the training data. For multiclassification problems, the main idea of SVM classifier is to combine multiple binary classifiers to achieve the construction of multiclassifiers. The most common construction method is "one-against-one" (one-against-one). This method requires two combinations of classes of training data to build $n(n-1)/2$ SVMs, each of which is trained with two different classes of data, and the final classification result is decided by "voting."

In the early days of natural language, the main choice for data processing was to use One-hot Representation as the most common word representation method. But this method has some drawbacks. The words represented by the one-hot method are isolated from each other and lack the connection between words. Besides, as the words in the word list increase, the dimensionality of the matrix also rises, and a dimensional disaster occurs. This requires a word vector representation that can represent both the word itself and the semantic connection between words, which is the emergence of Word2Vec [5]. But Word2Vec is only based on the dimensionality of words for semantic analysis, ignoring the connection between contexts.

The Doc2Vec algorithm can handle variable-length text data [6]. It can infer the connection between sentences and sentences and possesses the ability to analyze the semantics of context. Two methods also exist in Doc2Vec: distributed memory and distributed bag of words. Distributed memory is to predict the probability of a word given the context and the paragraph vector. Distributed bag of words is to predict the probability of a set of random words in a paragraph given the paragraph vector. In this paper, the main function of Doc2Vec is to represent the string in binary code as a vector form. Compared to Word2Vec, Doc2Vec has a better understanding of the semantic content of the whole sentence, which makes the features used in subsequent analysis have a better classification effect.

2.3. Feature Selection. In this paper, the following properties of a firmware are used as features.

(1) File size: the size of a firmware file is directly related to the hardware design and function design. The more complex the function is, the larger the corresponding firmware will be and vice versa. Meanwhile, the size of a firmware will not exceed the memory limit of the corresponding embedded device. Therefore, the size of a firmware file can well reflect the corresponding device model and manufacturer information.

(2) File entropy: since there has not been a standardized form requirement for writing firmware and there are some differences in firmware structures among major firmware vendors, the information contained in each vendor's firmware has a specific distribution and density. Therefore, the knowledge of information theory is introduced here, and entropy values are used as a feature to distinguish vendors. The

Shannon entropy formula is used to calculate the entropy value of each byte. The entropy value is taken in the range of 0 to 8. The larger the value is, the higher the compression rate of the whole file and the denser the information is. It also means that the distribution of bytes in the current firmware is highly random.

(3) Arithmetic mean of bytes: this value is computed by summing up all the byte values and then calculating the average value.

(4) Strings in firmware: visible characters could exist in a firmware. Currently, if a byte is a digital number or an English letter, it is called a visible character. Adjacent visible characters could be regarded as a string, and these strings are used as a feature of a firmware. Figure 2 shows examples of strings that exist in firmware.

2.4. Extra Trees with Doc2Vec. In this paper, a firmware classifier is proposed by combining the techniques of both natural language processing and Extra Trees. This combination will produce an algorithm with better accuracy as well as applicability than the current common algorithms (such as Decision Tree, Random Forest, and XGBoost) for firmware classification.

Figure 3 shows the main structure of the whole firmware classifier. The construction of the dataset can be divided into two parts. The first part is for the processing of strings in the firmware. The second step is to build the corpus. It consists of all words in the strings that are extracted from a firmware file. Words of a string are extracted by a segmentation algorithm. Then the extracted words of a firmware file are passed into the trained Doc2Vec model to obtain a vector representation. Finally, the vector representation is combined with other features inputting to the classifier.

3. Evaluation

3.1. Setup. All models in this experiment were evaluated on the same dataset and conducted on a computer running Windows 10 operating system. This computer is equipped with 16 GB memory, 512 GB Samsung 981 SSD, and Intel Core i5-8300H processor. The prototype is implemented with Python 3.7, Scikit-learn 0.23.2, Gensim 3.8.3, Pandas 1.3.4, and NumPy 1.18.5.

The datasets in the experiment are sourced from the official websites of major manufacturers, so all sources are regular and secure, ensuring the accuracy and authenticity of this classification experiment. A total of 1,190 firmware files were collected from 5 different companies as the dataset in this experiment. All firmware files are not duplicated, while some of them with large similarity in versions were eliminated. Table 1 shows the detail of firmware files in each category.

The frameworks used in this experiment are Scikit-learn and Gensim. Scikit-learn is mainly applied to three classical machine learning models, and Gensim is mainly applied to Doc2Vec. Table 2 lists the detailed parameter settings for each model. All parameters not given in Table 2 use the default parameters of frameworks.

The fourth feature, the Doc2Vec-processed document vector, is used to verify the impact on the classification results. Since the number of firmware from each vendor in the dataset is uneven, the dataset is created by selecting firmware from each category by percentage as training samples. A portion of the data set is taken as the training set, starting from 10% and going up to 90%. For each training set extraction, a random sampling method is used to select the firmware. For each percentage of the training set results, the average classification accuracy is calculated using the results of 50 experimental runs as the final results.

3.2. Result. Strings of a firmware that are too short may not be helpful for recognition. Different experimental results were obtained by limiting the minimum length of words separately (Figure 4). For Random Forest, Decision Tree, XGBoost, SVM, and Extra Trees, it can be seen that the minimum word lengths for which they have the best accuracy are 3, 6, 6, 2, and 4, respectively. For these classifiers, these values are also taken in the later experiments. Figure 5 shows how the accuracy of these three classifiers with and without string features varies with the size of the training set. Overall, as the training set increases, the accuracy increases.

The best classification results for each classifier are given in Table 3. As a comparison, the results of Binwalk are also listed in the table. From Table 3, it can be seen that Extra Trees is the best classifier. Table 4 gives the classification results of Extra Trees for each vendor firmware. Table 5 gives the specific classification results of Extra Trees on the test set. Figure 6 shows the time spent by models.

3.3. Discussion

3.3.1. Accuracy. Figure 5 shows three facts. First, on the whole, the accuracy of classifiers with Doc2Vec are higher than that of them without Doc2Vec. Therefore, the introduction of string features has a positive impact on these classifiers and can significantly improve their accuracy.

Second, the accuracy of each classifier increases gradually with the increase of the number of training sets. In particular, the accuracy of all three sets of classifiers improves dramatically as the percentage of training sets increases from 10% to 75%. This experiment uses a dataset consisting of 1,190 firmware, which has a higher accuracy compared to the classifier trained from a few firmware files [7]. If more datasets are collected later, it can be optimistically estimated that the classifier will have better classification performance.

Third, the accuracy of both Extra Trees and Random Forest is better than that of XGBoost. Random Forest is slightly less effective than Extra Trees. Thus, the best firmware classifier is Extra Trees + Doc2Vec.

Table 4 shows that Extra Trees + Doc2Vec has excellent classification results in most of the vendors, but the accuracy for Mercury firmware is slightly lower than other vendors.

HiWiFi:
HiWiFi R34 Sysupgrade image images boot openwrt ipq40xx R34 boot stripped elf 00000002 ...

Mercury:
Rar MW54R V1 071219 mw54rpv1 up bin PH Tc Da 0a Wl1K ...

MiWiFi:
HDR1 xiaoqiang version config core version ROM ver option ROM 10 14 channel option CHANNEL ...

Tenda:
PK US 4G300mt V1 01 38 TD BIN Vg Tw Linux Kernel Image r9aQ …

TP-LINK:
PK QL4K TL IPC333K V1 20170904 up yTL IPC333K V1 20170904 PK JW ...

Figure 2: Strings in firmware.

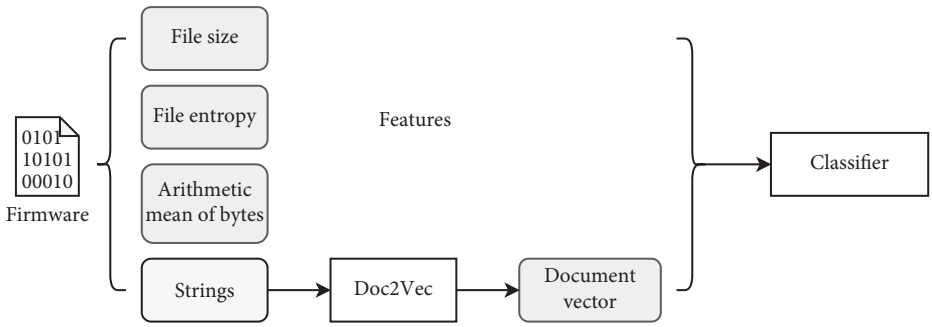

Figure 3: Overview of the proposed model.

Table 1: Dataset.

Manufacturer	#of firmware	File size (GB)
HiWiFi	30	0.34
Mercury	190	0.61
MiWiFi	31	0.68
Tenda	273	1.17
TP-LINK	666	6.65
Total	1,190	9.45

Table 2: Model parameters.

Model	Framework	Parameters
Decision Tree	Scikit-learn	criterion = "entropy," random_state = 0
Random Forest	Scikit-learn	criterion = "entropy," min_samples_leaf = 2, class_weight = "balanced"
XGBoost	Scikit-learn	random_state = 27
Extra Trees	Scikit-learn	Default
SVM	Scikit-learn	Default
Doc2Vec	Gensim	min_count = 1, window = 5, size = 10, sample = 1e-3, negative = 5, workers = 4

This may be due to the similarity of Mercury's classification features with other vendors. For HiWiFi and MiWiFi, which have only a small dataset, they have better classification results in this experiment. Especially, this paper greatly improves the accuracy of MiWiFi compared with the work of Lee S [7] where their accuracy is only 66% for MiWiFi. With the evaluation metrics such as Recall and F1-score, the accuracy of the classifier proposed in this paper for MiWiFi possesses no less than other vendors, indicating that the Extra Trees + Doc2Vec classifier proposed in this paper possesses a good classification performance.

3.3.2. Time. Figure 6(a) shows that the training time of most of the models gradually increases as the percentage of the training set keeps increasing. This phenomenon is particularly evident in XGBoost, and it can be seen that the length of the features also affects the training time of a model to a

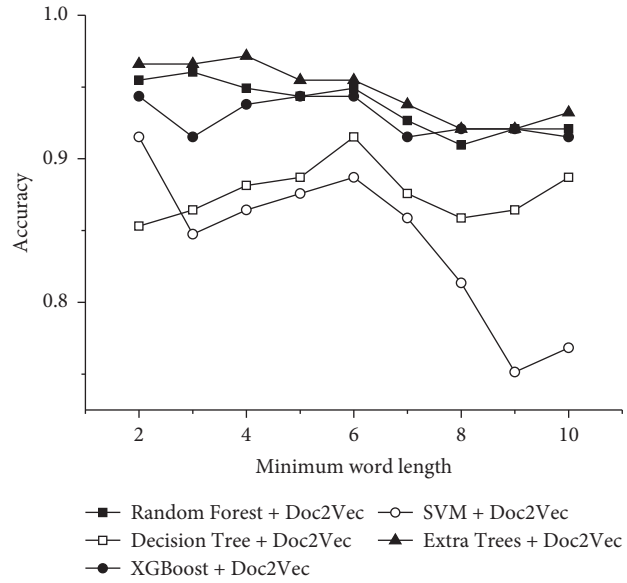

Figure 4: Accuracy of classifiers with different minimum word lengths.

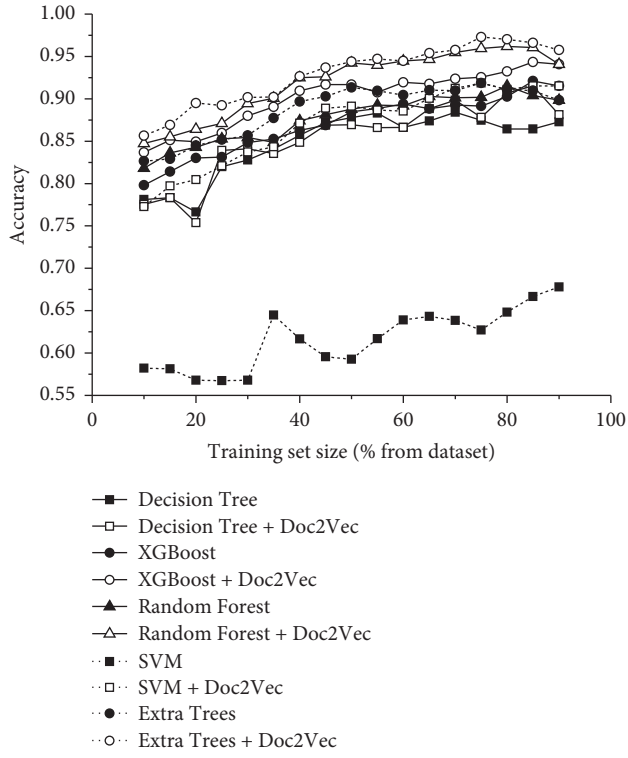

Figure 5: Accuracy of classifiers with and without Doc2Vec.

certain extent. The longer the features, the longer the training time of a model. The difference between the remaining four models is not obvious in Figure 6(a), except for XGBoost, which requires a longer training time. This may be due to the structure of the models themselves. The classification process of the tree classifier is similar to the feature selection process, in which the final category

Table 3: Evaluation result.

Model	Precision	Recall	F1-score
Random Forest + Doc2Vec	0.9670	0.9661	0.9648
Decision Tree + Doc2Vec	0.9180	0.9153	0.9162
XGBoost + Doc2Vec	0.9440	0.9435	0.9423
Extra Trees + Doc2Vec	**0.9718**	**0.9718**	**0.9713**
SVM + Doc2Vec	0.9168	0.9153	0.9143
Binwalk	0.9261	0.9261	0.9261

The bold values show the largest value in the corresponding column. The higher the value in each column, the better.

Table 4: Evaluation result of Extra Trees + Doc2Vec.

Label	Precision	Recall	F1-score
HiWiFi	1.0000	1.0000	1.0000
Mercury	0.9524	0.9091	0.9302
MiWiFi	1.0000	0.8000	0.8889
Tenda	0.9706	1.0000	0.9851
TP-LINK	0.9730	0.9818	0.9774

Table 5: Classification result of Extra Trees + Doc2Vec on the test set.

Label	Predict					
	HiWiFi	Mercury	MiWiFi	Tenda	TP-LINK	Total
HiWiFi	7	0	0	0	0	7
Mercury	0	20	0	0	2	22
MiWiFi	0	0	4	0	1	5
Tenda	0	0	0	33	0	33
TP-LINK	0	1	0	1	108	110

determination is achieved by gradually filtering certain feature values. The SVM multiclassifier, on the other hand, combines multiple binary classifiers to form the final linear classifier. The structure of these two classifiers determines that they do not require much training data to obtain a good classifier, and the internal structure of the classifier is fixed after the training is completed, so that the category determination can be done quickly by judging the values of each feature value step by step with fixed rules.

Figure 6(b) shows that the classification time of a model is extremely short, and even the longest time of XGBoost is within the acceptable range. The reason for the long judgment time of XGBoost compared with other models is that the internal structure of XGBoost is more complicated. This is because the decision tree only needs to train a tree classifier, and the SVM only needs to train a few binary SVMs to achieve the effect of multiple classifications. In contrast, Random Forest, Extra Trees, and XGBoost need to train hundreds or even more tree classifiers to achieve the final firmware classifier, which is the fundamental reason for the relatively long time. The experimental results of the above two figures show that Extra Trees can achieve a more desirable classification result in a shorter training time, and the combined classification results show that Extra Trees is clearly superior to other classifiers.

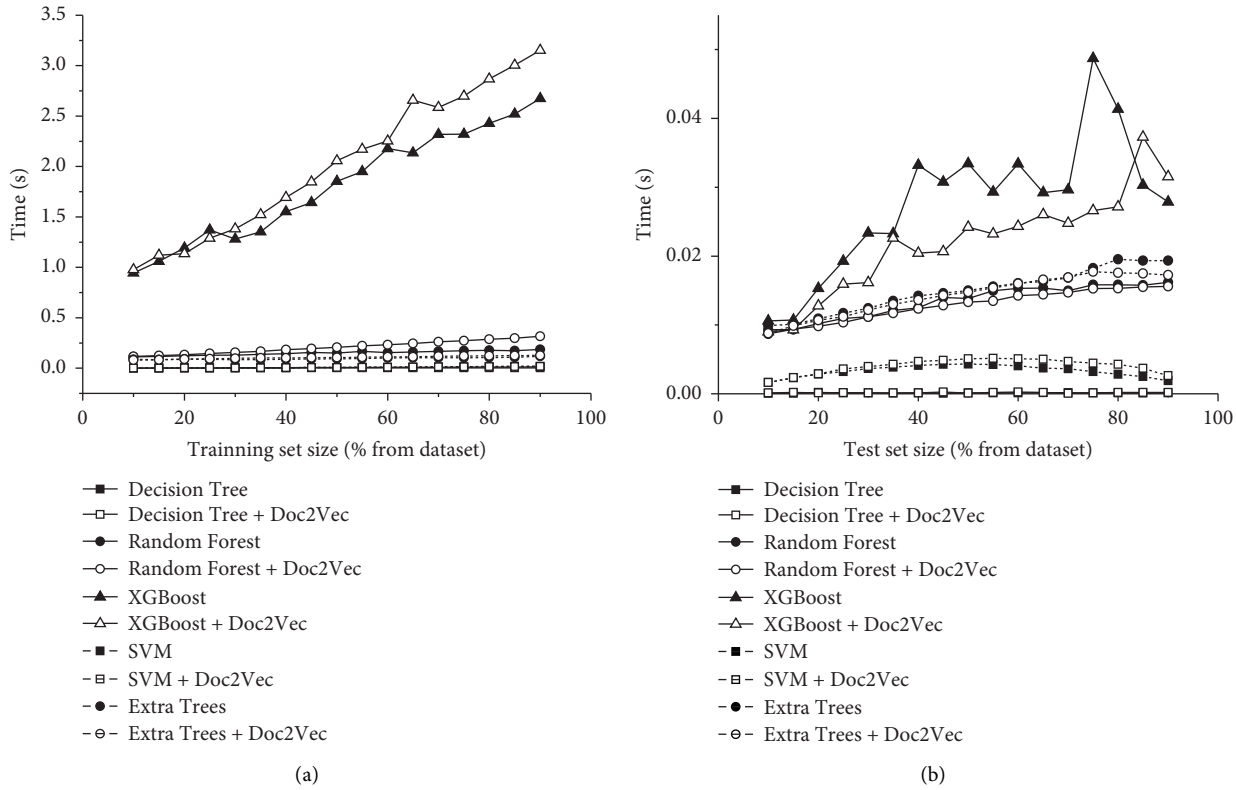

Figure 6: Training/testing time of classifiers with the size of the training/testing set. (a) Training time. (b) Testing time.

3.3.3. Error Classification. The error classification is mainly distributed among three types of firmware in Mercury, Tenda, and TP-LINK (Table 5). The highest percentage of these errors is the mutual classification error between Mercury and TP-LINK. Mercury is a sub-brand of TP-LINK. Some firmware of Mercury is based on improvements made to a particular version of TP-LINK firmware, which leads to misclassification of the classifier (Figure 7). Also, the extracted strings are very similar to that of TP-LINK. It makes the classifier unable to fully learn the string features.

Another example of error classification is shown in Table 6, where the first three items are correctly classified data and the last one is the misclassified data. Compared to the three correctly classified samples above, the first feature of the misclassified sample, that is, the file size, is clearly different from the first three samples. Thus, it is inferred that the cause of the misclassification of this sample is mainly the obvious difference in file size.

To further verify this inference, the case of TP-LINK being incorrectly classified as Mercury was sought in the error case, as shown in Table 7. Firmware files of TP-LINK are generally quite large. The file size of the bolded sample in Table 7 is significantly smaller than the file sizes of the other samples, being 8.71% of the first one and 8.67% of the second one. Thus, it leads to being misclassified. There are specific models of routers that need to add or remove some specific features, causing such firmware files to be too large or too small compared to similar firmware.

4. Related Work

With the widespread use of IoT devices, their security is also a growing concern [8, 9]. Firmware analysis is an important way to analyze the security of IoT devices, and firmware classification is a fundamental work for firmware analysis. It could be done on compressed and unpacked firmware images, the web system running on a device, or by side-channel power analysis [10].

4.1. Firmware Classification without Unpacking. Mkhativari et al. identified firmware supply vendors by using K-means clustering and unsupervised spectral clustering [11], and they used 7, 819 uncompressed firmware for detection, ultimately achieving over 90% accuracy.

Costin A et al. addressed the firmware classification problem by using machine learning and embedded web interface fingerprinting [12]. The firmware images are first distinguished from other types of files, and then the firmware images are classified by the vendor or device type. The main classifier used in this paper is random forest. It is trained through a total of 215 firmware to achieve about 93% accuracy, but its accuracy in a broader dataset is not clear due to the small amount of data.

Lee S et al. proposed to build a classifier using a combination of neural network and SVM [7] and collected 480 datasets from 7 companies. The best classification result for merchants as classification labels reached 91.7%. However, the classification results for some firmware, such as the

```
mr808v2.bin:
Phk Copyright 2005 TP LINK TECHNOLOGIES PU PU 5B P1r 6b yM VxWorks VxWorks5 Oct 20 2008 15 ...

mr824v2.bin:
0I Copyright 2005 TP LINK TECHNOLOGIES PU PU 5B P1r 6b yM VxWorks VxWorks5 Oct 21 2008 14 ...

mr816v1.bin:
81 Copyright 2005 TP LINK TECHNOLOGIES PU PU 5B P1r 6b xM VxWorks VxWorks5 Oct 20 2008 16 ...
```

Figure 7: Some Mercury firmware has the same strings as in the TP-LINK firmware leading to misclassifications.

Table 6: Example of error classification where the file size of a firmware file is too large compared to the others of the same vendor.

File size	File entropy	Mean of bytes	Document vector	Label
1,631,139	0.99998	127.63984	[2.15851, −0.35126, ...]	Mercury
1,601,440	0.99998	127.57772	[−0.69567, 1.79206, ...]	Mercury
1,601,591	0.99998	127.51732	[0.08802, 1.29891, ...]	Mercury
2,327,890	0.99999	127.43466	[2.48045, −1.90191, ...]	Mercury

Table 7: Example of error classification where the file size of a firmware file is too small compared to the others of the same vendor.

File size	File entropy	Mean of bytes	Document vector	Label
11,914,969	0.99999	127.50059	[1.37727, 1.01095, ...]	TP-LINK
11,980,813	0.99999	127.50492	[1.87083, 1.52759, ...]	TP-LINK
1,038,373	0.99997	127.71125	[1.85315, 1.06753, ...]	TP-LINK

firmware of Xiaomi routers, were not very good. It may be due to the small number of their datasets and the problem of feature extraction and selection.

These works have good experimental results for their datasets. But these classifiers lack applicability and cannot be applied to a wider range of firmware classifications. Also, the difficulty of dataset collection can be seen from the side by the limited number of datasets in the early papers, which cannot make a larger and more diverse prediction.

4.2. Firmware Classification after Unpacking. There have been several works on firmware classification and analysis [13–15], where they extract binary images from firmware images to analyze and eventually classify the firmware. They usually used Binwalk [1] to unpack and collect firmware as the training set, or only for a specific type or vendor of firmware.

Qian et al. proposed a control flow graph-based bug search engine for firmware images [13]. Control flow graphs are converted into high-level numeric feature vectors which are robust to code variation across different architectures.

Zhang et al. proposed a firmware information extractor based on graph neural networks [16]. For a firmware, its directories or files are taken as graph nodes, and the relationship between nodes is considered as edges. The experimental results show that it outperforms random forest for the manufacturer, device type, device model, and firmware version recognition.

4.3. Firmware Classification by Web Footprinting. Dan Yu et al. proposed a firmware identification method by analyzing web pages content [17]. The features of the login page of a device is extracted to identify the device type and brand and then use classification and page segmentation to identify the model and firmware version. ARGUS is a simple and practical framework to identify device models and firmware versions [18]. It uses a heuristic fingerprint scheme and improves efficiency by an average of 156 times compared to scanning fingerprints of all web files by default.

Web footprinting can be used to improve the accuracy and time efficiency of identification. However, it can only be applied to running devices.

5. Conclusion and Future Work

It is hard to classify compressed firmware because there is no uniform format for firmware. In this paper, we use the combined model of Doc2Vec and Extra Trees to complete the firmware classification, which makes up for Binwalk's shortage of matching firmware by signatures. The combined model is able with greater accuracy to classify the firmware not yet collected by Binwalk signature library. Natural language processing techniques are used to process the features in firmware. Through experimental validation, the document vectors processed by Doc2Vec as features can significantly improve the accuracy of a classifier, and the final accuracy of 97.18% is achieved by combining Extra Trees.

In the future, we will collect more router firmware and apply the proposed method to a broader classification of firmware, including surveillance devices, smart homes, etc. Moreover, with the expansion of the dataset, deep learning methods can be used for firmware classification.

Data Availability

The firmware data used to support the findings of this study have been deposited in the GitHub repository (https://github.com/qiujing/firmware).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (61702140), Heilongjiang Provincial Natural Science Foundation of China (F2018017), the Fundamental Research Foundation for Universities of Heilongjiang Province (LGYC2018JC015), and Zhejiang A&F University Research Development Fund Talent Initiation Project (2021LFR048).

References

[1] H. Craig, *Binwalk: Firmware Analysis Tool*, p. 4, 2021, https://github.com/ReFirmLabs/binwalk.

[2] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, New York, NY, USA, August 2016.

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of the 1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, Scottsdale, AZ, USA, Jan 2013, https://iclr.cc/archive/2013/workshop-proceedings.html.

[6] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning*, May 2014.

[7] S. Lee, J.. Y. Paik, R. Jin, and E.. S. Cho, "Toward machine learning based analyses on compressed firmware," in *Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 586–591, IEEE, Milwaukee, WI, USA, Jul 2019.

[8] B. Cruz, S. Gomez-Meire, D. Ruano-Ordás, H. Janicke, I. Yevseyeva, and J. R. Méndez, "A practical approach to protect iot devices against attacks and compile security incident datasets," *Scientific Programming*, vol. 2019, Article ID 9067512, 11 pages, 2019.

[9] C. Wright, W. A. Moeglein, S. Bagchi, M. Kulkarni, A. Abraham, and Clements, "Challenges in firmware rehosting, emulation, and analysis," *ACM Computing Surveys*, vol. 54, no. 1, 2021.

[10] D. Krishnankutty, R. Ryan, N. Banerjee, and C. Patel, "Fiscal: firmware identification using side-channel power analysis," in *Proceedings of the 2017 IEEE 35th VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, Las Vegas, NV, USA, Apr 2017.

[11] N. Mkhativari, *Towards Big Scale Firmware Analysis*, INF/01 INFORMATICA, Norfolk Island, 2018.

[12] A. Costin, A. Zarras, and A. Francillon, "Towards automated classification of firmware images and identification of embedded devices," in *Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 233–247, Springer, US, 2017.

[13] F. Qian, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 480–491, ACM, New York, NY, United States, Aug 2016.

[14] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for linux-based embedded firmware," in *Proceedings of the 2016 Network and Distributed System Security Symposium*, Jan 2016.

[15] A. Costin, Z. Jonas, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proceedings of the 23rd USENIX Security Symposium*, pp. 95–110, ACM, San Diego, CA, United states, Aug 2014.

[16] W. Zhang, Li Hong, H. Wen, H. Zhu, and L. Sun, "A graph neural network based efficient firmware information extraction method for iot devices," in *Proceedings of the 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, IEEE, Orlando, FL, USA, May 2018.

[17] D. Yu, L. Zhang, Y. Chen, Y. Ma, and J. Chen, "Large-scale iot devices firmware identification based on weak password," *IEEE Access*, vol. 8, pp. 7981–7992, 2020.

[18] W. Xie, C. Zhang, P. Wang, Z. Wang, and Q. Yang, "Argus: assessing unpatched vulnerable devices on the internet via efficient firmware recognition," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ASIA CCS '21*, pp. 421–431, Association for Computing Machinery, New York, NY, USA, May 2021.