RF-AdaCost: WebShell Detection Method that Combines Statistical Features and Opcode

Wenzhuang Kang[1](✉), Shangping Zhong[1], Kaizhi Chen[1], Jianhua Lai[2], and Guangquan Xu[3]

[1] College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350100, Fujian, China
kangkangajax@163.com, {spzhong,ckz}@fzu.edu.cn

[2] Fujian Institute of Scientific and Technological Information, Fuzhou 350003, Fujian, China
laijh@heidun.net

[3] Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
losin@tju.edu.cn

Abstract. WebShell is called a webpage backdoor. After hackers invade a website, they usually mix backdoor files with normal webpage files in the WEB directory of the website service area. Then, they use a browser to access the backdoor and obtain a command execution environment to control the website server. WebShell detection methods have stringent requirements because of the flexibility of the PHP language and the increasing number of hidden techniques used by hackers. The term frequency–inverse document frequency (TF-IDF) used in the existing random forest–gradient boosting decision tree (RF-GBDT) algorithm does not consider the distribution information and classification capabilities of feature words among classes, and no balance exists between false negative and false positive rates. This work proposes a PHP WebShell detection model called RF-AdaCost, which stands for random forest–misclassification cost-sensitive AdaBoost, based on RF-GBDT. We used the statistical characteristics of PHP source files, including information entropy and index of coincidence, and extracted the opcode sequences of PHP source files, thus merging statistical features and opcode sequences to improve the detection efficiency of the WebShell. Experimental results show that the RF-AdaCost algorithm demonstrates better performance than the RF-GBDT algorithm.

Keywords: WebShell detection · RF-GBDT · TF-IDF · Statistical features · Opcode sequence · RF-AdaCost

1 Introduction

WebShell is essentially a script file written in various languages, such as ASP, JSP, and PHP. It resides in the publicly accessible directory of a web server, and attackers can directly access it by using a browser or customized client software. WebShell provides remote access to various key functions, such as executing arbitrary commands, traversing file directories, viewing and modifying arbitrary files, enhancing access permissions, sending spam, and phishing emails [1]. Statistics [2] show that PHP accounts for 78.4% of web server programming languages. This study focuses on PHP types, and the proposed detection method can be extended to other WebShell types.

By detecting traffic, listening to network traffic to extract WebShell or upload or execute http requests, and paying attention to payload characteristics, WebShell detection cannot establish potential dangers in advance. In addition, extremely heavy traffic affects server performance. Log detection involves detecting WebShell by analyzing whether the page request and response characteristics recorded in the web log contain relevant malicious behavior after the file is executed. However, the attack has already occurred at this time; hence, this type of detection is suitable for use as an auxiliary detection method. With regard to file detection, due to the flexibility of the PHP language, various detection tools can be fooled through special transformation methods to avoid detection and killing. Currently popular WebShell killing systems are implemented through the principle of rule matching or a single statistical principle. Each time a new WebShell appears, these systems need to be updated after a certain period to achieve killing. A zero-day WebShell does not achieve good results. Regular expressions of WebShells are widely used in web application firewall on the border of the web server, but regular expressions are easily confused and bypassed [3] and can only detect known WebShell. Current research generally focuses on the extraction of WebShell statistics (information entropy, coincidence index, longest word, file compression ratio, and dangerous function) or n-gram segmentation of the source file extraction opcode sequence to extract term frequency–inverse document frequency (TF-IDF) features. Current algorithms rarely use the two together. In addition, research on logs and traffic cannot fundamentally ensure that no WebShell script exists on the server, that is, no backdoor file is present after the attack on the server.

Cui et al. [4] proposed the random forest–gradient boosting decision tree (RF-GBDT) algorithm, trained the model by using TF-IDF [5] and HASH vectors, predicted the dataset, used the prediction results as features, and combined the statistical features to obtain a model that uses integrated learning. This method is more accurate than models trained with statistical features or opcodes alone. The TF-IDF used in the algorithm does not consider the distribution information and classification capability of feature words among classes [6], which affect model performance. Moreover, the model does not consider the cost of WebShell underreporting in practical applications. Hence, reducing the underreporting rate as much as possible under the condition that the false alarm rate is acceptable is worth studying.

With reference to RF-GBDT, this study proposes a WebShell detection method based on RF-AdaCost. First, an opcode is extracted from a WebShell file, and n-gram segmentation is performed [7]. Second, the TFIDF–chi feature is extracted [8], and RF [9] is used for training. Third, the predicted results are utilized as a feature after obtaining the model. Lastly, combined with the statistical characteristics of the WebShell, AdaCost [10] (misclassification cost-sensitive AdaBoost) is used for training to obtain the final model.

The main contributions of this work include the following:

(1) An improved TFIDF–chi feature is used to extract WebShell opcode features;
(2) AdaCost is adopted to reduce the underreporting rate and verify the effectiveness of the proposed model.

The rest of the paper is organized as follows. Related technologies are discussed in the second section. The algorithm and framework used in this study and comparative tests are presented in the third section. The experimental methods and results are given in the fourth section. The final section concludes the study and discusses future research directions.

2 Related Work

Hu et al.'s study [11] on WebShell detection methods based on Bayesian theory has similarities with our study, such as the use of common statistical characteristics (e.g., information entropy, coincidence index, and compression ratio). However, this previous study has certain disadvantages. First, the researchers used only 600 samples. The samples were too few to guarantee the generalization performance of the experiment. Second, the features the researchers selected could not clearly reflect the PHP WebShell, that is, the features were also suitable for detecting other WebShell, such as JSP or ASP. Our method is different. To ensure the reliability of the experiment, we collected 5000 samples for training and extracted the characteristics of the PHP file opcode sequence to reflect the characteristics of the PHP file effectively.

The text vector-based PHP WebShell detection method proposed by Zhang et al. [12] uses n-gram and TF-IDF algorithms to convert the opcode sequence generated during the execution of the PHP script into a text vector, with the text vector as the input feature. The researchers used the limit gradient lifting algorithm XGBoost [13] to classify the PHP script through the judgment of the classification results to achieve WebShell detection. The experimental results showed that the proposed method can effectively detect the WebShell and improve the accuracy of WebShell detection. However, this detection model has disadvantages. The TF-IDF used in the algorithm does not consider the distribution information and classification capability of feature words among classes, which affect model performance. In addition, the statistical characteristics of the utilized WebShell are disregarded. The experimental results showed that the added statistical characteristics are improved.

Tu et al. [14] proposed a WebShell recognition method based on the optimal threshold of malicious signatures, malicious function samples, and longest file characters. By scanning and searching for malicious code in each file of the web application, the administrator provides a list of suspicious files and a detailed log analysis table for each suspicious file for further inspection. In view of the shortcomings of traditional machine learning algorithm-based detection models, Yan et al. [15] proposed the application of convolutional neural networks [16] to WebShell detection. Deep learning models do not require complex artificial feature engineering. Model features trained through model learning can enable attackers to avoid targeted bypass in WebShell detection. With the accumulation of training samples, the accuracy of the detection model in different application environments gradually improves. This method has obvious advantages compared with traditional machine learning algorithms. However, this method is applicable to all types of WebShell, and it does not flexibly apply text classification to WebShell detection. Obviously, it does not use PHP-type opcode, and the model's accuracy still has room for improvement.

Cui et al. [4] developed a WebShell detection method based on the RF-GBDT algorithm for detecting PHP web pages. The developed method is a two-layer model that combines RF and GBDT [17]. At the first layer, preliminary prediction results are obtained with the RF classifier by using the characteristics of the opcode sequence. Then, the statistical characteristics of the PHP file are combined with the preliminary prediction results of the first layer, participate in the training of the next layer on the basis of the GBDT classifier, and produce the final prediction results. Effective features, such as information entropy, coincidence index, compression ratio, and text features of the opcode sequence, were selected, and the trained model achieved good prediction performance. However, the model has several problems. The TF-IDF used does not consider the distribution information and classification capability of feature words among classes. In addition, its failure to consider WebShell underreporting in applications can cause serious problems.

Different from these previous researchers, we adopted the advantages of WebShell detection in previous work, improved RF-GBDT, and proposed the RF-AdaCost model. The model uses the statistical characteristics of WebShell and pays attention to the opcode sequence of the PHP WebShell. To reduce the false negative rate of the WebShell, we introduced AdaCost and added cost sensitivity while sacrificing the false positive rate, reducing the false negative rate, and minimizing the loss.

3 Method

We proposed RF-AdaCost, which is shown in Fig. 1. RF-AdaCost, which merges RF and AdaCost, is a two-layer model. At the first layer, we obtained preliminary prediction results on the basis of the opcode sequence of the random forest classifier. Then, we merged the statistical characteristics of the PHP file and its preliminary prediction results to participate in the next training layer with the AdaCost classifier to generate the final prediction result.

RF is an ensemble learning method that consists of many decision tree classifiers. It was proposed by Tin Kam Ho [18] and subsequently developed by Leo Breiman [9]. It constructs many child datasets of the original dataset, conducts sampling with replacement, and leverages the child datasets to build a decision tree. The final prediction results are obtained based on the majority of the predictions produced by all decision trees. RF has two key points: randomness and forestry. Randomness means that the process of building child datasets is completely random, that is, every feature and every sample may contribute to the growth of a decision tree. Forestry means that RF generates its prediction from many decision trees that can form a forest. RF can process a large amount of information within a short time, has high accuracy, and usually has better

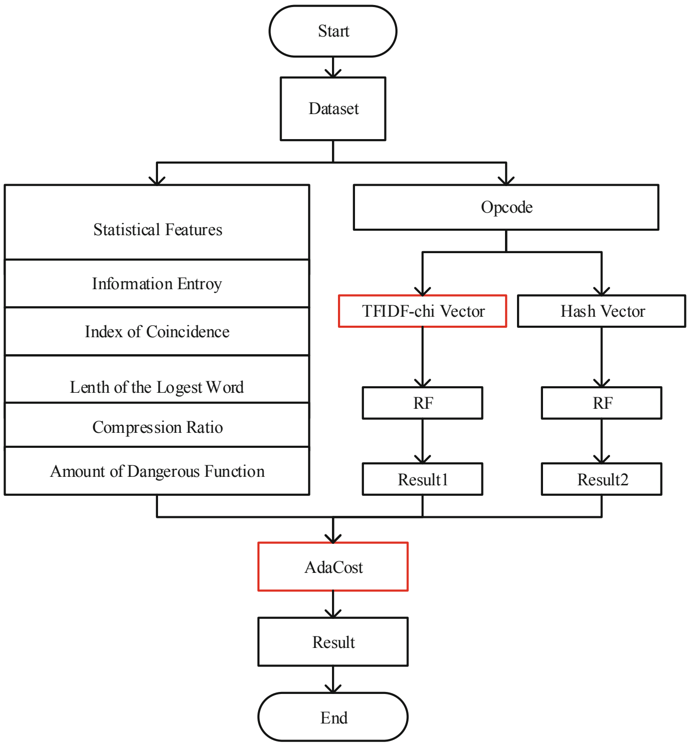

Fig. 1. RF-AdaCost is the fusion of RF and AdaCost. The red box shows the different parts of the model and RF-GBDT. TF-IDF and AdaBoost are used in RF-GBDT. (Color figure online)

performance than decision trees. Therefore, it has a wide range of applications in science and technology.

AdaBoost, which was introduced by Freund et al. [19], learns many "weak" hypothetical high-precision voting sets. Generally, each hypothesis outputs a prediction and the confidence of the prediction. Each hypothesis is trained on the same dataset, but the distribution is different. Different assumptions are made in different booster wheels. In each round, AdaBoost increases the weight of incorrectly classified training instances and decreases the weight of correctly predicted instances. AdaBoost allows arbitrary initial distribution. For classification errors, each example has an equal weight. Costly examples can be given a higher weight than the other examples to reduce the accumulated cost of misclassification. AdaBoost reduces the weighting error of the initial distribution. Schapire et al. [20] provided different weights to false positives and false negatives when applying AdaBoost in text filtering. Karakoulas et al. [21] used a similar method. However, the misclassification cost is not used in the weight update rule of AdaBoost. In AdaCost [10], the weight update rule actively increases the weight of cost misclassification but conservatively reduces the weight of correct cost classification. This condition is achieved by introducing a misclassification cost adjustment function in the weight update formula. Under this update rule, the weight of expensive examples is high, and the weight of cheap examples is low. Each weak hypothesis correctly predicts an expensive example of this distribution. The final voted ensemble also correctly predicts expensive examples. The AdaCost algorithm is shown in Algorithm 1. The red box shows the different parts of AdaCost and AdaBoost.

Algorithm 1 AdaCost[10]

```
● input: $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$,  $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$;
● output: G(x)
● Initialize $D_1$
      $$D_1 = (w_{11}, \cdots, w_{1i}, \cdots, w_{1N}),\ w_{1j} = \frac{1}{N}, i = 1,2,\cdots,N$$
● For m = 1, …, M:
    1. Use the training data set with weight distribution $D_m$ to learn and get the basic classifier:
          $$G_m(x): \mathcal{X} \to \{-1, +1\}$$
    2. Calculate the classification error rate of $G_m(x)$ on the training data set:
          $$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$
    3. Calculate the weight of $G_m(x)$:
          $$\alpha_m = \frac{1}{2}\log\frac{1 - e_m}{e_m}$$
    4. Update the weight distribution of the training data set:
          $$D_{m+1} = (w_{m+1,1}, \cdots, w_{m+1,i}, \cdots, w_{m+1,N})$$
          $$w_{m+1,i} = \frac{w_{m,i}}{Z_m}\exp(-\alpha_m y_i G_m(x_i)\boxed{\beta(i)}, i = 1,2,\cdots,N$$
       where β(i) is a cost-adjustment function. $Z_m$ is a normalization factor chosen so that $D_{m+1}$ will be a distribution.
● Get $G(x)$:
      $$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$
      $$G(x) = \text{sign}(f(x)) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$$
```

3.1 Feature Extraction

We extracted the statistical features, including information entropy, index of coincidence, length of the longest word, compression ratio, and dangerous function, of the PHP file samples. By combining these typical features, we obtained a five-dimensional feature vector. With regard to the PHP file, the text features can be used effectively. Therefore, we extracted the opcode sequence from each PHP sample and determined the text feature TF-IDF–chi and hash vector of the opcode sequence.

1) Information Entropy

Information entropy was introduced by Shannon [22] to represent the disorder and uncertainty of given information by measuring the average uncertainty of all possible information sources. The greater entropy is, the messier the information is. In general, to achieve ambiguity, a WebShell is encrypted and encoded, and several random strings are introduced, leading to an increase in information entropy. Therefore, the value of information entropy can be used to identify WebShell. Information entropy is calculated as

$$Entropy = -\sum_{i=1}^{255} p_{i\ \log p_i}, \tag{1}$$

where $\mathrm{p}_i = \frac{\mathrm{X}_i}{\mathrm{S}}$, S is the total number of characters, X_i is the i-th ASCII code that appears in the file, and $i \neq 127$. Given that 127 is a space, it is not counted.

2) Index of Coincidence

The index of coincidence is also known as IC and was developed by W.F. Friedman [23]. It can evaluate the probability of finding two identical letters by randomly selecting two letters from a given text. Given that the randomness of the encrypted text is improved, the IC value of a WebShell is lower than the IC value of ordinary files, thus providing us new evidence to identify the WebShell. IC is calculated as

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)}, \tag{2}$$

where N is the length of a given text and n_i is the frequency of letter i in the text. Notably, the text has c different letters.

3) Length of the Longest Word

We resort to the proof that an encrypted WebShell may have several abnormal words or strings with an extremely large length. Thus, by measuring the length of the longest word from a given text, we can determine whether it is suspicious and a WebShell.

4) Compression Ratio

Data compression reduces the size of a file by using fewer codes instead of the original presentation. The compression ratio is the ratio of the uncompressed size of the file to the compressed size, as shown in Formula (3). By assigning short codes to high-frequency characters and mapping long codes to low-frequency characters, data compression eliminates the unbalanced distribution of specific characters [24]. A compression algorithm makes the characters of the original file uniform; thus, the more unevenly the bytes are used, the greater the proportion of compression is. After a series of steps, such as WebShell or small encryption, the internal bytes become uneven and thus cause the compression ratio of the WebShell to increase. The compression ratio of the file can therefore be used as one of the characteristics of checking WebShell.

$$Compression\,Ratio = \frac{UncompressedSize}{CompressedSize} \tag{3}$$

5) Dangerous Function

WebShell usually has functions that accidentally appear in normal files, including eval, python eval, and base64 decode. We can calculate the number of characters that can match these functions in each test file and use this number as a feature to help identify the WebShell.

6) TFIDF–chi vector

PHP opcodes are part of machine language instructions and used to specify the operations that need to be performed [25]. Compared with extracting text features directly from PHP source files, extracting text features from opcode sequences is more effective because opcodes can filter some noise in the PHP source code, such as comments. In our experiments, we extracted the opcode sequence of all samples and used it to extract the text features from the opcode sequence, namely, the TFIDF–chi vector [8].

TF-IDF

In the TF-IDF algorithm, IDF represents inverse document frequency, and its characteristics are as follows: if the number of documents containing the same word is large, then the classification capability of the feature word is poor. For example, although several pronouns appear frequently, they do not have the capability to distinguish text. IDF is calculated as

$$idf_i = log\left(\frac{N}{n_i} + 1\right), \tag{4}$$

where N represents the total number of text in the entire training sample and n_i represents the number of text containing the feature word t_i.

Term frequency refers to the frequency of words appearing in the file and is usually expressed as TF. In TF, if the feature word appears frequently in an article, then the feature can effectively express the main information of the text and is suitable for text classification. Keywords appear often in the text, that is, the number of words and the importance of words have a certain positive correlation. However, regardless of the importance of words, a long document may have more words than a short document. TF is the normalization of the number of words to prevent bias toward long files. TF is calculated as

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}, \tag{5}$$

where n_{ij} represents the number of times that word t_i appears in document d_j and $\sum\limits_{\mathrm{k}} n_{\mathrm{k}j}$ represents the sum of all words in document d_j.

In the traditional TF-IDF calculation method, the TF word frequency statistical method is used to describe high-frequency features, and these high-frequency features are often noise words that are not helpful for text classification. Several low-frequency words that can represent text information well because of the low frequency of occurrence are ignored. IDF enhances the weight of feature words with a low frequency of occurrence to make up for the shortcomings of TF [26]. The TF-IDF weighting method combines the two methods, as follows:

$$W_{ij} = tf_{ij} * idf_i. \tag{6}$$

TF-IDF does not consider the distribution of feature words within categories [8]. Under normal circumstances, feature words with an unstable distribution should be given a lower weight than feature words with a relatively stable distribution. Similarly, if a feature word only appears in a certain type of text, then the word does not help in text classification and should be given a low weight. This scenario is not reflected in the traditional TF-IDF.

Advanced TF-IDF algorithm [8]: TFIDF–chi

The chi-square statistical algorithm is used to measure the deviation degree of two variables, in which the theoretical and actual values are independent of each other. A large deviation means that the two variables are independent of each other. Yang et al. [27] showed that the chi-square statistic is one of the most effective feature selection methods in existing text classification. The greater the chi-square value of a feature item in a category is, the more distinctive the feature vocabulary is. However, this statistic has disadvantages. For example, the traditional chi-square does not consider the situation where the feature words are evenly distributed within the category. Specifically, the chi-square statistical formula does not reflect the large weight that feature words should be given when they appear uniformly in a certain type of document; it only focuses on the procedure of giving weights when the frequency of other types of feature words is high and when the frequency of this type is low. The calculation formula of chi-square statistics is

$$chi = \frac{N(AD-BC)^2}{(A+C)(B+D)(A+B)(C+D)}, \tag{7}$$

where A is the frequency of documents containing the feature word t and belonging to category c, B is the frequency of documents that contain the feature word t but do not belong to category c, C is the frequency of documents that belong to category c but do not contain the characteristic word t, D is the frequency of documents that neither belong to c nor contain the characteristic word t, and N is the total number of documents.

The traditional TF-IDF weight calculation method ignores the difference in the distribution of feature words between categories, but the chi-square statistic of feature words can effectively describe the distribution information of feature words between categories. Specifically, the classification capability of the characteristic vocabulary is proportional to its chi-square value. In view of this situation, we introduced chi-square statistical methods to improve the classification capability of feature words between classes. The calculation formula of the feature word weight improvement algorithm based on chi-square statistics is

$$Tfidf - chi = W_{ij} * chi. \tag{8}$$

7) Hash vector

A hash vector can map arbitrary data blocks to a fixed number of locations, and the values of TF are added to these locations. It can almost ensure that the same input produces the same output and that different inputs produce different outputs. This vector is also called a hashing trick, which means that it can use the hash function to determine the index position of the eigenvector and does not need to create a large dictionary. The advantage of this trick is what the TF-IDF method lacks. The TF-IDF method has to create a dictionary, which may require a large memory space. Mathematically, if the hash function hashes the i-th feature to position j, i.e., $h(i) = j$, then the TF value of the j-th original feature $\phi(i)$ will be added to the TF value of j-th feature $\bar{\phi}(j)$, as follows:

$$\bar{\phi}(j) = \sum_{i \in J; h(i)=j} \phi(i), \tag{9}$$

where j is the dimension of the original feature.

3.2 Algorithm

Given this background, this study developed a WebShell detection method that combines statistical features and an opcode, as shown in Algorithm 2.

Algorithm 2 RF-AdaCost

```
1. Prepare the php webshell data set, and randomly divide the data set into a training set and a test set in a 7:3 ratio.
2. Extract the statistical characteristics of each WebShell, such as information entropy, index of coincidence, length of the longest word, compression ratio and danger function. Record as $D = \{d_1, d_2, \ldots, d_N\}$, D is the WebShell training set, where $d_i$ is the i-th WebShell file. $d_i = \{x_{i1}, x_{i2}, x_{ij} \ldots, x_{i5}\}$, $x_{ij}$ is the j-th dimension feature of the i-th webshell.
3. Extract the 146-dimensional Tfidf-Chi vector in the D dataset, and the i-th data is recorded as $d_i' = \{\alpha_{i1}, \alpha_{i2}, \alpha_{ij} \ldots, \alpha_{i146}\}$, RF classifier is used for training to obtain an intermediate model, and then $d_i'$ is input to this model, and the prediction result is used for the sixth dimension feature $x_{i6}$ of $d_i$.
4. Extract the 200-dimensional hash vector in the D dataset, and the i-th data is recorded as $d_i'' = \{\beta_{i1}, \beta_{i2}, \beta_{ij} \ldots, \beta_{i200}\}$, RF classifier is used for training to obtain an intermediate model, and then $d_i''$ is input to this model, and the prediction result is used for the seventh dimension feature $x_{i7}$ of $d_i$.
5. Dataset $D = \{d_1, d_2, \ldots, d_N\}$, $d_i = \{x_{i1}, x_{i2}, x_{ij} \ldots, x_{i5}, x_{i6}, x_{i7}\}$ .Use AdaCost classifier (Algorithm 1), and set different cost factors β for training.
   END FOR
```

3.3 Analysis of the Algorithm

The proposed RF-AdaCost algorithm is a fusion of RF and AdaCost. In this algorithm, the opcode of the WebShell is obtained, and the TFIDF–chi and hash vectors are extracted. The RF classifier is then used to train and predict the results, and the statistical characteristics of the WebShell are ignored. Current detection models cannot detect encrypted variant files well, so this study proposed the RF-AdaCost algorithm, which improves the indicators of the model on the basis of sacrificing time and space. The following experimental results verify this improvement. Using RF to train the opcode of WebShell, the complexity of time and space is O (n). Using AdaCost to train the statistical feature of WebShell, the time and space complexity is O (m). Through analysis, the time and space complexity of RF-AdaCost proposed in this paper is O (m + n).

4 Experiment

The main aim of the experiment was to use the proposed RF-AdaCost model in detecting WebShell and compare it with a model without improvement and other models to verify the effectiveness of the proposed method.

4.1 Dataset and Evaluation Method

The dataset used in this experiment was from Fujian Strait Information Technology Co., Ltd. Given that some data may be repeated, the content is the same, but the file name is different. Thus, the dataset needed to be de-duplicated by message-digest algorithm 5 (MD5) to obtain 2162 WebShell samples and 2236 normal samples at a training set: test set ratio of 7:3. Each group of experiments was repeated thrice, and the average value was obtained. Accuracy, Recall, Precision, FNR, and FPR were used to evaluate performance.

4.2 Data Handling

MD5 deduplication calculates the MD5 value of each file in accordance with the content of the file. If the MD5 value is the same, then the file is considered to be the same and deduplication is performed. After a careful analysis of the PHP file, we found that comments exist in the file. If the comments are also included in the statistical features, then they will affect the extracted features and cause errors in the various indicators of the model results. Regular expressions are therefore required to eliminate comments. An opcode is an intermediate language after the PHP script is compiled. It includes operators, operands, instruction formats and specifications, and data structures that store instructions and related information. PHP script execution has four stages, namely, lexical analysis, grammatical analysis, opcode compilation, and opcode execution. The compiler binds the opcode with the corresponding parameter or function call in the third stage. Even if the WebShell dangerous function is confused and encrypted, the opcode statement will be different from the normal file compilation result during compilation. Therefore, according to this feature, an opcode can be used to distinguish between WebShell and normal files, and the detection of files can be converted into the detection of opcode sequences. This work uses the PHP plug-in logic extension module (Vulcan logic dumper or VLD) to compile and obtain the PHP files' opcode.

4.3 Experiment Results

The experimental environment is a Windows 7 64-bit operating system, python3.6. The processor is Intel Core i5-4570 CPU at 3.20 GHz, and the memory is 16 GB.

4.3.1 Effect of Advanced TFIDF-Chi on the Results

A set of comparative experiments was designed specifically to verify that the opcode's TFIDF–chi vector can represent WebShell features better than the TF-IDF vector can. The experimental results are shown in Table 1.

Table 1. Effect of advanced TFIDF–chi on the results

Feature	Algorithm	Accuracy	Precision	Recall	FNR	FPR
TF-IDF	RF	92.26%	92.66%	91.51%	8.11%	7.01%
TFIDF-chi	RF	93.55%	93.51%	93.36%	6.41%	6.27%

4.3.2 Comparison with the Method Before Fusion

To verify the effectiveness of the RF-AdaCost model, we compared the experimental results of the method before fusion and the method in this study, as shown in Table 2. First, we extract the statistical features of WebShell, and then use AdaCost classifier to train. Then we use opcode of WebShell to extract TFIDF-chi and Hash vectors, and then use RF classifier to train them. Compare these experimental results with our method, and the experimental results are shown in Table 2.

Table 2. Comparison with the method before fusion

Feature	Algorithm	Accuracy	Precision	Recall	FNR	FPR
Statistical features	AdaCost	87.17%	87.13%	86.73%	12.78%	12.39%
TFIDF-chi	RF	93.55%	93.51%	93.36%	6.41%	6.27%
HASH	RF	91.58%	92.15%	90.59%	8.96%	7.46%
Statistical features, TFIDF-chi, HASH	RF-AdaCost	95.30%	95.36%	95.06%	4.76%	4.48%

4.3.3 Comparison with Other Machine Learning Models

To verify that the RF-AdaCost model is better than other methods, we compared it with Hu's method [11] and Zhang's method [12]. In addition, we made a comparison with RF-GBDT. The experimental results are shown in Table 3.

Table 3. Comparison with other machine learning models

	Accuracy	Precision	Recall	FNR	FPR
RF-AdaCost	95.30%	95.36%	95.06%	4.76%	4.48%
RF-GBDT	94.39%	94.70%	93.83%	5.92%	5.07%
Hu Method	85.28%	85.33%	84.82%	14.26%	14.78%
Zhang Method	92.41%	90.90%	93.49%	8.58%	6.12%

4.3.4 Influence of β Factor on the Experimental Results of RF-AdaCost

The effect of setting different cost factors β (Algorithm 1) in RF-AdaCost on various indicators of the experimental results is shown in Table 4.

4.3.5 Comparison with Popular Detectors on the Internet

To verify the performance of RF-AdaCost, we downloaded several popular advanced WebShell detectors from the Internet, namely, D-Shield, SHELLPUB, and WEBDIR+

Table 4. Influence of β factor on the results of RF-AdaCost

RF-AdaCost	Accuracy	Precision	Recall	FNR	FPR
$\beta = 1$	95.30%	95.36%	95.06%	4.76%	4.48%
$\beta = 1.2$	95.14%	94.51%	95.68%	4.23%	5.37%
$\beta = 1.5$	94.84%	93.15%	96.60%	3.41%	6.87%
$\beta = 1.8$	94.01%	92.76%	96.91%	3.17%	8.81%

[28–30], and used the same test set as the model of this article for scanning detection and measurement. The experimental results are shown in Table 5. D-Shield demonstrated good performance in terms of precision and false positive rate. But in other ways, its performance not as good as ours.

Table 5. Comparison with popular detectors on the Internet

Detector	Accuracy	Precision	Recall	FNR	FPR
D Shield	94.92%	96.33%	93.21%	6.37%	3.43%
SHELLPUB	89.98%	93.73%	85.34%	13.05%	5.52%
WEBDIR+	92.26%	95.81%	88.12%	10.66%	3.73%
RF-AdaCost	95.30%	95.36%	95.06%	4.76%	4.48%

4.4 Analysis of Experimental Results and Reasons

The experimental results in Sect. 4.3.1 indicate that the overall performance of TFIDF–chi was better than that of TF-IDF, and the recall rate increased by 1.85%. The reason may be that the TF-IDF weight calculation method based on chi-square statistics makes up for the defect of not considering the distribution of feature words among classes and provides a low weight to words that are evenly distributed among classes but do not have a classification capability. To a certain extent, it improves the traditional calculation method, effectively enhances the accuracy of weight calculation and text classification, and can extract word vectors with classification capability.

The experimental results in Sect. 4.3.2 show that the statistical characteristics of the PHP WebShell alone were not good, and the accuracy was less than 90%. In addition, the TFIDF–chi and hash vectors were better than the statistical characteristics. We conclude that the opcode feature of PHP can effectively reflect the essence of WebShell, but statistical features can also be used as a basis for judging WebShell.

The experimental results in Sects. 4.3.3 and 4.3.5 reveal that the proposed RF-AdaCost model exerted a better overall effect in comparison with the other machine learning algorithms and detectors. Specifically, the accuracy rate of D-Shield is 96.33%, but it is not as good as our model in terms of Recall and FNR, which are aspects that enterprises attach great importance to. In addition, our model has advantages over other models. We can try our best to reduce the false negative rate according to an acceptable range of the false positive rate. Section 4.3.4 shows that the best performance was observed when $\beta = 1.5$. When the β value increased again, the false positive rate increased by 1.94%, and the false negative rate only decreased by 0.24%.

5 Conclusion

In the detection of WebShell, we should pay attention to essential features and use text features, both of which can improve the detection efficiency of WebShell. In practice, the omission of WebShell is disastrous to enterprises. The rate of omission must be reduced under the condition that the rate of false positives is acceptable.

With the development of deep learning, text detection can be used to detect WebShell. If we apply the idea of text detection to WebShell detection, then the essential characteristics of WebShell will be ignored. The combination of the essential characteristics of WebShell with deep learning is worth studying.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (NSFC) under Grant 61972187, the Scientific Research Project of Science and Education Park Development Center of Fuzhou University, Jinjiang under Grant 2019-JJFDKY-53 and the Tianjin University-Fuzhou University Joint Fund under Grant TF2020-6.

References

1. Kim, J., Yoo, D.H., Jang, H., et al.: WebSHArk 1.0: a benchmark collection for malicious web shell detection. J. Inf. Process. Syst. **11**(2), 229–238 (2015)
2. WEB TECHNOLOGY SURVEYS: Usage statistics of server-side programming languages for websites [EB/OL]. https://w3techs.com/technologies/overview/programming_lan-guage. Accessed 15 May 2020
3. Argyros, G., Stais, I., Kiayias, A., et al.: Back in black: towards formal, black box analysis of sanitizers and filters. In: 2016 IEEE Symposium on Security and Privacy (SP). IEEE (2016)
4. Cui, H., Huang, D., Fang, Y., et al.: Webshell detection based on random forest–gradient boosting decision tree algorithm. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), pp. 153–160. IEEE (2018)
5. Salton, G., Yu, C.T.: On the construction of effective vocabularies for information retrieval. ACM **9**(3) (1973)
6. Debole, F., Sebastiani, F.: Supervised term weighting for automated text categorization. In: Sirmakessis, S. (ed.) Text Mining and its Applications. Studies in Fuzziness and Soft Computing, vol. 138, pp. 81–97. Springer, Berlin (2004). https://doi.org/10.1007/978-3-540-45219-5_7
7. Cavnar, W.B., Trenkle, J.M.: N-gram-based text categorization. In: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, p. 161175 (1994)
8. Ying, Ma., Hui, Z., WanLong, L.: Optimization of TF-IDF algorithm combined with improved CHI statistical method. Appl. Res. Comput. **9**, 2596–2598 (2019)
9. Breiman, L.: Random forests. Mach. Learning **45**(1), 5–32 (2001)

10. Stolfo, W.F.S.J.: AdaCost: misclassification cost-sensitive boosting. In: Sixteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc. (1999)
11. Biwei, H.: Research on webshell detection method based on Bayesian theory. Sci. Mosaic (2016)
12. Hewei, Z., Xiaojie, L.: PHP webshell detection method based on text vector. Data Commun. **04**, 16–21 (2019)
13. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM (2016)
14. Truong, T.D., Cheng, G., Guo, X.J., et al.: Webshell detection techniques in web applications In: International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE (2014)
15. Lv, Z.-H., Yan, H.-B., Mei, R.: Automatic and accurate detection of webshell based on convolutional neural network. In: Yun, X., et al. (eds.) CNCERT 2018. CCIS, vol. 970, pp. 73–85. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-6621-5_6
16. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, pp. 1746–1751 (2014)
17. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**(5), 1189–1232 (2001)
18. Ho, T.K.: Random decision forests. In: 1995 Proceedings of the third International Conference on Document Analysis and Recognition. IEEE Computer Society (1995)
19. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**, 119–139 (1999)
20. Schapire, R., Singer, Y., Singhal, A.: Boosting and Rocchio applied to text filtering. In: International ACM SIGIR Conference on Research & Development in Information Retrieval. ACM (1998)
21. Karakoulas, G., Shawe-Taylor, J.: Optimizing classifiers for imbalanced training sets. In: Annual Conference on Neural Information Processing Systems, pp. 253–259 (1999)
22. Shannon, C.E.: A mathematical theory of communication. Bell Labs Tech. J. **27**(4), 379–423 (1948)
23. Friedman, W.F.: The index of coincidence and its applications in cryptology. Department of Ciphers. Publ 22. Geneva, Illinois, USA: Riverbank Laboratories (1922)
24. Bell, T.C., Cleary, J.G., Witten, I.H.: Text Compression. Prentice Hall, Upper Saddle River (1990)
25. Sklar, D.: "Understanding PHP Internals" Essential PHP Tools: Modules, Extensions, and Accelerators, pp. 265–274. Apress, Berkeley (2004)
26. Xue, X.B., Zhou, Z.H.: Distributional features for text categorization. IEEE Trans. Knowl. Data Eng. **21**, 428–442 (2006)
27. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. In: Proceedings of the Fourteenth International Conference on Machine Learning (ICML), pp. 412–420. Morgan Kaufmann Publishers Inc., San Francisco (1997)
28. D shield takes the initiative to protect your website[CP/DK]. http://www.d99net.net/. Accessed 15 May 2020
29. SHELLPUB.COM Focus on killing[CP/DK]. https://www.shellpub.com/. Accessed 15 May 2020
30. Next generation webshell detection engine[CP/DK]. https://scanner.baidu.com/. Accessed 15 May 2020