XSS Attack Detection Model Based on Semi-supervised Learning Algorithm with Weighted Neighbor Purity

Xinran Li, Wenxing Ma, Zan Zhou(✉), and Changqiao Xu

Beijing University of Posts and Telecommunications, Beijing 100191, China
{lxr_bupt,cqxu}@bupt.edu.cn, mwx0519@sina.com, zan.leon.zhou@gmail.com

Abstract. With the popularity of web applications, cyber security is becoming more and more important. The most common web attack is cross-site scripting (XSS), which can be easily constructed in malicious URLs. However, the existing methods of detecting XSS attacks are suffering from the lack of labeled data, and some semi-supervised methods still have the problem of mislabeling. In this paper, we propose a novel XSS attack detection model based on semi-supervised learning algorithm with weighted neighbor purity. Semi-supervised learning can make best use of little labeled data, and a simple mechanism of neighbor purity using weighted-kNN is applied to rectify mislabeled samples, improving classification accuracy. To verify the feasibility of our solution in real-world scenario, we collected real HTTP requests in the China Education and Research Network (CERNET) as training data. The comparison experiment shows that proposed method performs better than a well-known semi-supervised algorithm and a recently published ensemble learning method in different initially labeled rates.

Keywords: XSS attack · Machine learning · Semi-supervised learning · Binary classification

1 Introduction

As the Internet has become an indispensable part in many people's lives, a lot of Internet companies make profit from their web services. However, adversaries can collect information from such a large quantity of Internet traffic and conduct tentative attacks [30]. These attacks have resulted in increasing security incidents like leakage of massive sensitive data [29], causing huge economic losses. The most widespread threat among the top 10 critical web security risks in 2017, according to the Open Web Application Security Project (OWASP) [17], was Cross-site scripting (XSS), which was unfortunately found in about two-thirds of all the web applications. Besides, the data provided by a cybersecurity leader team Imperva [3] showed that - XSS remained as the runner up category of web vulnerabilities in 2019, and probably one of the dominant categories in 2020.

L. A. Grieco et al. (Eds.): ADHOC-NOW 2020, LNCS 12338, pp. 198–213, 2020.
https://doi.org/10.1007/978-3-030-61746-2_15

The great threat of information safety and economic losses warn us to focus on XSS attack detection.

Many web attacks are executed by simply clicking a malicious URL from web browsers, including XSS attacks under discussion. The majority of successful XSS attacks are built by deliberate URLs with malicious code. When victims click an elaborately constructed URL, an evil request is sent to the web server. At the same time, the malicious code is executed on victims' browser, resulting in information leakage, keylogging, dynamic downloads and other serious consequences [18]. Sometimes the evil input sent to the server side will be stored in database, threatening more visitors of this webpage.

To detect XSS attack, this paper focuses on distinguishing malicious URLs with XSS attacks from the benign. In the field of URL detections using artificial intelligence, actually in all the cyber security areas using machine learning, the limited malicious database is a serious problem [16,22], leading many approaches to use not real representative or intentionally generated dataset. Thus, these approaches are lack of empirical evidence against real-world scenarios. Some studies use the semi-supervised learning method to take more advantage of existing labeled data. However, the problems of mislabeling and overfitting in the developed model still need to be solved [12].

In face of the issues above, this paper develops a novel classification model for automatically detecting XSS attacks in malicious URL requests based on a semi-supervised algorithm using weighted neighbor purity. Machine learning method can greatly improve the classification accuracy [20], especially a semi-supervised method based on ensemble learning can eliminate the lack of labeled data. To address mislabeling and overfitting problems in semi-supervised learning, we add two novel restrictions on a well-known algorithm. Also, real-world scenarios are considered by using the real data flows in both training and testing data set.

The main contributions of this paper are summarized as follows:

(1) To improve the classification accuracy of semi-supervised learning algorithm, we ameliorate traditional Co-Training algorithm [1] by calculating weighted neighbor purity as a threshold to reduce mislabeled data, and introducing an excess ratio to address the overfitting problem.
(2) Considering the practical use of real-world application, we train our model by captured URLs from HTTP requests in CERNET and feasible attacks that have caused loss collected by XSSed [4]. To achieve the usability of our detection model, practical data cleaning method is also introduced.
(3) Finally, comparison experiments are conducted to compare our improved method with the original algorithm and another ensemble learning approach published last year, showing that the proposed method outperforms the other two methods.

The rest of this paper is structured as follows: Sect. 2 summarizes the related work about URL detection and semi-supervised learning algorithm. Section 3 explains the overview of proposed detecting model. Section 4 describes the

improved semi-supervised algorithm in detail. Section 5 represents the implementation and experimental result of three methods. Section 6 concludes this paper.

2 Related Work

In this section, we introduce the studies of URL attack detection w ith machine learning and semi-supervised learning algorithm. The knowledge in these two respects mostly supports our research.

2.1 Research on URL Attack Detection with Machine Learning

Many web attacks can be simply constructed by embedding or injecting malicious code in URLs [27], and XSS attack is the dominant type among them [19]. Using a machine learning algorithm, we can extract representative static features from URL [20] and train a prediction model to classify benign and malicious samples. Thus, the performance of machine learning depends largely on features. Machine learning methods can detect unknown malicious code, unlike universally used methods based on blacklists. Furthermore, a classifier built by machine learning method consumes little time to predict whether an URL is malicious, that is the reason why more and more web servers choose to use machine learning method for security issues.

Cui et al. [2] designed a feasible feature process procedure. They used gradient learning to perform statistical analysis and their feature extraction is conducted based on a sigmoidal threshold level. Yang [26] created a multi-classifier model for URL with syntax and domain features, indicating that the best performance is achieved by random forest algorithm. Joshi et al. [10] proposed an ensemble classification method with lexical static features to detect malicious URLs, which is currently being used in the FireEye Advanced URL Detection Engine. Zhou et al. [28] also used an ensemble learning method which is based on Bayesian networks, but they combined domain knowledge and threat intelligence with the traditional lexical features, achieving a good result.

The current studies paid more attention to feature extraction, but few of them believes the balanced dataset should be manually checked, though it could consume a large amount of manpower. Instead of using real-world data, most of these studies used generated data. These data could have high-level of identity for training their models, which reduces the reliability of their methods to some extent.

2.2 Research on Semi-supervised Algorithm

In traditional supervised learning, although it's easy to obtain plenty of unlabeled URL samples, it takes a lot of manually labeling time and cost to provide labels for them. To address this challenge, semi-supervised learning is proposed to improve learning performance with mostly unlabeled data. Disagreement-based semi-supervised learning using ensemble learning method is one of the

mainstream paradigms in this field [31]. It applies multiple classifiers to utilize unlabeled data, and the disagreement between learners is critical to learning effectiveness. Research on disagreement-based semi-supervised learning began with the work of Blum and Mitchell on Co-Training [1].

There have been many studies on improving Co-Training. Zhou and his collaborators proposed Tri-training [32], which can be seen as an extension of the binary classifier cooperative training. It generated three classifiers from a single-view training set and then used these three classifiers by the rule of majority to generate pseudo-labeled samples and solved the confidence evaluation problem of pseudo-labeled samples. After training, the three classifiers were used as one classifier through a voting mechanism. Li et al. proposed Co-Forest that puts emphasis on the importance of ensemble learning [11]. Gu et al. proposed an ensemble multi-train method of heteromorphic multi-classifiers [7]. The training process of this method is similar to that of Co-Forest but required each base classifier to use a different learning algorithm, and at the same time use different attribute reduction strategies. In this paper, the random forest algorithm is integrated into the semi-supervised learning framework, which can further improve the learning performance of the classifier, and the introduction of multiple classifiers simplifies the calculation of the pseudo-labeled confidence.

In Co-Training, mislabeled data can accumulate during training, which influences diversity and accuracy of the combined classifier. Xiang et al. [24] developed a visual analysis method, which could improve the quality of labeled samples interactively. The quality improvement was achieved through the use of user-selected trusted items and much manual work was required. The study in [12] used data editing in Tri-training [32], which significantly improved the classification performance. Li et al. proposed an improved naive Bayes self-training algorithm based on weighted K-nearest neighbors. Selecting the samples with the similar spatial structure of the labeled samples, the naive Bayes classifier assorted unlabeled samples on a better spatial structure and reduced mislabeled samples effectively [21]. In this paper, the weighted K-nearest neighbor rule is used to rectify the mislabeled data more accurately.

Based on these existing studies, we combine the wisdom of URL attacks detection method and machine learning with a semi-supervised algorithm, for designing a novel detection model of XSS attacks in URL. The lack of labeled data is solved by machine learning, in which real scene and data are considered. Furthermore, to solve the mislabeling problem in semi-supervised learning, we propose a weighted neighbor purity method to rectify pseudo-labeled samples.

3 XSS Attack Detection Model Based on Improved Semi-supervised Learning

This section describes the proposed detection model, as shown in Fig. 1. Following the process lines, raw URL data collected from real world are divided into training set and testing set, and then they are sent into URL processing procedure. The URL processing procedure transfers raw data in training set and

testing set, respectively, into practicable feature vectors. Particularly in training process, we add a data cleaning step to eliminate noise in features. After URL processing, the refined data represented as feature vectors from training set are sent to learning process to train a detection model. The detection model is then evaluated by classifying the feature vectors generated from the testing set. If the evaluation result is good enough, the generated detection model can be put into application. Otherwise the control flow goes back to URL processing procedure, trying to extract more representative features.

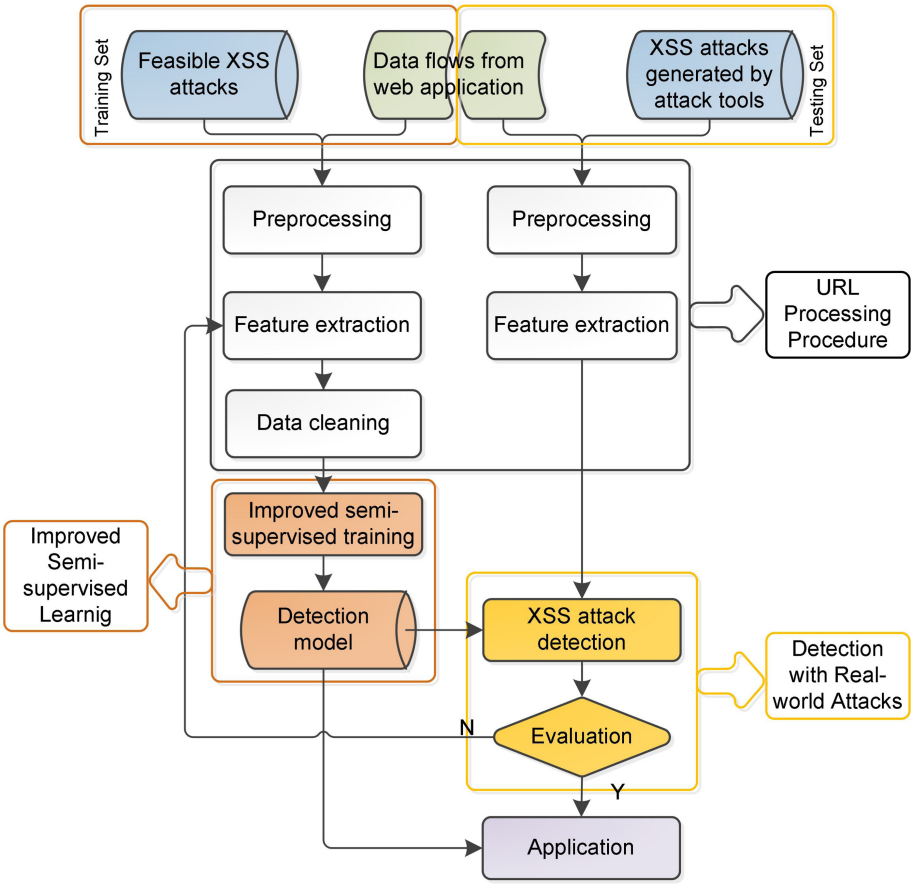


Fig. 1. XSS attack detection model based on improved semi-supervised learning

3.1 URL Processing Procedure

Preprocessing. URLs in HTTP request are often encoded, leading to confusion of both human and artificial intelligence. Many attacks also use encoding method

to bypass traditional attack detectors. That is why we need a preprocessing step to check possible attacks. Figure 2 shows the preprocessing procedure, transforming encoded data into human-readable strings. We firstly pick out the URL query section from normal data packet shown in the first box. In the second box, we get the extracted URL query. After URL decoding, HTML entity decoding and lowercasing, URL query is transferred into apprehensible string in the third box.

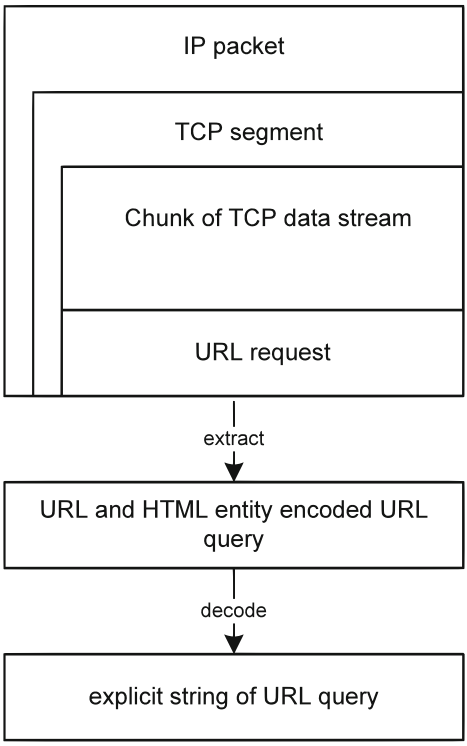


Fig. 2. Preprocessing procedure

Feature Extraction. Feature extraction is conducted on the decoded URL query strings of the last step. According to the previous researches [15,25], we use 70 URL static features as the original attributes. To eliminate the deviation of redundant attributes, we carry out a feature selection algorithm, which is based on correlation between features and its category. After the selection, 14 effective features are selected. These useful features consist of 5 URL structural features, 1 XSS risk level feature, 7 evil char features, and 2 evil keyword features.

Data Cleaning. Considering the adaptation of different practical scenarios, data cleaning is necessary to filter out outliers in training process. The outliers in training set can be regarded as noise, which badly affects the universality of the model to be trained. Feature vectors in training set are projected into a plane space by a dimensionality reduction method t-SNE [14]. Then a density-based clustering algorithm DBSCAN [5] can easily find out the outliers. For test process, as it will omit some real attacks in test data, we do not conduct data cleaning method.

3.2 Improved Semi-supervised Learning

The core algorithm in detection model is a semi-supervised machine learning method improved by weighted neighbor purity. We can describe the target of detecting XSS attacks as a binary classification issue, where positive samples are malicious and negative samples are benign [2]. As for classification, the fundamental assumption is that malicious attacks and benign requests have different features. In this way, the classification model can learn how to predict a new feature vector by adjusting itself according to labeled feature vectors.

Unlike supervised learning method, semi-supervised machine learning can learn from both labeled and unlabeled data, saving a large amount of manpower for labeling. We improve the existing method by introducing weighted neighbor purity and an excess ratio, to ameliorate the accumulating mislabeled error problem and the overfitting problem caused by imbalanced training data. Section 4 describes the improved machine learning algorithm in detail.

3.3 Detection with Real-World Attacks

As attackers usually use attack tools to automatically build XSS attacks, our model is intended to detect newly constructed attacks from normal requests. Therefore, the test set is mixed with XSS attacks generated by attack tools and benign URL requests from life scene in the training set. Although the evil data in training set is collected from real-world XSS attack in website XSSed [4], which is not similar with the test set, our model performs well according to the experiment result.

To evaluate the performance of the detection result, we use classification accuracy as the metric. Classification accuracy is the proportion of correct predictions to all predictions. Equation 1 defines the accuracy, where P and N is the number of positive and negative predictive value respectively, $TP + TN$ is the number of correctly predicted samples, and $P + N$ is the total number of predictions.

$$Accuracy = \frac{TP + TN}{P + N} \tag{1}$$

4 Improved Semi-supervised Algorithm

This section describes the core algorithm in our detection model. An improved semi-supervised algorithm is proposed similar to Co-Forest [11] for binary classi-

fication problems. In the following discussion, we use L to denote labeled samples and U to denote unlabeled samples. $C = \{C_1, C_2, ..., C_n\}$ represents the classification space of labeled samples. First, a labeled training set L is used to build a random forest composed of N base classifiers $H = \{H(1), H(2), ..., H(N)\}$. $H(i)(i = 1, ..., N)$ represents a base classifier in H, and a new ensemble classifier composed of the remaining $N - 1$ classifiers is called the companion classifier of $H(i)$, denoted by $H^*(i)$. During iterations, each companion classifier provides its most confident pseudo label to its base classifier and expand the training set of base classifiers.

4.1 Solution for Overfitting

When building a model for detecting malicious URL, there are often far more samples of normal URL than malicious URL ones. To solve the overfitting problem resulted by extreme imbalance of positive and negative samples, we propose a simple processing mechanism. For each iteration, we measure the proportion of newly added pseudo-labeled samples of different classifications according to the proportion of initially labeled data in different classifications. Suppose the initially labeled samples have N_p positive samples and N_n negative samples, and the ratio of positive and negative samples is r, which is defined in Eq. 2. The allocation of pseudo-labeled samples added to each classifier in each iteration should follow the ratio r, and the excess samples will be put back in U.

$$r = \frac{N_p}{N_n} \tag{2}$$

4.2 Rectification for Mislabeled Data

To reduce the number of mislabeled samples and to improve the performance of the classifier, this paper treats the samples differently according to the marginality of the samples. If the nearest neighbors of a sample mostly belong to the same category, then it has a higher probability of belonging to this category. On the contrary, if the categories of the nearest neighbors are more uniformly distributed, it will be difficult to determine the category according to the neighbors. For each unlabeled sample $x(x \in U)$, we construct a neighbor set $neighbor(x)$ composed of labeled samples. The purity of the neighbors [13] can describe the distribution of neighbor sample categories. This paper proposes weighted neighbor purity, which can more accurately describe the distribution of the neighbors. In order to avoid calculation errors caused of value interval difference in different attributes, it is necessary to normalize each attribute value first. Then, all samples are mapped into points in a multidimensional space, and the distance $dis(x, z)$ between the unlabeled samples x and z refers to their Euclidean distance. The weight of each point w_v is the confidence of the sample, which is defined in Eq. 3. And the weight of each edge $w_e(x, z)$ is defined in Eq. 4.

$$w_v(x) = confidence(x) = \frac{\max\{N_1, N_2\}}{N - 1} \tag{3}$$

$$w_e(x,z) = e^{-\frac{dis^2(x,z)}{8}} \tag{4}$$

Where N_1, N_2 denotes the number of y_1, y_2 labeled by $H^*(x)$ respectively. We use k to denote the number of the nearest neighbors. Then, the weight of the negative samples in k neighbors $W_n(x)$ is defined in Eq. 5. The weight of the positive samples in k neighbors $W_p(x)$ is defined in Eq. 6, where n_i denotes the i^{th} nearest neighbor of x. The weighted neighbor purity is defined in Eq. 7.

$$W_n(x) = \sum_{i=1}^{k} w_v(x) * w_e(x,n_i), y(n_i) = y_1 \tag{5}$$

$$W_p(x) = \sum_{i=1}^{k} w_v(x) * w_e(x,n_i), y(n_i) = y_2 \tag{6}$$

$$wpurity(x) = \frac{W_p(x)}{W_p(x)+W_n(x)} \ln \frac{W_p(x)}{W_n(x)} + \ln \frac{2 * W_n(x)}{W_p(x)+W_n(x)} \tag{7}$$

In this paper, samples with low neighbor purity are called margin samples. The introduction of margin samples is important to improve the generalization ability of the classifier and approximate the ideal hypothesis. We add margin samples that have higher confidence than θ_h to the set of labeled samples, while manually label the samples that have lower confidence than θ_l. For samples with lower marginality, we combine the idea of weighted K-nearest neighbor algorithm to rectify pseudo-labeled samples. That is to say, we sample the k neighbors closest to x, then compare the weights of the positive and negative samples, and the label of the sample is rectified according to the category with larger weight.

The main algorithm flow that applies the semi-supervised learning is given in Algorithm 1. First, N random trees are constructed using labeled samples to build a random forest. For each classifier, we sample some unlabeled data and label them with the companion classifier. If the confidence level and the weighted neighbor purity are high, data editing is employed. On the other hand, if the confidence level and the weighted neighbor purity are low, the sample is manually labeled. Excess samples of a classification are put back in U according to the ratio r. When all classifiers stop updating, training ends.

5 Experiment

This section describes the detailed experiment settings, consisting of data resources, feature selection method, training parameters and comparison experiment results about our method, and other two competitive methods.

Algorithm 1. Improved Semi-supervised Algorithm

Input: L: the labeled set; U: the unlabeled set; θ_h: the high confidence threshold; θ_l: the low confidence threshold; θ_p: the threshold of the weighted neighbor purity; N: the number of random trees; K: the number of the nearest neighbors

Output: classifiers $H = \{H(1), H(2), ..., H(N)\}$;

build a random forest consisting N random trees with L
for $i \in \{1, ..., N\}$ **do**
    $\hat{e}_{i,0} = 0.5$
    $W_{i,0} = 0$
end for
$t = 1$
repeat
    for $i \in \{1, ..., N\}$ **do**
        compute the estimated error rate $\hat{e}_{i,0} = EstimateError(H_i, L)$
        $L^{'}_{i,t} = \phi$
        if $\hat{e}_{i,t} < \hat{e}_{i,t-1}$ **then**
            sample some unlabeled data $U^{'}_{i,t} = SubSampled(U, \frac{\hat{e}_{i,t-1} W_{i,t-1}}{\hat{e}_{i,t}})$
            for each $x \in U^{'}_{i,t}$ **do**
                if $confidence(H_i, x) > \theta_h$ **then**
                    if $wpurity(x) > \theta_p$ **then**
                        compare $W_p(x)$ and $W_n(x)$ to correct the mislabeled data
                    end if
                    add x to the labeled dataset $L^{'}_{i,t} = L^{'}_{i,t} \cup \{(x, H_i(x))\}$
                    $W_{i,t} = W_{i,t} + confidence(H_i, x)$
                else if $confidence(H_i, x) < \theta_l$ and $wpurity(x) < \theta_p$ **then**
                    label the data manually and add it to $L^{'}_{i,t}$
                end if
            end for
            put excess samples back in U according to the ratio r
        end if
    end for
    for $i \in \{1, ..., N\}$ **do**
        if $e_{i,t} W_{i,t} < e_{i,t-1} W_{i,t-1}$ **then**
            update the classifier $h_i = LearnRandomTree(L \cup L^{'}_{i,t})$
        end if
    end for
    $t = t + 1$
until none of the trees in random forest changes

5.1 Dataset

To simulate the practical application of our model, we captured 54.8 MB data flows from outgoing traffic of Beijing University of Posts and Telecommunications network as the normal samples. After extracting the investigated URL request from the flows, we obtain 39596 distinct queries. The majority of these samples are used as white data in training set, and the remaining 3770 normal samples are randomly selected into testing set.

The evil XSS attacks in training set are collected from a well-known security website XSSed [4], containing 28776 unique attacks in URL requests. The XSS attacks in testing set are collected from several GitHub repositories, consisting of 3770 distinct attacks. Because of the difference between training and testing data, our model cannot completely learn the feature pattern of testing samples during training procedure, which greatly supports the validity of our evaluation result.

5.2 Feature Selection

Feature selection is an important procedure that determines the efficiency of machine learning algorithm. This part is executed after URL decoding, HTML entity decoding and lowercasing in the preprocessing procedure of training process. After roughly selecting 70 static features according to [15,25], we use a built-in algorithm of Waikato Environment for Knowledge Analysis (WEKA) [23], and finally pick out 14 useful features shown in Table 1, where URL structural features are statistical features for the whole clean strings; XSS risk level feature is calculated by the cumulative number of the XSS keywords occurrence. The evil char and evil keyword features are the respective numbers of certain char and keyword occurrence.

The selection algorithm named CfsSubsetEval [8] calculates the individual predictive ability of each feature in a subset of attributes, the degree of redundancy is evaluated as well. Searched by greedy algorithm, subsets of features having low intercorrelation and high correlation with the category will be recommended.

5.3 Training Parameters

This subsection explains the detailed parameters in our comparison experiments of improved semi-supervised method, original Co-Forest algorithm and the reproduction of an ensemble learning method published last year [28].

In the proposed semi-supervised method, the algorithm benefits from ensemble learning, using Random Tree algorithm as the base classifier, and the N value is set to 10. The other parameters use the default parameters of the random forest package in WEKA. The high confidence threshold θ_h is 0.75 and the low confidence threshold θ_l is set to 0.65. The weighted neighbor purity threshold is set to 0.02. The size of the nearest neighbor set is related to the confidence of the pseudo-label samples and the number of iterations and has no significant impact on the accuracy of classifiers. The nearest neighbor set size is set to 5 in this experiment.

The original Co-Forest [11] algorithm in comparison uses consistent configuration for base classifier with the improved algorithm. Its only confidence threshold of the labeled samples is set to 0.75, as high as our high confidence threshold.

Table 1. Selected features

Category	Feature name	Description
URL structural	URL_length	the total length of a URL query
	digit_percentage	the percentage of digit in a URL query
	letter_percentage	the percentage of letter in a URL query
	parameter_number	number of parameters in a URL query
XSS risk level	XSS_count	accumulative number of XSS keywords occurrence
evil char	“	existence of char “
	<	existence of char <
	\	existence of char \
	,	existence of char ,
	%	existence of char %
evil keyword	img	existence of word *img*
	eval	existence of word *eval*

In the reproduction of another ensemble learning method using Bayesian network as base classifier, we exactly use the same parameters as the paper mentioned [28]. For ensemble learning, bagging and majority voting methods are used to generate different training subsets and predict results. The number of distinguishing base learners is 5. For each base classifier, Tabu [6] search algorithm and BDeu [9] scoring function are applied.

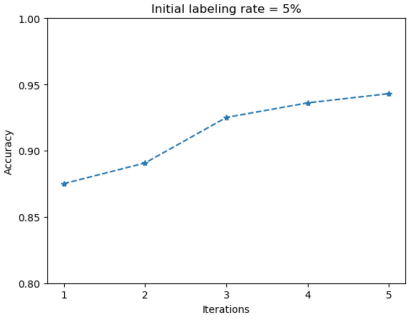
All the approaches are conducted with different initial rates of labeled training sets, which are respectively labeled as 5%, 10%, 20%, and 30%. The average classification accuracy of five tests is finally used for comparison.

5.4 Experiment Result

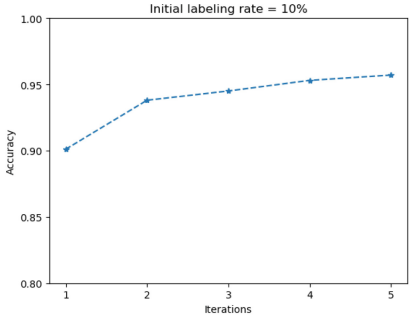
Figure 3 shows the improved algorithm classification accuracy of different initially labeled proportions. From this figure, it can be seen that in each initially labeled proportion of 5%, 10%, 20%, and 30%, the accuracy increases as the number of iterations increases. Table 2 shows the number of samples that need to be manually labeled as well as exercise classification accuracy for each iteration. Only a few samples need to be manually labeled each iteration, which meets the actual production requirements.

Table 2. The accuracy and manually labeled number of each iteration under different initially labeled proportions. N_{manual} denotes the manually labeled number.

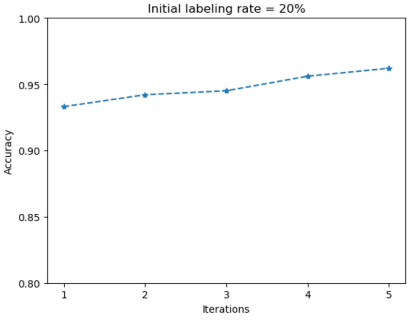
	5%		10%		20%		30%	
	Accuracy	N_{manual}	Accuracy	N_{manual}	Accuracy	N_{manual}	Accuracy	N_{manual}
1	0.875	53	0.901	36	0.933	18	0.942	28
2	0.906	39	0.938	18	0.942	14	0.956	19
3	0.925	48	0.945	61	0.945	27	0.958	20
4	0.936	27	0.953	46	0.956	23	0.965	13
5	0.943	18	0.957	24	0.962	10	0.973	06



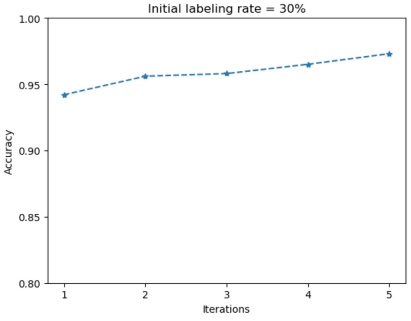
(a) Initially labeled proportion = 5%.



(b) Initially labeled proportion = 10%.



(c) Initially labeled proportion = 20%.



(d) Initially labeled proportion = 30%.

Fig. 3. The improved algorithm classification accuracy of different initially labeled proportions.

Moreover, we compare our work with the Co-Forest algorithm and another ensemble learning algorithm. Table 3 shows the comparison of the accuracy of three algorithms under different initially labeled proportions. As can be seen from the Fig. 4, the improved semi-supervised algorithm has the highest accuracy among the three methods in each initially labeled proportion. The fewer labeled samples, the more obvious the advantages of the proposed algorithm.

Table 3. Algorithm classification accuracy comparison.

Labeled proportion	Improved algorithm	Co-Forest	Ensemble Bayesian Network
5%	94.3%	88.2%	57.6%
10%	95.7%	92.3%	75.7%
20%	96.5%	94.0%	68.9%
30%	97.3%	95.2%	77.5%

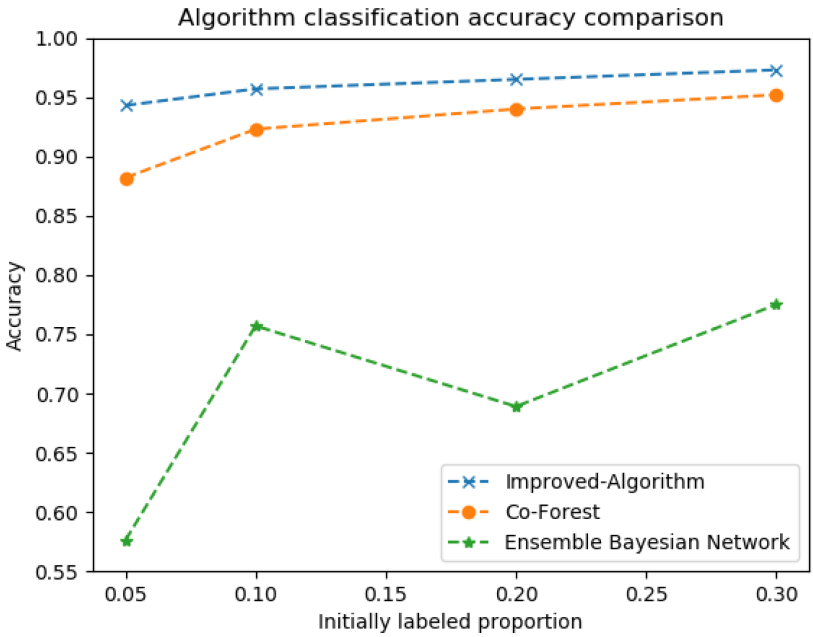


Fig. 4. Algorithm classification accuracy comparison.

Although all of the methods use ensemble learning, the ensemble Bayesian network performs worst in low labeling rates. That is because it does not apply semi-supervised learning to gain more knowledge from limited data, though it actually performs well when learning a fully labeled data set with the classification accuracy of 0.96. In the comparison of two semi-supervised algorithms, the improved one takes the importance of margin samples into account and rectifies mislabeled samples, thereby improving the performance of our classifier.

In general, applying the proposed algorithm to XSS detection has high accuracy and requires little manpower to label the data, proved to have high application value.

6 Conclusion

This paper developed a novel classification model for automatically detecting malevolent URL request with an improved semi-supervised algorithm. To improve the classification accuracy of semi-supervised algorithm, we introduced the weighted purity of edge samples to address the problem of accumulating mislabeled data, and an excess ratio is taken into account for the overfitting problem. In addition, we collected real network traffic in the CERNET and feasible XSS attacks that had caused loss in history for training, achieving practical value of real-world scenario. The experiment showed that our method exceeded a well-known semi-supervised method Co-forest and another competitive ensemble learning method.

In future work, more precise features and the semantic features of the attack can be analyzed and they will greatly improve the universalism of the detection. Furthermore, the proposed method can only detect the evil URLs with XSS attack, and more malicious behaviors in URLs will be taken into consideration, such as code injection, filename attack and so on.

References

1. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, pp. 92–100 (1998)
2. Cui, B., He, S., Yao, X., Shi, P.: Malicious URL detection with feature extraction based on machine learning. Int. J. High Perform. Comput. Networking **12**(2), 166–178 (2018)
3. Dima, B., Sarit, Y.: The state of web application vulnerabilities in 2019. https://www.imperva.com/blog/the-state-of-vulnerabilities-in-2019/. Accessed 5 Mar 2020
4. DP, KF: Xssed archive. http://www.xssed.com/archive. Accessed 5 Mar 2020
5. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. KDD **96**, 226–231 (1996)
6. Glover, F.: Tabu search: a tutorial. Interfaces **20**(4), 74–94 (1990)
7. Gu, S., Jin, Y.: Multi-train: a semi-supervised heterogeneous ensemble classifier. Neurocomputing **249**, 202–211 (2017)
8. Hall, M.A.: Correlation-based feature selection for machine learning (1999)
9. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. Mach. Learn. **20**(3), 197–243 (1995)
10. Joshi, A., Lloyd, L., Westin, P., Seethapathy, S.: Using lexical features for malicious URL detection-a machine learning approach. arXiv preprint arXiv:1910.06277 (2019)
11. Li, M., Zhou, Z.H.: Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **37**(6), 1088–1098 (2007)
12. Liu, R., Verbič, G., Ma, J.: A new dynamic security assessment framework based on semi-supervised learning and data editing. Electr. Power Syst. Res. **172**, 221–229 (2019)
13. Liu, Z., Gao, Z., Li, X.: Co-training method based on margin sample addition. Chin. J. Sci. Instrum. **39**(3), 45–53 (2018)

14. Maaten, L.V.D., Hinton, G.: Visualizing data using t-sne. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
15. Mereani, F.A., Howe, J.M.: Detecting cross-site scripting attacks using machine learning. In: Hassanien, A.E., Tolba, M.F., Elhoseny, M., Mostafa, M. (eds.) AMLTA 2018. AISC, vol. 723, pp. 200–210. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74690-6_20
16. Mokbal, F.M.M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., Xiaoxi, W.: MLPXSS: an integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique. IEEE Access **7**, 100567–100580 (2019)
17. OWASP, T.: Top 10–2017. The Ten Most Critical Web Application Security Risks. OWASP^{TM} Foundation. The free and open software security community (2017). https://www.owasp.org/index.php/Top_10-2017_Top_10
18. Raman, P.: JaSPIn: JavaScript based Anomaly Detection of Cross-site scripting attacks. Ph.D. thesis, Carleton University (2008)
19. Rodriguez, G., Torres, J., Flores, P., Benavides, E.: Cross-site scripting (XSS) attacks and mitigation: a survey. Comput. Networks 106960 (2019). https://doi.org/10.1016/j.comnet.2019.106960
20. Sahoo, D., Liu, C., Hoi, S.C.H.: Malicious URL detection using machine learning: a survey. CoRR abs/1701.07179 (2017). http://arxiv.org/abs/1701.07179
21. Tingting, L., Jia, L.: Improved Naive Bayes self-training algorithm based on weighted k-nearest neighbor. J. Wuhan Univ. (Nat. Sci. Ed.) (2019)
22. Vinayakumar, R., Soman, K., Poornachandran, P., Mohan, V.S., Kumar, A.D.: Scalenet: scalable and hybrid framework for cyber threat situational awareness based on DNS, URL, and email data analysis. J. Cyber Secur. Mobil. **8**(2), 189–240 (2019)
23. Witten, I.H., Frank, E.: Data mining: practical machine learning tools and techniques with java implementations. ACM SIGMOD Record **31**(1), 76–77 (2002)
24. Xiang, S., Ye, X., Xia, J., Wu, J., Chen, Y., Liu, S.: Interactive correction of mislabeled training data. In: 2019 IEEE Conference on Visual Analytics Science and Technology (VAST), pp. 57–68 (2019)
25. Yang, J., Yang, P., Jin, X., Ma, Q.: Multi-classification for malicious URL based on improved semi-supervised algorithm. In: 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), vol. 1, pp. 143–150. IEEE (2017)
26. Yang, P.: A Study on Real-time Detection of URL Attack Behavior Based on Machine Learning. Master's thesis, Beijing University of Posts and Telecommunications (2018)
27. Yang, W., Zuo, W., Cui, B.: Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network. IEEE Access **7**, 29891–29900 (2019)
28. Zhou, Y., Wang, P.: An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. Comput. Secur. **82**, 261–269 (2019)
29. Zhou, Z., Qiao, Y., Zhu, L., Guan, J., Liu, Y., Xu, C.: Differential privacy-guaranteed trajectory community identification over vehicle ad-hoc networks. Internet Technol. Lett. **1**(3), e9 (2018)
30. Zhou, Z., Xu, C., Kuang, X., Zhang, T., Sun, L.: An efficient and agile spatio-temporal route mutation moving target defense mechanism. In: ICC 2019–2019 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2019)
31. Zhou, Z.H.: Disagreement-based semi-supervised learning. Acta Automatica Sinica **39**(11), 1871–1878 (2013)
32. Zhou, Z.H., Li, M.: Tri-training: Exploiting unlabeled data using three classifiers. IEEE Trans. Knowl. Data Eng. **17**(11), 1529–1541 (2005)