Classification of XSS Attacks by Machine Learning with Frequency of Appearance and Co-occurrence

Sota Akaishi
School of Computer Science
Tokyo University of Technology
1404–1 Katakuramachi, Hachioji, Tokyo 192–0982
Email: c0115001a8@edu.teu.ac.jp

Ryuya Uda
School of Computer Science
Tokyo University of Technology
1404–1 Katakuramachi, Hachioji, Tokyo 192–0982
Email: uda@stf.teu.ac.jp

***Abstract*—Cross site scripting (XSS) attack is one of the attacks on the web. It brings session hijack with HTTP cookies, information collection with fake HTML input form and phishing with dummy sites. As a countermeasure of XSS attack, machine learning has attracted a lot of attention. There are existing researches in which SVM, Random Forest and SCW are used for the detection of the attack. However, in the researches, there are problems that the size of data set is too small or unbalanced, and that preprocessing method for vectorization of strings causes misclassification. The highest accuracy of the classification was 98% in existing researches. Therefore, in this paper, we improved the preprocessing method for vectorization by using word2vec to find the frequency of appearance and co-occurrence of the words in XSS attack scripts. Moreover, we also used a large data set to decrease the deviation of the data. Furthermore, we evaluated the classification results with two procedures. One is an inappropriate procedure which some researchers tend to select by mistake. The other is an appropriate procedure which can be applied to an attack detection filter in the real environment.**

I. Introduction

Cross Site Scripting (XSS) attack is one of the attacks on the web. XSS causes information leaking and incorrect access to a web site with HTTP cookies. One of the usual effective countermeasures against XSS is escapement to specific letters not to make script codes. However, some of web site designers and web administrators do not understand the meaning of the escapement and the reason of the prevention of XSS. Moreover, XSS code can be inserted to a web page when a web server or a web system is vulnerable. Especially, unknown attacks to a new vulnerability are difficult to be prevented.

There are existing researches for detecting XSS attacks. Pattern matching is one of the usual methods for the detection of attacks and malware, and it has been applied to XSS attack detection. Also, machine learning becomes popular to detect attacks. Especially, URL type attacks can be classified by machine learning methods with high percentage. However, both malicious and benign samples can be classified by mistake in machine learning, especially deep learning, while benign codes which do not include the specific malicious patterns are never classified into attacks by pattern matching method. In existing researches, there are more than one percent misclassified samples. Therefore, we improved the preprocessing method for training samples and increased the accuracy of the classification. Moreover, we introduce an inappropriate case for the classification of XSS attacks with machine learning, and explain the reason.

II. Related Works

We introduce existing researches to prevent XSS attacks in this chapter.

A. Automatic Classification of XSS Attacks

Kaiho et al. proposed a method for automatic classification of XSS attacks [1]. They focused on characteristic letters which are included in XSS scripts in order to classify XSS scripts from benign scripts. In a research by Matsuda et al., it is described that 32 types letters are often used in XSS scripts [2]. However, Kaiho et al. found that the 32 types letters are also used for usual inputting, so they chose four types letters which are only found in XSS scripts among the 32 types letters. In their method, the four types letters are extracted from string data to calculate the evaluation score. The data are classified in two classes with the evaluation score and the threshold value is predetermined. They classified 350 samples and tested with 5-fold cross validation. The highest accuracy in their experiments was 98%.

We think there are two problems. One is the classification accuracy. The misclassification rate with two percent is too high for actual use. Also, in the two percent, both of benign samples misclassified to malicious and malicious samples misclassified to benign are included. Especially, when benign samples are misclassified to malicious, users hesitate to use it. Moreover, the highest score in their experiments was the 98%, so the percentage may decreases in the actual use. The other problem is that users must set the threshold value by themselves although the threshold value in their experiments is decided by the results of the classification. That is, the threshold value must be set after everyone knows which sample is malicious and which sample is benign. It shows that their method cannot be applied to unknown XSS attacks which are newly created. If the old threshold value by the past experiments is used for the new unknown attacks, the classification accuracy will extremely decrease.

B. XSS Attack Classification with Machine Learning

Unknown XSS attacks cannot be detected with some rate with the method by Kaiho et al. as described in section II-A.

978-1-7281-1151-3/19/$31.00 ©2019 IEEE

In this section, we introduce a method to detect XSS scripts with machine learning.

Nunan et al. classified samples in two classes whether XSS scripts or not by Support Vector Machine (SVM) and Naive Bayes (NB) [3]. Obfuscated code, URL length, the number of domains, duplicated special characters, keywords and HTML/JavaScript Schemes are used as features. They used weka for the preprocessing and the classification of data. The number of samples with XSS attacks was 15,366. Also, 57,207 samples of Dmoz data and 158,847 samples of ClueWeb data were used as benign samples. They evaluated their method with the average of 10-fold cross validation.

As results of their evaluation, the accuracy of Dmoz data with NB was 98.54% and that of ClueWeb data with NB was 99.70%. Also, the accuracy of Dmoz data with SVM was 98.58% and that of ClueWeb data with SVM was 99.89%. The number of samples of ClueWeb was larger than that of Dmoz so that the accuracy of ClueWeb was higher than that of Dmoz. The accuracy 99.89% was the highest rate among those in related works.

However, there are problems. One is ratio in the dataset. The number of malicious samples was 15,366 although the number of benign samples was 158,847 in ClueWeb data. The ratio was 1 : 10.3. The dataset with that ratio is called imbalanced dataset. According to a research by He et al. [4], it is indicated that the performance extremely decreases when an imbalanced dataset is used. They also mention that penalty for misclassification must be adjusted or amount of data must be balanced when an imbalanced dataset is used. However, Nunan et al. did not apply those measures. Another problem is the preprocessing for the classification. Nunan et al. chose Obfuscated code, URL length, the number of domains, duplicated special characters, keywords and HTML/JavaScript schemes as features. However, when attackers know the features, they can adjust some of these features in XSS scripts in order to be classified in benign. For example, URL length can be easily adjusted to be far from the average length of existing XSS scripts. The number of domains also can be camouflaged by IP address or free short domain names, while Nunan et al. mention that this feature corresponds to the number of domains found in the URL. We also think that some of other parameters are not appropriate for machine learning. For example, duplicated special characters such as "<sCrIpt>" and HTML/JavaScript schemes such as "<script>", "<iframe>" and "write()" can be found by usual pattern matching. Moreover, some keywords are also inappropriate. For example, XSS, banking, spray and evil can be intentionally avoided while root, password, crypt and shell may be required for making a specific attack.

Umehara et al. also classified samples by focusing on some special characters which are included in XSS attacks [5]. They used SVM and Exact Soft Confidence-Weighted Learning (SCW) for learning. They put 500 XSS attack samples and 500 benign samples in a dataset. The samples were transformed to four dimensional feature vectors and inputted. The first group and the second group in the feature vectors consist of special characters, and the characters in the first group more frequently appear than those in the second group. The third group consists of numeric values, and the fourth group consists of alphabets. SVM was implemented with scikit-learn0.15.2 which is a machine learning library for Python. Gaussian kernel and linear kernel were also implemented. SCW was implemented based on the method by Wang et al. [6]. In training and testing, 5-fold cross validation was used. The values for the evaluation were the average values of five times tests.

The highest accuracy for their classification with SVM was 95.3% with Gaussian kernel. The accuracy with SCW was 76.4%. When the dimensions decreased to three in their experiments, the accuracy also decreased to 53.0%. It means that the larger the number of dimensions is the more the accuracy increases. The problem of their method is the low percentage of the accuracy.

Umehara et al. also classified samples by SVM, SCW and Random Forest (RF) [7]. They transformed data into 128 vectors. They evaluated their method with five dimensional feature vectors dataset and 128 dimensional feature vectors dataset. The five dimensional feature vectors consist of five groups. The first group consists of special characters which are often used in correct inputting. The second group consists of special characters which are used more than twice in XSS. The third group consists of alphabets, the fourth group consists of numeric values and the fifth group consists of other characters. The algorithms for machine learning were the same as those in their past research, and Gaussian kernel was used for SVM. RF was implemented with scikit-learn0.15.2 which is a machine learning library for Python. In training and testing, 5-fold cross validation was used. The values for the evaluation were the average values of five times tests.

The highest accuracy in their classification of five dimensional feature vectors dataset with SVM was 97.7%. On the other hand, the highest accuracy of 128 dimensional feature vectors dataset with SVM was 98.9%. The highest accuracy in their classification of five dimensional feature vectors dataset with RF was 97.6%. On the other hand, the highest accuracy of 128 dimensional feature vectors dataset with RF was 99.2%. The highest accuracy in their classification of five dimensional feature vectors dataset with SCW was 66.7%. On the other hand, the highest accuracy of 128 dimensional feature vectors dataset with SCW was 89.8%. They proved that the more dimensional vectors a dataset had the more the accuracy increased. Also, they indicated that SCW was not suitable for XSS detection.

We think the problem of their method is an algorithm for the preprocessing. In both of their two papers, they managed letters just with frequency of appearance. For example, in their vectorization algorithm, string "aabb" had the same vector as string "abab" since each letter appeared with the same frequency. We improved the preprocessing algorithm to increase the accuracy of classification.

III. Proposed Method

A. Outline of Proposed Method

In existing researches, algorithms for machine learning and classifiers are compared. Accuracy is also evaluated to find the best method to classify XSS attacks. Also, in preprocessing for the classification, frequency of appearance of special characters and URL length are extracted as features. The best accuracy in those researches was 99.89%. On the other hand, we found problems of the methods in those researches. Therefore, we improved the methods to increase the accuracy of classification.

One of our ideas is using the combination of classifiers. In existing researches, classifiers were used alone. On the other hand, we use the classifiers as a double stage filter. The other idea is that we manage an XSS script as a natural sentence with meaning. The sentence is divided into words and the frequency of appearance and co-occurrence of the words are compared in the classification. To implement it, we used word2vec which enables the vectorization by the investigation of words in a sentence and a sentence of words. We think that XSS like grammars can be distinguished from not XSS like grammars when XSS scripts are managed as sentences in natural languages.

We also implemented the double stage filter as CNN + NB or CNN + SVM. We think that CNN is the best classifier for XSS classification since it is usually used for the classification of sentences in natural languages, and is often used with word2vec. The reason why we used NB and SVM is they are used in existing researches. We chose Multinomial Naive Bayes (MNB) in NB since MNB is often used for the classification of sentences in natural languages. We evaluated whether our method is the best for both preprocessing and classifying. The flow of our method is as follows; data collection; preprocessing; training of classifier; test with CNN; test with MNB or SVM; evaluation.

B. Data Collection

We used scripts in XSSed [8] as malicious scripts. We collected benign scripts with both GET and POST methods by an online web page crawler [9]. Collected samples were stored in CSV files.

C. Preprocessing

The sequence of words must be transformed to vectors before machine learning. We used word2vec for the transformation. On the other hand, MNB cannot manage samples with negative values. Therefore, frequency of appearance of words was only used for generating vectors for MNB. Thus, we mainly used positions and frequency of words in attack codes for the classification.

The flow of the preprocessing is as follows. First, we separated XSS attack data into words. All of the attack data are URL formed. Therefore, the data can be divided with '/', '=', '.', '<' and '>'. For example, 'aaa/bbb' is divided into 'aaa', '/' and 'bbb'. Next, X kinds of words which frequently appear are chosen. The value X is predetermined. The other words are set together to 'other'. The words are transformed to values. The value of a word which the most frequently appears is set to X + 1. The second word is set to X. The third word is set to X - 1, and the transformation continues to Xth word with the same algorithm. Finally, the other words 'other' are set to 1. Thus, all words are transformed to integer values.

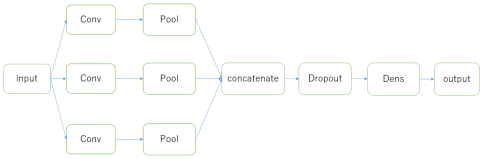

Fig. 1. Construction of CNN.

D. Training with word2vec

We transformed words in XSS scripts to vectors with word2vec. word2vec is an algorithm which calculates co-occurrence of words around a word. XSS scripts are transformed to vectors to be inputted in CNN. On the other hand, the vectors cannot be used for MNB. We used Word2Vec function in gensim library. The first argument of the function is dataset and the second argument is the number of embedding dimensions. The dataset was words in XSS scripts. The output of the Word2Vec function is a trained model. We saved the model for the use with CNN.

E. Preprocessing for CNN

Data for CNN are vectorized by word2vec as mentioned in section III-D. Words in a dataset are transformed to integer values as mentioned in section III-C. Therefore, the dataset consists of one dimensional integer arrays. The maximum length of the arrays 'maxlen' is acquired in order to fix the length of the arrays. When the length of an array is smaller than maxlen, elements which are set to 0 are added to the array until the length is equal to maxlen. After that, all arrays are vectorized with the trained model of word2vec. The words in the dataset are inputted in model.wv function as an argument. The output of model.wv function is one dimensional arrays which length is the same as the number of embedding dimensions.

Next, we set labels on samples of the data. The label is a list type one dimensional array and the values are [0 1] or [1 0]. We put [0 1] on XSS attack samples and [1 0] on benign samples. We divided the whole samples into k numbers. At this time, samples were randomly chosen.

F. Training with CNN

We have not known any research in which XSS attacks are classified with CNN. Therefore, we referenced a research by Kim [10] in which CNN is used for classification of documents. The construction of CNN in our method is shown in Fig. 1.

In Fig. 1, inputted samples are divided into three channels. We inputted k-1 numbers of groups among k grouped samples

as described in section III-E. The remain is used for the test. After finishing the test, another k-1 numbers of groups are chosen for the next training.

G. Tests for Classification

Tests for classification are executed with CNN, MNB and SVM. Test samples which are not used for training are used for the test. First, test samples are inputted in a trained model of CNN and classified. The evaluation values and misclassified samples are recorded. Next, test samples are inputted in MNB and SVM and evaluated. Finally, misclassified samples of CNN are inputted in MNB and SVM and evaluated.

IV. Implementation

In this chapter, we explain the implementation of data preprocessing and classifiers described in chapter III. All of the implementation has done in python3.6.5 environment.

A. Implementation of data preprocessing

We explain the implementation of data preprocessing described in section III-C. XSS attacks samples and benign samples are extracted from CSV files described in section III-B. We implemented it with CSV library. csv.reader function was used for reading CSV files and the arguments were file path of the CSV files and 'r'. The read data was stored in a list type valuable X. The valuable X is one dimension array and the length is the number of samples. The elements of X are str type URLs. One element in X was extracted and divided into words by regexp_tokenize function whose arguments were delimiter strings and division rules described in section III-A. All elements were processed one by one with the same procedures. After that, all words were transformed to integer values described in section III-C. A correspondence table for the transformation was implemented with python dictionary. Integer values were set in key attribute and original words were set in value attribute in the table. Words which did not appear in the table were transformed to 1 as 'other'. After the transformation, the length of arrays was fixed to the same number by zero padding with pad_sequences function whose argument was a dataset and whose output was fixed arrays. In our experiments, the maximum length in a dataset was 598, so the length of all arrays was set to 598 by zero padding.

B. Implementation of word2vec Training

We implemented word2vec training described in section III-D with Word2Vec function in gensim3.4.0 library. The arguments of Word2Vec function were a dataset, the number of embedding dimensions and a model. The dataset consisted of words from XSS scripts. A parameter sg was set to 1 to use skig-gram model. The output was a trained model.

C. Implementation of Preprocessing for CNN

Words were transformed to vectors with trained word2vec model described in section III-D. A dataset was set in the argument of model.mv function for creating training samples for CNN.

D. Implementation of CNN and Training

CNN described in section III-F was implemented by referencing a research by Kim. The construction of layers and the values of hyper parameters were the same as those in the research. For the implementation, we used a deep learning library keras2.0.8 and tensorflow1.8.0, and also used numpy1.13.3 library for matrix operations. The network of implemented CNN consisted of input layer, convolution layer, pooling layer, full-connected layer, hidden layer and output layer. The activation function in convolution layer was relu. The pooling layer was implemented with maxPooling1D function. Dropout function with argument 0.5 was also used in the layer to suppress over learning. The hidden layer was implemented with Dense function with arguments of the number of hidden layers '300' and activation function 'relu'. The output layer was implemented with Dense function with arguments of the number of output classes '2' and activation function 'softmax'. Finally, a training method was set with compile function with arguments of 'binary_crossentropy' and 'adadelta'. The arguments indicated that it was two classes classification and adadelta was used for optimization of parameters. For the training of CNN, model.fit function was used with arguments of a dataset, label, batch size and the number of epochs. The output of the function was a trained model which was saved by model.save function with an argument of a file name for the saved file.

E. Implementation of MNB and SVM

We implemented MNB and SVM with sklearn0.19.1 library. Training and test were done with preprocessed data described in section III-C. To construct a model, MultinorimalNB function without argument and LinearSVC function without argument were used. For training, fit function with arguments of traning data and labels was used. The output of the fit function was a trained model. Those above were used for the implementation of MNB and SVM.

F. Implementation of Classification Program

First, trained CNN model was loaded by load_model function with an argument of the path for the trained model file. Next, tests were done by predict function with arguments of test data and batch size. The output of the predict function was a predict label.

V. Preliminary Experiment

Before using CNN, we decided parameters by a preliminary experiment. What we decided by the preliminary experiment are 'skip-gram model or cbow model' and the number of epochs in CNN. Implementation for the preliminary experiment is the same as that for other experiments described in chapter IV. skip-gram can be used when the argument of Word2Vec function is set to 'sg=1'. We trained a dataset to 100 epochs with skip-gram and cbow, and the result is shown in Fig. 2.

In Fig. 2, the vertical axis shows accuracy (accuracy value in machine learning) and the horizontal axis shows the number

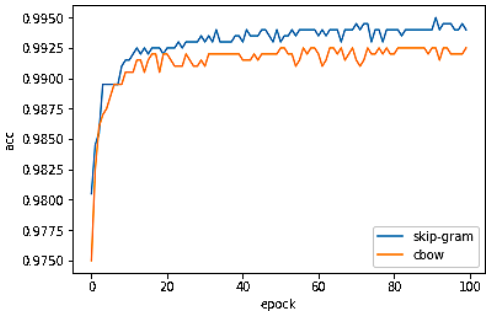

Fig. 2. Comparison of accuracy with skip-gram and cbow.

of epochs. The result of skip-gram is shown with blue line and that of cbow is shown with red line. The accuracy of skip-gram reaches to the maximum value 0.9950 when the number of epochs is 91. On the other hand, the accuracy of cbow reaches to the maximum value 0.9925 when the number of epochs is 50. When the number of epochs is 7 or 8, the accuracy of both methods is equal. When the number of epochs is 3, the accuracy of cbow is higher than that of skip-gram. In other cases, the accuracy of skip-gram is always higher than that of cbow. Therefore, we decided to choose skip-gram and to train to 100 epochs.

VI. Evaluation

We evaluated our method in two ways to clarify a problem when security engineers try to use machine learning. The problem is mentioned later in this chapter.

A. Evaluation Method

The classification result of test samples mentioned in section III-G is evaluated as follows. Benign and malicious samples are classified with two classes classification of CNN, MNB and SVM. At this time, the same combination of training samples is inputted in CNN, MNB and SVM. After the training, trained models of CNN, MNB and SVM are evaluated with test samples which are not used for training. In addition, misclassified test samples in CNN are inputted again in MNB and SVM. Therefore, finally we get results of CNN, MNB, SVM, CNN+MNB and CNN+SVM. Indicators of the evaluation results are accuracy, precision, recall and f-score. In this chapter, "accuracy" means accuracy in machine learning. That is, the accuracy is a rate of samples which are correctly classified. We put positive labels on malicious samples and negative labels on benign samples since the objective of this classification is the detection of XSS attacks. The results values are acquired from tests with 5-fold cross validation, and the average values are used for the final evaluation.

B. Creation of Dictionary

Words in a dataset are related with integer numbers as described in section III-C. XSS attack scripts for the experiments consist of about 100,000 words with 21,529 kinds of words. We think that to assign numbers on every kind of words is meaningless since words with low frequent appearance like only once do not contain characteristics of XSS attacks. Therefore, we decided to use words which have 5 or more number of appearance since the average of the number of appearance of all words is 4.77. In conclusion, 1451 kinds of words were registered in our dictionary for the experiments, and remaining words were also registered as 'other' as shown in Table I.

TABLE I
Keys and Values Registered in Dictionary

key	value
1452	The most frequent word
⋮	⋮
2	word with 5 times appearance
1	'other'

C. Parameters for word2vec

Parameters which we decide are models and the number of embedding dimensions. The models for word2vec are skip-gram and cbow and we chose skip-gram as described in chapter V. Also, we decided the number of embedding dimensions as follows. Trained word2vec by Google [11] has 3,000,000 words and the number of embedding dimensions is set to 300. Therefore, we chose 10 for the number of embedding dimensions since we have a dataset with about 100,000 words. Actually, we set 12 to the number of embedding dimensions since the number must be a multiple of 3 based on the output filter size of convolution layer in CNN.

D. Parameters for CNN

Parameters for CNN are the same as those in a research by Kim as described in section III-F. However, we changed some parameters since our dataset is different from Kim's dataset. In the research by Kim, the number of embedding dimensions is 300 and output filter size is 100. However, we set the output filter size to 4 since we chose 12 for the number of embedding dimensions as described in section VI-C, and 4 is 1/3 of 12. Basic parameters for CNN in our experiments are as follows; filters: 3; kernel size: 3, 4, 5; pooling size: 3, 4, 5; dropout size: 0.5; hidden layer: 300. Also, batch size is set to 50 and the number of epochs is set to 100 based on the preliminary experiment in chapter V.

E. Division of Test Samples

For the tests, k-fold cross validation is used, and the result value is the average value of k times tests. In the related works in chapter II, k is set to 5 in three researches. Therefore, we chose 5 for k. There are 10,000 samples in our dataset, so the number of test samples in one test is 2,000 since 2,000 is 1/5 of 10,000.

F. Evaluation Results with Inappropriate Knowledge

Evaluation results are shown in Table II and standard deviation values are also shown in Table III. We explain what is inappropriate in section VI-G.

TABLE II
RESULTS WITH INAPPROPRIATE KNOWLEDGE

	accuracy	precision	recall	f1-score
CNN	0.9921	0.9979	0.9861	0.9920
MNB	0.8411	0.8608	0.8308	0.8545
SVM	0.8688	0.9206	0.8155	0.8624
CNN+MNB	0.9940	0.9996	0.9883	0.9939
CNN+SVM	0.9937	0.9978	0.9886	0.9936

TABLE III
STANDARD DEVIATION RESULTS WITH INAPPROPRIATE KNOWLEDGE

	accuracy	precision	recall	f1-score
CNN	0.0041	0.0055	0.0034	0.0040
MNB	0.0057	0.0056	0.0120	0.0122
SVM	0.0130	0.0145	0.0064	0.0077
CNN+MNB	0.0015	0.0013	0.0017	0.0008
CNN+SVM	0.0016	0.0015	0.0010	0.0007

TABLE IV
RESULTS WITH APPROPRIATE KNOWLEDGE

	accuracy	precision	recall	f1-score
CNN	0.9928	0.9980	0.9875	0.9927
MNB	0.6944	0.6513	0.8370	0.7325
SVM	0.9928	0.9980	0.9875	0.9927
CNN+MNB	0.9976	0.9986	0.9966	0.9976
CNN+SVM	0.9963	0.9984	0.9942	0.9963

G. Evaluation Results with Appropriate Knowledge

The results in section VI-F are inappropriate and security engineers sometimes make the same mistake. If an actual filter is created like that, most of new XSS attacks will not be detected in a real environment. IP address and Fully Qualified Domain Name (FQDN) are included in both of malicious and benign samples, and the words in them are also transformed to integer values according to a table in a dictionary. IP address and FQDN in malicious samples are real values in actual attacks so that they take specific addresses or domains which are not included in benign samples. In addition, new attacks which will come in future do not contain the same IP address and the same FQDN as those in training samples.

Therefore, we eliminated the influence of IP address and FQDN from training of machine learning. We changed all IP addresses in all samples to 192.168.1.1 and all FQDNs in all samples to aaa.bbb.ccc.com so that they no longer influence the results. The evaluation results are shown in Table IV and standard deviation values are also shown in Table V.

VII. CONSIDERATION

In the result of CNN and SVM, both of accuracy and f1-score increased with appropriate knowledge as shown in Table II and IV. The accuracy and f1-score of CNN increased 0.07% The accuracy and f1-score of SVM increased 13% On the other hand, in the result of MNB, the accuracy decreased 15%.

TABLE V
STANDARD DEVIATION RESULTS WITH APPROPRIATE KNOWLEDGE

	accuracy	precision	recall	f1-score
CNN	0.0023	0.0017	0.0036	0.0025
MNB	0.0131	0.0164	0.0043	0.0112
SVM	0.0023	0.0017	0.0036	0.0025
CNN+MNB	0.0007	0.0007	0.0017	0.0007
CNN+SVM	0.0013	0.0004	0.0024	0.0013

It means that CNN and SVM are suitable for filters in a real environment. On the other hand, MNB highly depends on the knowledge of IP address and FQDN. Note that, the increase in CNN is ignorable according to the standard deviation in Table III and V.

VIII. CONCLUSION

In this paper, we distinguished XSS attacks from benign samples with machine learning. As a result, our classification method marks higher score than that in existing researches since preprocessing method before training is better than that in existing researches. On the other hand, we pointed out inappropriate training which security engineers sometimes do. Finally, we proved that our method is useful in a real environment.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP18K11248.

REFERENCES

[1] T. Kaiho, T. Matsuda, M. Sonoda and J. Chao, *Feature Extraction of Embedded URL Cross-Site Scripting Attacks*, The 77th National Convention of IPSJ, Vol.1, pp.427–428, 2015 (Japanese).
[2] T. Matsuda, D. Koizumi and M. Sonoda, *Cross Site Scripting Attacks Detection Algorithm Based on the Appearance Position of Characters*, The 5th International Conference on Communications, Computers and Applications (MIC-CCA2012), pp.65–70, 2012.
[3] A. E. Nunan, E. Souto, E. M. dos Santos and E. Feitosa, *Automatic Classification of Cross-Site Scripting in Web Pages Using Document-based and URL-based Features*, IEEE Symposium on Computers and Communications (ISCC), pp.702–707, 2012.
[4] H. He and E. A. Garcia,
Learning from Imbalanced Data, IEEE Transactions on Knowledge and Data Engineering, Vol.21, No.9, pp.1263–1284, 2009.
[5] A. Umehara, T. Matsuda, M. Sonoda, S. Mizuno and J. Chao, *Consideration on the Cross-Site Scripting Attacks Detection Using Linear Classifiers*, IEICE and IPSJ, The 14th Forum on Information Technology (FIT), Vol.1, pp.155–156, 2015 (Japanese).
[6] J. Wang, P. Zhao and S. C. H. Hoi, *Exact Soft Confidence-Weighted Learning*, The 29th International Conference on Machine Learning (ICML), 2012.
[7] A. Umehara, T. Matsuda, M. Sonoda, S. Mizuno and J. Chao, *Consideration on the Cross-Site Scripting Attacks Detection Using Machine Learning*, IPSJ SIG Technical Report, Vol.2015-CSEC-71, No.13, pp.1–4, 2015 (Japanese).
[8] XSSed — Cross Site Scripting (XSS) attacks information and archive, http://www.xssed.com
[9] Page Crawler (Japanese), http://tshinobu.com/lab/get-page-link/
[10] Y. Kim, *Convolutional Neural Networks for Sentence Classification*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp.1746–1751, 2014.
[11] Google Code Archive - Long-term storage for Google Code Project Hosting, https://code.google.com/archive/p/word2vec/