An Evaluation of Word Embeddings on Vulnerability Prediction with Software Metrics

Sousuke Amasaki, Tomoyuki Yokogawa
Department of Systems Engineering
Okayama Prefectural University
Soja, Japan
{amasaki,t-yokoga}@cse.oka-pu.ac.jp

Hirohisa Aman
Center for Information Technology
Ehime University
Matsuyama, Japan
aman@cse.oka-pu.ac.jp

Abstract—**CONTEXT: Software vulnerability is a crucial risk for a digital world. Developers dedicate enormous effort to removing vulnerable code from their software products. Vulnerability prediction aims to spot which modules are more vulnerable using software metrics. Recent studies conducted empirical experiments using textual information and software metrics. The result showed that the textual information did not help improve the predictive performance. However, their evaluations only considered Bag-of-Words (BoW) as textual information, and semantic relations among words have never been examined. OBJECTIVE: To examine the performance of vulnerability prediction with textual information considering semantic relations. Word2Vec was employed for capturing semantic relations. METHOD: A comparative study among BoW and two Word2Vec embeddings was conducted. For easy evaluation, we replicated a recent study that employed BoW. The Word2Vec embeddings were obtained from pre-trained models based on Google News and Stack Overflow. The former used large but non-SE-related texts, while the latter used small but SE-related texts. RESULTS: The non-SE Word2Vec improved vulnerability prediction in term of prediction stability. The SE-specific Word2Vec was less effective. CONCLUSION: Practitioners should consider textual information with non-SE Word2Vec for better vulnerability prediction.**

Keywords-component; vulnerability prediction; word embeddings; empirical study

I. INTRODUCTION

Software vulnerability prediction is a sort of defect prediction that uses software metrics such as complexity measures without analyzing program syntax and semantics. Contrastingly, vulnerable code pattern recognition analyzes program syntax and semantics to obtain textual and/or structural information from source code and construct prediction (recognition) models. These two categories use supervised/unsupervised learning algorithms, and the information of those categories can be combined for better prediction and recognition.

A recent study [1] conducted an empirical experiment that combined software metrics and textual information and used them for vulnerability prediction. The result showed that their textual information was seldom helpful in improving the predictive performance. However, they used classical Bag-of-Words (BoW) for textual information, and modern techniques that can consider semantic relations among words have not been employed for evaluation.

In this study, we investigated the effectiveness of a modern technique for textual information for vulnerability prediction. Our empirical study employed a replication kit of [1] that combined process and product metrics and BoW for vulnerability prediction. We used Word2Vec [2] as a modern technique for textual information instead of BoW. Word2Vec is a method to learn quality distributed representations for words. Words are represented as multi-dimensional vectors, and similar words are placed close together in that dimensional space. Our study used two pre-trained models from SE and non-SE texts to obtain Word2Vec vectors for evaluation.

II. METHODOLOGY

We investigated three research questions. The questions and their motivations are described in the following subsections.

A. Datasets

This study used the datasets provided through a replication package of a past study [1]. The datasets were collected from 9 Java projects. We used the same sample of 8,991 commits among 56,286 for evaluation. For each commit, 24 product metrics and 9 process metrics were collected. Their details such as definitions are in [1]. Also, our study collected BoW and Word2Vec vectors for evaluation.

B. Word2Vec-based features

This study defined two types of Word2Vec-based vectors as well as BoW-based vectors in [1]. One vector is the sum of Word2Vec vectors of the files in a commit. Each file modified in a commit was parsed to extract identifiers. Here, camel cases and snake cases were divided into individual words. These words were lowered and converted to Word2Vec vectors. The sum of the vectors was used as the vector of the file. The same procedure was used to make Word2Vec vectors of added lines and deleted lines of a commit. The other vector is the absolute difference between the Word2Vec vectors of added lines and deleted lines. Finally, the two types of vectors were concatenated to make Word2Vec-based features.

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

In the above procedure, we used pre-trained Word2Vec models to convert words to vectors. Pre-trained models are based on a large amount of training data and are expected to learn many words and their relations that would not be able to realize from a small amount of data of 8,991 commits and the modified files in those commits. For that purpose, we prepared two pre-trained Word2Vec models.

The first pre-trained mode was trained with Google New dataset. The training data was obtained from non-SE documents, and the similarity among words differs from that of software engineering documents. The vector size is 300, and the vocabulary size is approximately 3 million words. The second model [3] was trained with 15GB of Stack Overflow posts (i.e., SE documents) collected from August 2008 to December 2017. All code snippets were removed before training, and natural language terms only appeared in the textual data. Although some symbols such as `+' are often used in a software engineering context (e.g., programming), we did not consider those terms as identifiers in Java code does not include them. The pre-trained model was trained with 1.7 million keywords. The vector size is 200, smaller than the first model.

C. Experimental Design

As noted in [1], the number of vulnerabilities is too small in some projects to conduct neither cross-validation nor a time-sensitive approach. Therefore, cross-project training was adopted. The cross-project training of N projects uses N-1 project data for training and predicts the remaining project. This experiment design finally made N predictive results.

D. Prediction Models

We used the same prediction models as [1], namely, SVM, KNN, Decision Tree, Random Forest, Extremely Randomized Trees, AdaBoost, Gradient Boosting, and XGBoost with hyper-parameter tuning.

E. Performance Measure

AUC (Area Under the Receiver Operating Characteristic Curve) was adopted for evaluation as well as [1]. AUC measure is not sensitive to thresholds that determine faultiness or not. It is also robust to class imbalance, which happens in vulnerability prediction. A classifier is perfect if an AUC value is 1.0. A meaningful classifier results in an AUC value of more than 0.5. If equal or less, the classifier is random guessing at best.

III. Research Questions

We investigated the following research questions. The questions and their motivations are described in the following subsections.

A. Effectiveness of Non-SE Word2Vec against BoW

Although the merits of word embeddings were confirmed in some software engineering tasks, past studies such as [1] on vulnerability prediction have adopted BoW only. The pre-trained model based on the Google News dataset was the most popular one and was suited to a baseline for evaluation. This research question aimed to answer whether using such a general-purpose pre-trained model is effective for vulnerability prediction compared to using BoW.

B. Effectiveness of SE-Specific Word2Vec against BoW

Efstathiou et al. [3] examined the effectiveness of domain-specific word2vec models compared to non-SE models. They demonstrated that the similarity of technical words was often different from general words. For example, “cookie” was like “session” in software engineering while it was to “cupcake” in general. This difference in specialty might also affect the predictive performance of vulnerability prediction. Therefore, this research question aimed to answer whether using a domain-specific model is effective for vulnerability prediction.

C. Contribution of Word2Vec Features on Vulnerability Prediction with Software Metrics

Two studies [1][4] introduced BoW into conventional vulnerability prediction based on software metrics. One of their interests was the effectiveness of textual information though they did not investigate other representations of textual information such as Word2Vec. Therefore, this research question aimed to answer whether word embeddings could add value for vulnerability prediction with software metrics.

IV. Results

This section answers the three research questions posed in Section III. The results were also discussed.

A. Effectiveness of Non-SE Word2Vec against BoW

Fig. 1 represents the result of cross-project validation with the prediction methods we employed. There are three groups corresponding to the types of textual information, namely, BoW, non-SE Word2Vec, and SE-specific Word2Vec. Each boxplot in each group represents the predictive performance of vulnerability prediction using a prediction method we employed. To answer RQ1, this subsection discussed the differences in predictive performance between BoW (the leftmost group) and the non-SE Word2Vec (the middle group).

The leftmost group in the figure showed the AUC of the prediction methods using BoW. It was observed that the prediction methods affected the predictive performance. Decision Tree, Random Forest, Extra Trees, and XGBoost were lesser than random guessing with AUC values of almost less than 0.5. SVM and Gradient Boost were better than random guessing with mean and median values of more than 0.6. KNN showed higher performance with AUC values around 0.7. AdaBoost was the best performer in terms of mean and median values.

The middle group in Fig. 1 showed the AUC of the prediction methods using the non-SE Word2Vec. The predictive performance of the prediction methods in this group was also diverse but showed slightly different trends from the leftmost group. While Decision Tree and XGBoost were still lesser than random guessing, Random Forest and Extra Trees performed better than those with BoW. The boxplot of SVM was not so different from that of SVM in the leftmost group. However, their mean and median values were improved with non-SE word embeddings. Gradient Boost and KNN were also improved with

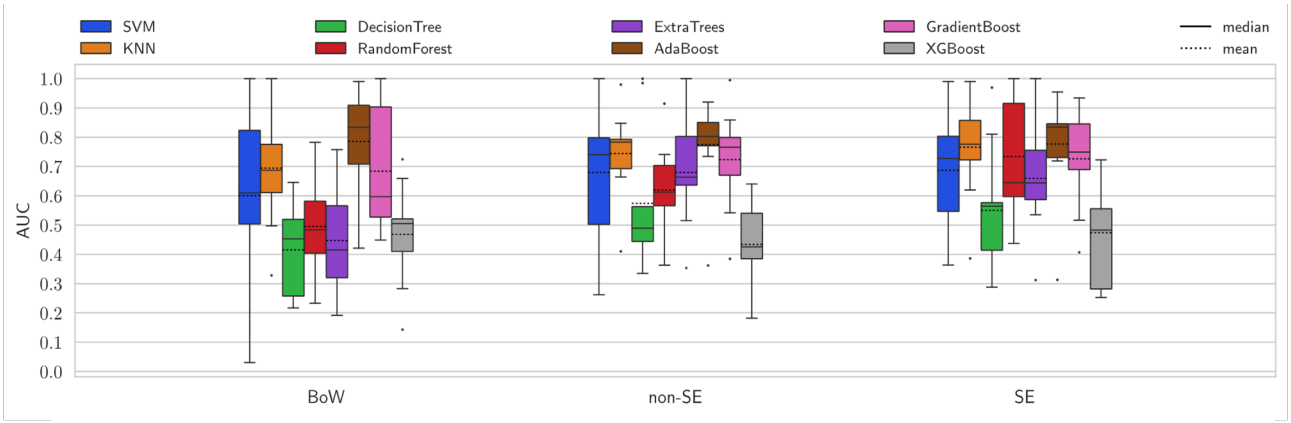

Figure 1. The predictive performance of vulnerability prediction methods with textual information

the use of the non-SE Word2Vec. Their boxplots got shorter than those with BoW. Although AdaBoost was still the best performer in terms of mean and median values, using the non-SE Wor2Vec worsened the median value compared to BoW. These observations implied that using the non-SE word embeddings was supported in terms of AUC. While the best performer in terms of median was AdaBoost with BoW, AdaBoost with non-SE Word2Vec showed good and stable performance.

In summary, using the non-SE Word2Vec was better than using BoW for Random Forest, Extra Trees, Gradient Boost, and KNN. The best performer in terms of median is AdaBoost with BoW while the statistical test did not reject the difference against AdaBoost with non-SE Word2Vec. The advantage of non-SE Word2Vec was in the stability represented by the smaller boxplot.

B. *Identify the Headings Effectiveness of SE-Specific Word2Vec against BoW*

To answer RQ2, the differences in predictive performance between the non-SE Word2Vec and the SE-specific Word2Vec were discussed in this subsection. The rightmost group in Fig. 1 showed the AUC of the prediction methods using the SE-specific Word2Vec. The predictive performance of the prediction methods in this group was also diverse but showed different trends from the middle group. The boxplots of Decision Tree and XGBoost crossed the line of random guessing. However, the mean and median of Decision Tree were improved and got beyond 0.5. Random Forest also showed better results regarding the upper side of the boxplot and the mean value. Extra Trees with the SE-specific Word2Vec showed slightly lower performance than that with the non-SE Word2Vec. The performance of SVM, KNN, AdaBoost, and Gradient Boost was stable. No clear effect of using SE-specific Word2Vec was observed. AdaBoost with the SE-specific Word2Vec was competitive with that with the non-SE Word2Vec.

In summary, using SE-specific word embedding was also a better option than using non-SE word embeddings for Decision Tree, Random Forest, and AdaBoost. Although the best performer was AdaBoost using the SE-specific Word2Vec, it was not clearly supported.

C. *Contribution of Word2Vec Features on Vulnerability Prediction with Software Metrics*

Fig. 2 showed boxplots of the AUC of the prediction models. The leftmost group in Fig. 2 showed the AUC of the prediction methods using BoW. In comparison to Fig. 1, adding software metrics improved the predictive performance of some prediction methods. The mean and median values of Decision Tree, Random Forest, Extra Trees, Gradient Boost, and XGBoost were improved. The boxplots of them also moved to higher places. The middle group in Fig. 2 showed the AUC of the prediction methods using the non-SE Word2Vec. In comparison to Fig. 1, adding software metrics improved the predictive performance of some prediction methods. The mean and median values of Decision Tree, Random Forest, Extra Trees, AdaBoost, Gradient Boost, and XGBoost were improved. The boxplots of them also moved to higher places. The rightmost group in Fig. 2 showed the AUC of the prediction methods using the SE-specific Word2Vec. In comparison to Fig. 1, adding software metrics improved the predictive performance of some prediction methods. The mean and median values of Decision Tree, Random Forest, Extra Trees, AdaBoost, Gradient Boost, and XGBoost were improved. The boxplots of them also moved to higher places. These observations implied that using word embeddings was supported in terms of AUC for some prediction methods while the statistical test did not reject the difference. Using non-SE Word2vec contributed to stable prediction with competitive performance. AdaBoost with non-SE Word2Vec and software metrics was considered a practically best choice for its stability.

V. Threats to Validity

We investigated the following research questions. The questions and their motivations are described in the following subsections.

A. *Internal Validity*

The main threat is in the implementation of the experiment we conducted. We used the same experimental design as the replication package of [1] to minimize that threat. We only modified some code to use Word2Vec vectors for evaluation. We executed the original experiment with that code, and even

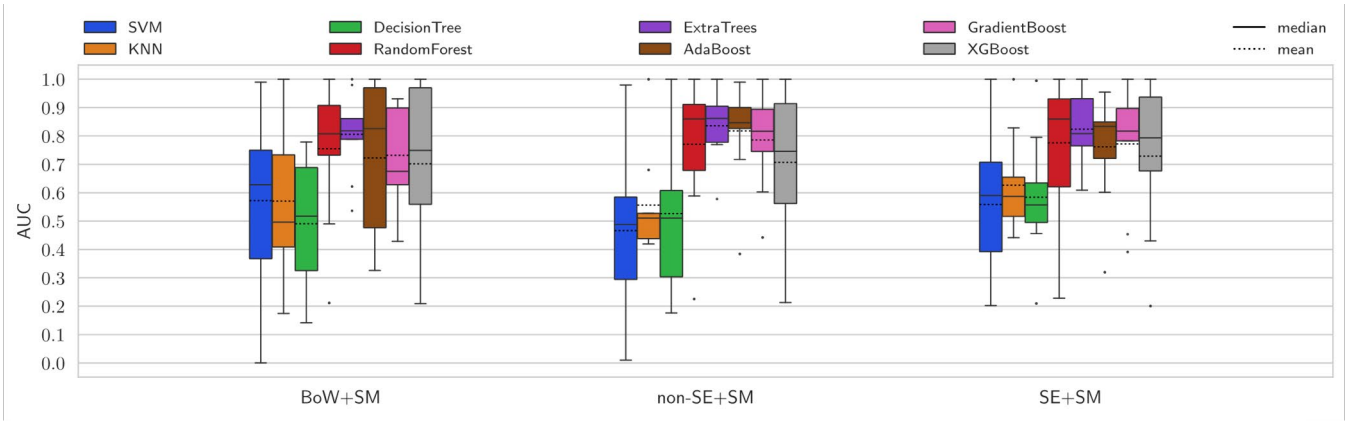

Figure 2 The predictive performance of vulnerability prediction methods with textual information and software metrics

if any bugs existed, we could evaluate the effectiveness of Word2Vec under the same condition as BoW.

B. Construct Validity

The vulnerable contributing commits (VCCs) of the datasets were labeled automatically with the SZZ algorithm. The SZZ algorithm was widely used for defect prediction but was not perfect. Although some mislabeling might thus exist, manual labeling was unrealistic for a large number of commits. For word embeddings, we used two Word2Vec pre-trained models only. The effectiveness of other word embeddings, such as Glove, was still unknown.

C. External Validity

This study used 9 Java datasets only. Although their domain and the statistics, such as commit counts, are diverse, the small number of projects still threats external validity. However, we also think that threats to external validity were not more serious than those of past studies using fewer datasets.

VI. CONCLUSIONS

This study conducted an evaluation of word embeddings on vulnerability prediction with software metrics. Using 9 Java projects with vulnerabilities registered in NVD, two Word2Vec pre-trained models were compared with the classical bag-of-words. The experiment results suggested the use of non-SE Word2Vec model was better than the use of BoW in term of stability. No clear difference in terms of mean was observed. Note that the effect was not found on all the prediction methods.

Our future work includes the use of additional information. Vulnerable code pattern recognition studies have used N-gram of source text in addition to BoW. Furthermore, recent studies have focused on deep learning-based features other than word embeddings. For instance, Chakraborty et al. [5] examined a deep learning-based technique based on graph embeddings and triplet loss with a dataset they collected from C/C++ code. Seeking better features is one of the top priority issues for better vulnerability prediction. An implication for practitioners is that trying to use word embeddings is a good idea for vulnerability prediction.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI Grant #21K11831, #21K11833, and #23K11382.

REFERENCES

[1] F. Lomio, E. Iannone, A. De Lucia, F. Palomba, and V. Lenarduzzi, “Just-in-time software vulnerability detection: Are we there yet?” The Journal of Systems & Software, vol. 188, p. 111283, 2022.

[2] T. Mikolov, K. Chen, G. Cornado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” in Proc. of Workshop at the International Conference on Learning Representations, 2013.

[3] V. Efstathiou, C. Chatzilenas, and D. Spinellis, “Word embeddings for the software engineering domain,” in Proc. of Working Conference on Mining Software Repositories, ser. MSR ’18. ACM, 2018, p. 38–41.

[4] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, “Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits,” in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York, NY, USA: ACM, 2015, p. 426–437.

[5] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, “Deep learning based vulnerability detection: Are we there yet?” IEEE Transactions on Software Engineering, vol. 48, no. 09, pp. 3280–3296, 2022