

A Study on Misclassification of Software Vulnerabilities when using Deep Learning and Machine Learning Algorithms

Vishnu Ramesh¹, Sara Abraham¹, Vinod P², Isham Mohamed¹, Corrado A. Visaggio³ and Sonia Laudanna³

¹SCMS School of Engineering & Technology, Ernakulam, Kerala, Indian

²Cochin University of Science and Technology, Kochi, Kerala, Indian

³Department of Engineering, University of Sannio, Benevento, Italy

Abstract

As the field of computer science has advanced over the years, there has been a tremendous increase in the software being created, and this increase has been accompanied by a growth of software vulnerabilities. A software vulnerability is a security flaw found in software that can potentially be exploited by attackers to perform cyber attacks. Since automatic approaches for identifying and analyzing vulnerabilities has become a trending topic in research community, the classification of vulnerability is still an open issue. Machine and deep learning has been applied as promising approaches for automatically classifying vulnerabilities; unfortunately such methods could produce errors due to misclassification. With this paper we compare five shallow learning models and fourteen deep learning models with the aim of characterizing quantitatively the differences in terms of classification's errors.

Keywords

vulnerability, deep learning, machine learning, malware classification, cybersecurity

1. Introduction

The Skybox's Vulnerability And Threat Trends Report of 2020 [1], highlights a considerable growth of the volume of medium severity vulnerabilities and the related Common Vulnerability Scoring System (CVSS) scores [2]. The report advises that the number of vulnerabilities in one of the most widespread operating systems, Microsoft Windows OS, has increased by 66% in the period between 2018 and 2019. Vulnerability descriptions are collected in publicly accessible repositories, like the Common Vulnerability and Exposures (CVE) [3] database and the National Vulnerability Database¹, after a process of validation and classification. Classification of software security vulnerability facilitates the understanding of security-related information and speeds up the analysis of a vulnerability [4], improving the effectiveness and efficiency of the vulnerability fixing process. It even helps analysts to identify new vulnerabilities. Since vulnerability

ITASEC21: Italian Conference on Cybersecurity, April 07–09, 2021, Online

✉ vishnu.ramesh@scmsgroup.org (V. Ramesh); sara.abraham@scmsgroup.org (S. Abraham); vinod.p@cusat.ac.in (V. P); isham.mohamed@scmsgroup.org (I. Mohamed); visaggio@unisannio.it (C. A. Visaggio); slaudanna@unisannio.it (S. Laudanna)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



CEUR Workshop Proceedings (CEUR-WS.org)

¹<https://nvd.nist.gov/search>

descriptions are written in natural language, automatizing the process of classification is not trivial, while manual classification is a time-consuming process and error-prone. For these reasons, there are some attempts in literature for finding machine and deep learning approaches for classifying vulnerabilities, as will be discussed in the section of related work. Unfortunately, one of the main drawbacks of such an approach is the incidence of misclassification, i.e. the case in which a vulnerability is associated with a wrong class.

For this reason, we built several machine and deep learning classifiers and evaluate the proneness of misclassification of each one, aiming at establishing which are the classifiers that are less affected by misclassification. We built fourteen deep learning models and five shallow learning models and test them on two datasets, one with duplicates and the other one without duplicates. The experiment allowed us to quantify the impact of misclassification in each classifier, in order to provide research and industry community evidence about which are the more robust classification algorithms.

The rest of the paper is organized as follows. Section 2 introduces some of the related work performed in this field. Section 3 presents the methodology adopted in the study. The experiments and results obtained, along with their analysis are presented in Section 4 while conclusions and future work are provided in Section 5.

2. Related Work

Proposals for extensions of the CVE for automated security management of the software engineering process have been the focus of various work. Indeed, several studies use text mining techniques on CVEs for retrieving textual information from this repository, in order to perform a variety of tasks. Le et al. [5] presented an application of Natural Language Tool (NLT) to analyze the text-based vulnerability descriptions to retrieve vulnerability properties and evaluate their relationships. They suggested the use of text processing and natural language process to support automatic derivation of Vulnerable Property Relation Graph (VPRG) model extraction from text-based vulnerability descriptions. Simmons et al. [6] proposed an issue resolution system (IRS) to detect and extract information from external vulnerability repositories and internal log files to classify attack vector information from the national vulnerability database (NVD). The percentage of correctly classified CVEs is about 92%. Yamamoto et al. [7] attempted to use CVE documents to estimate the impact of vulnerability information. Their analysis was based on machine learning techniques, on the descriptions in the dictionary to extract the base metrics, and they found that there was a correlation between the estimation performance and temporal distance of the CVE documents. Khazaei et al. [8] introduced an objective method for CVSS score calculation. First, feature vectors were extracted using text mining tools and techniques from CVE descriptions, and then the SVM and Random-Forest algorithms, as well as fuzzy systems, were examined to predict the concerned CVSS score. In [9], the authors proposed a novel automated system, so-called ThreatZoom, to estimate the CWE classes corresponding to a CVE instance using both statistical and semantic features extracted from the description of a CVE. ThreatZoom takes the CVE's description as input and assigns a list of CWE classes along with the path connecting the roots of the tree to the lowest possible level class. Tested on the MITRE and NVD dataset, the tool achieved an accuracy of between 75-94%. Bozorgi et al. [10]

Table 1

number of samples in each class of the data set (dataset I)

Name of Vulnerability Class	No of descriptions	Percentage
Code Execution	32510	24.26
Denial Of Service	23569	17.59
Buffer Overflow	18501	13.80
Cross-Site Scripting	15046	11.23
Gain Information	10973	8.19
SQL Injection	7778	5.80
Authentication Bypass	6306	4.71
Memory Corruption	5318	3.97
Gain Privilege	5090	3.80
Directory Traversal	4060	3.03
Cross-Site Request Forgery	2471	1.84
File Inclusion	2232	1.67
Http Response,Splitting	165	0.12
Total number of descriptions	134019	

proposed the first work that focused on textual information hidden in vulnerabilities databases. Using tools from machine learning, they showed how to train classifiers that predict whether vulnerabilities are likely to be exploited, and if so, how soon. Differently from these previous works, we provide an investigation for evaluating misclassification produced by the different machine and deep learning algorithms based on CVE description in order to support security engineers and developers in making decisions.

3. The Methodology

The dataset used in the experiment is a collection of 134,019 vulnerability descriptions as shown in Table 1, extracted from the CVE Mitre database², through a web scraper developed by the authors. Since the samples gathered in the first dataset, that we will call DATASET-I from here on, showed redundancies, another dataset was created by removing the duplicates, DATASET-II in the remaining of the paper. Duplicates were identified when the Levenshtein distance between two descriptions was zero edit (100% similar). Since machine learning models can take in only numerical inputs, the descriptions had to be converted into feature vectors that can be used to train and test the models. For this purpose, the descriptions were tokenized first to sentences and then to words. The tokenized words are then stemmed using Porter stemmer [11], so as to obtain the root of every token. After stemming is done, vectors are created for each vulnerability description. For shallow learning models, a count vectorization tool is used to create feature vectors. Each vector represents the count of words in every vulnerability description. Since the count of words in a vulnerability description does not provide enough information, we employed Term Frequency-Inverse Document Frequency (TF-IDF) [12], for differentiating various vulnerability descriptions. For deep learning models, the feature vectors

²<https://cve.mitre.org/>

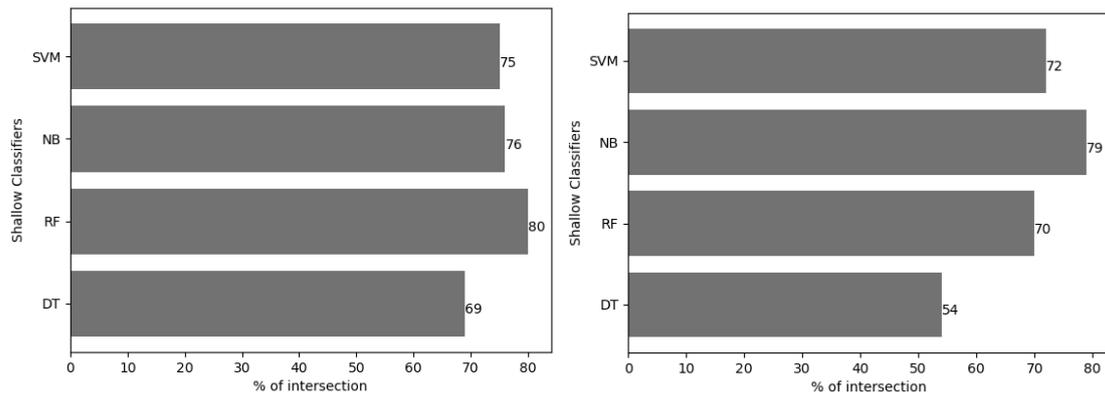
Table 2

vulnerability descriptions count in each class of the dataset with duplicates removed (dataset II)

Name of Vulnerability Class	No of descriptions	Percentage
Code Execution	31166	33.81
Denial Of Service	17063	18.51
Buffer Overflow	3999	4.34
Cross-Site Scripting	14430	15.66
Gain Information	10137	11
SQL Injection	1324	1.44
Authentication Bypass	4837	5.25
Memory Corruption	80	0.09
Gain Privilege	3421	3.71
Directory Traversal	3547	3.85
Cross-Site Request Forgery	1946	2.11
File Inclusion	81	0.09
Http Response Splitting	136	0.15
Total number of descriptions	92167	

are created by converting the text sequences to number sequences using pre-trained word embedding which is obtained with Google’s Word2Vec which is a neural network consisting of two layers of which the final layer is re-trained with the corpus. The window size taken is five, and the network is trained for twenty iterations. The vectors formed here have a dimension size of hundred. These vectors are used to obtain the contextual information of each word with respect to its sentence. Once pre-processing has been completed, the feature vectors are used to train the classifiers. After the model has been trained successfully, the model can then predict the class labels associated with each vulnerability description when new instances are given as input to the classification model. The performance of the classifiers is then evaluated using the performance metrics such as accuracy, precision, recall and F1-score. Shallow classifiers were realized with the following algorithms: Decision Tree, Random Forest, Support Vector Machine, Naïve Bayes, XGBoost. For Deep Learning based classifiers, the following systems were employed: Long Short Term, Memory (LSTM)M [13], Bidirectional-Long Short Term Memory (BiLSTM) [14], Convolutional Neural Network (CNN), attention network [15], and Graph Convolutional Neural Networks (GCN).

Both K -fold cross-validation and train-test split are performed on the dataset. In case of train-test split, first the dataset is loaded and split into training and testing sets where 80% is used for the training set, while the remaining 20% will constitute the test set. The train test split method may sometimes result in overfitting of the model. In order to avoid this, cross-validation (k -Folds where ($K=5$) is performed on the dataset. In 5-fold cross-validation, the entire dataset is divided into five subsets each of size $n/5$, where n is the size of the dataset. In each iteration, there will be four training sets and one testing set. The classification model performance is estimated as average accuracy computed after five iterations. Thirteen vulnerability classes were extracted by the dataset and are showed in Table 2.



(a) Shallow Classifiers with DATASET I

(b) Shallow Classifiers with DATASET II

Figure 1: The intersection of misclassified vulnerability descriptions between the best shallow classifier (stacking-XGBoost) and other shallow classifiers when trained and tested using: (a) DATASET I, (b) DATASET II

4. Misclassification Analysis

The shallow classifier with the highest F1-score was stacking-XGBoost in both the datasets. In DATASET I the number of samples misclassified is 3,301. Figure 1a shows the intersection of misclassified vulnerability descriptions between the best shallow classifier (stacking-XGBoost) and other shallow classifiers when trained and tested using DATASET I. It emerges that the highest intersection (80%), was obtained with random forest, i.e. among the 13,108 samples incorrectly predicted by random forest, 2637 samples were found to be commonly misclassified by both stacking-XGBoost and random forest. The lowest intersection (69%), was obtained with decision tree i.e. out of 8,918 samples were incorrectly classified by decision tree, 2,266 samples were classified wrongly by both shallow classifiers.

1) *Stacking Classifier and Random Forest:* The class to which the highest number of samples were wrongly predicted by stacking classifier is “Code Execution”. Out of 878 wrongly classified samples to the class “Code Execution” by stacking classifier, 363 samples were commonly misclassified by both the classifiers by 41%. Likewise, the next highest classes to which stacking classifier incorrectly classifies are, “DOS” (714 samples) and “Memory Corruption” (572 samples) respectively. It was found that 336 samples were commonly misclassified by both stacking and random forest towards DOS class which amounts to 47%, and 31% (182 samples) for the class “Memory Corruption”. The class to which the least number of samples were incorrectly predicted by stacking classifier is “HTTP Response Splitting” which amounts to 6 samples out of which 5 samples (83%) were commonly misclassified by both.

2) *Stacking classifier and decision tree:* Similarly, it is observed that out of 878 wrongly classified samples to the class “Code Execution” by stacking classifier, 565 (64%) samples were commonly misclassified by both the classifiers. Likewise, the next highest classes to which stacking classifier incorrectly classifies are, “DOS” (714 samples) and “Memory Corruption” (572 samples) respectively. It was found that 355 samples were commonly misclassified by both

stacking and decision tree towards “DOS” which amounts to 50%, and 9% (50 samples) for the class “Memory Corruption”. The class to which the least number of samples that were classified wrongly by stacking classifier is “HTTP Response Splitting” which amounts to 6 samples out of which both commonly misclassified one sample (17%).

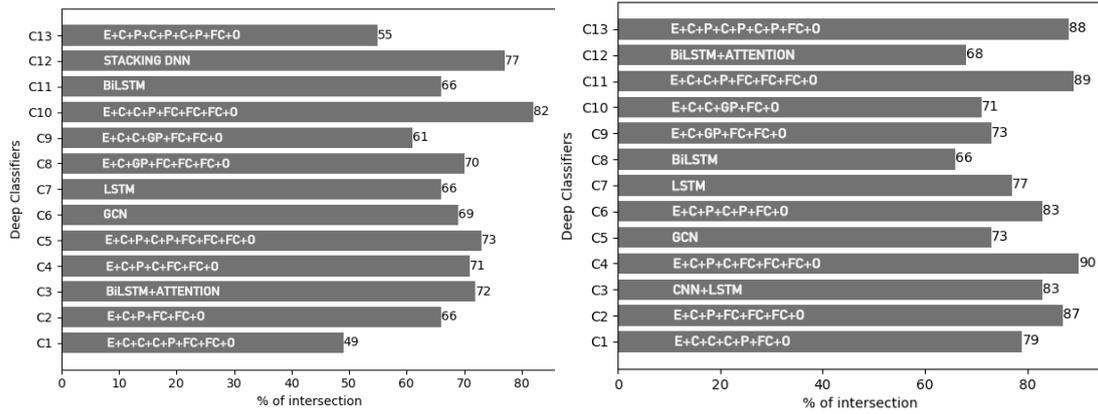
Similarly for DATASET II, the shallow classifier with F1-Score is stacking-XGBoost, which wrongly predicts 300 vulnerability descriptions. The intersection of misclassified vulnerability descriptions between the best shallow classifier (stacking-XGBoost) and other shallow classifiers when trained and tested using DATASET II, is depicted in Figure 1b. It is observed that the highest intersection (79%), was obtained with naïve bayes, i.e. out of the 1556 samples wrongly classified by naïve bayes, 237 samples were found to be misclassified by both stacking-XGBoost and naïve Bayes. The least intersection (54%) was obtained with decision tree.

1) *Stacking Classifier and Naïve Bayes*: The class to which the highest number of samples were wrongly classified by stacking classifier is “Gain Information”. Out of 78 wrongly classified samples to the class “Gain Information” by stacking classifier, 40 samples were commonly misclassified by both by 51%. Likewise, the next highest classes to which stacking classifier incorrectly classifies are, “Code Execution” (55 samples) and “Authentication Bypass” (39 samples) respectively. On further analysis, it was found that 31 samples were commonly misclassified by both stacking and naïve bayes towards “Code Execution” which amounts to 56% and amounts to 69% (27 samples) for “Authentication Bypass”. The class to which the least number of samples were predicted incorrectly by stacking classifier is “File Inclusion” which amounts to one sample. None of the samples was commonly misclassified by both.

2) *Stacking Classifier and Decision Tree*: Likewise, it is observed that out of 78 wrongly classified samples to the class “Gain Information” by stacking classifier, 46 samples were commonly misclassified by both the classifiers by 59%. The next highest classes to which stacking classifier incorrectly classifies are, “Code Execution” (55 samples) and “Authentication Bypass” (39 samples) respectively. It was found that 22 samples were commonly misclassified by both stacking and decision tree towards “Code Execution” class which amounts to 40%, and 54% (21 samples) for “Authentication Bypass” class. The class to which the least number of samples were misclassified by stacking classifier is “File Inclusion” which amounts to one sample, and none of the samples was commonly misclassified to this class.

The deep classifier with the highest F1-score is found to be CNN+LSTM in DATASET-I and stacking-DNN in DATASET-II. In DATASET I the number of samples wrongly classified by CNN+LSTM model is 8343. Figure 2a graphically represents the intersection of wrongly predicted vulnerability descriptions between CNN+LSTM and other deep classifiers when trained and tested using DATASET I. It is observed that the highest intersection (82%), was obtained with deep classifier (E+C+C+P+FC+FC+FC+O), i.e. among the 8771 wrongly predicted samples by the deep classifier (E+C+C+P+FC+FC+FC+O), 6864 samples were found to be predicted wrongly by both the deep classifiers. The lowest intersection (49%), was obtained with E+C+C+C+P+FC+FC+O, i.e. out of 8607 samples, 4079 were classified incorrectly.

1) *CNN+LSTM and E+C+C+P+FC+FC+FC+O*: It was found that the wrongly classified samples by both classifiers to the same class are as follows. The class to which the highest number of samples were predicted incorrectly by CNN+LSTM is “Memory Corruption”. Out of 2219, wrongly classified samples to the class “Memory Corruption” by CNN+LSTM, 1219 samples were commonly misclassified by both by 55%. Likewise, the next highest classes to which CNN+LSTM



(a) Deep Classifiers with DATASET I

(b) Deep Classifiers with DATASET II

Figure 2: The intersection of misclassified vulnerability descriptions between the best deep classifier and other deep classifiers when trained and tested using: (a) DATASET I, (b) DATASET II

incorrectly classifies are, “DOS” (1819 samples) and “Buffer Overflow” (1408 samples) respectively. On further analysis, it was found that 1091 samples were commonly misclassified by both CNN+LSTM and E+C+C+P+FC+FC+FC+O towards “DOS”, which amounts to 60% and amounts to 83% (1168 samples) for “Buffer Overflow”. The class to which the least number of samples were misclassified by CNN+LSTM is HTTP Response Splitting which amounts to 6 samples out of which 5 (84%) of them are incorrectly predicted by both deep classifiers.

2) *CNN+LSTM and E+C+C+C+P+FC+FC+O*: Likewise, it is observed that out of 2,219 wrongly classified samples to the class “Memory Corruption” by CNN+LSTM classifier, 526 samples were commonly misclassified by both the classifiers by 24%. The next highest classes to which LSTM+CNN incorrectly classifies are, DOS (1819 samples) and “Buffer Overflow” (1408 samples) respectively. On further analysis, it was found that 604 samples were commonly misclassified by both CNN+LSTM and E+C+C+C+P+FC+FC+O towards “DOS” class which amounts to 33%, and 9% (130 samples) for “Buffer Overflow” class. The class to which CNN+LSTM misclassified the least number of samples is “HTTP Response Splitting” which amounts to 6 samples out of which both deep classifiers misclassified 4 samples (67%).

For DATASET II, the deep classifier with the highest F1-Score is stacking-DNN which wrongly predicts 225 vulnerability descriptions. Figure 2b graphically represents the intersection of misclassified vulnerability descriptions between the best deep classifier (stacking-DNN) and other deep classifiers when trained and tested using DATASET II. It is observed that the highest intersection (90%), was obtained with deep classifier (E+C+P+C+FC+FC+FC+O), i.e. among the 349 samples incorrectly predicted by (E+C+P+C+FC+FC+FC+O), 203 of them were found to be misclassified by both the deep classifiers. The lowest intersection (66%) was observed with BILSTM, i.e. out of the 277 samples wrongly classified by BILSTM, both commonly misclassified 148 samples.

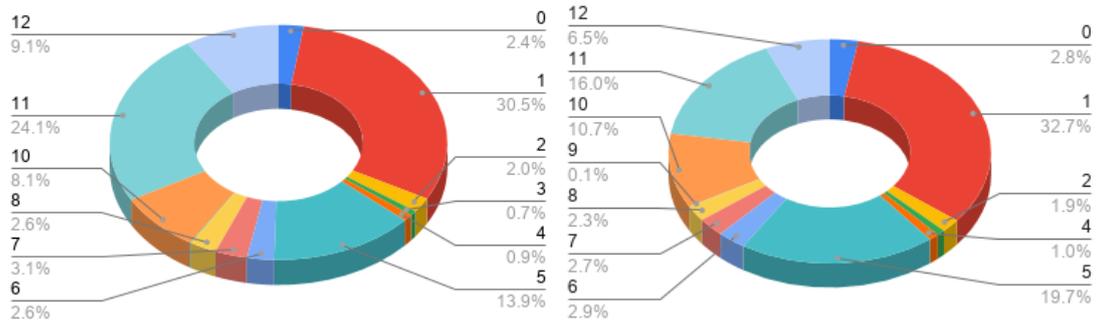
1) *Stacking-DNN and E+C+P+C+FC+FC+FC+O*: It was found that the wrongly classified samples by both classifiers to the same class are as follows. The class to which stacking-

DNN misclassified the highest number of samples is “Gain Information”. Out of 69 wrongly classified samples to the class “Gain Information” by stacking-DNN, 51 samples were commonly misclassified by both by 74%. Likewise, the next highest classes to which stacking-DNN incorrectly classifies are, “Buffer overflow” (39 samples) and “Authentication bypass” (32 samples) respectively. On further analysis, it was found that 135 samples were commonly misclassified by both stacking-DNN and E+C+P+C+FC+FC+FC+O towards “Buffer overflow”, which amounts to 90%, and 97% (31 samples) for “Authentication Bypass”. The class to which stacking-DNN incorrectly classified the least number of samples is “Memory Corruption” which amounts to 2 samples out of which both deep classifiers incorrectly predict 1 (50%) of them.

2) *Stacking-DNN and BILSTM*: Likewise, it is observed that out of 69 wrongly classified samples to the class “Gain Information” by stacking-DNN classifier, 48 samples were commonly misclassified by both the classifiers by 70%. The next highest classes to which stacking-DNN incorrectly classifies are, “Buffer Overflow” (39 samples) and “Authentication Bypass” (32 samples) respectively. It was found that 12 samples were commonly misclassified by both stacking-DNN and BILSTM towards “Buffer Overflow” class which amounts to 31%, and 84% (27 samples) for “Authentication Bypass” class. The class to which stacking-DNN misclassified the least number of samples is “Memory Corruption” which amounts to 2 samples out which 2 (100%) were misclassified by both deep classifiers. The intersection of misclassified vulnerability descriptions between the best shallow classifier (stacking-XGBoost) and other shallow classifiers when trained and tested using DATASET II, is depicted in Figure 1b. It is observed that the highest intersection (79%), was obtained with naïve bayes, i.e. among the misclassified samples, 79% of them were found to be misclassified by both stacking-XGBoost and naïve bayes. Figure 3a indicates the classes to which various proportions of misclassified samples in DATASET I actually belong to. It is found that 31% of the vulnerability descriptions that are misclassified actually belong to class 1 (Code Execution) (Refer to Table X). It is also noticed that none of the vulnerability descriptions belonging to class 9 (HTTP Response Splitting) is misclassified. Figure 3b shows the classes to which various proportions of misclassified vulnerability descriptions in DATASET I, were classified to. It is observed that 33% of the misclassified samples were classified to class 1 incorrectly. It is also observed that none of the samples was misclassified to class 9. Figure 4a depicts the classes to which various proportions of incorrectly classified samples in DATASET II actually belong to. It is found that 22% of the vulnerability descriptions that are misclassified actually belong to class 7. It is also noticed that none of the vulnerability descriptions belonging to class 6 (HTTP Response Splitting) is predicted incorrectly. The classes to which various proportions of misclassified vulnerability descriptions were classified to in DATASET II, is represented in figure 4b. It is observed that 23% of the misclassified samples were classified to class 1 incorrectly. It is also observed that none of the vulnerability descriptions was wrongly classified to class 9.

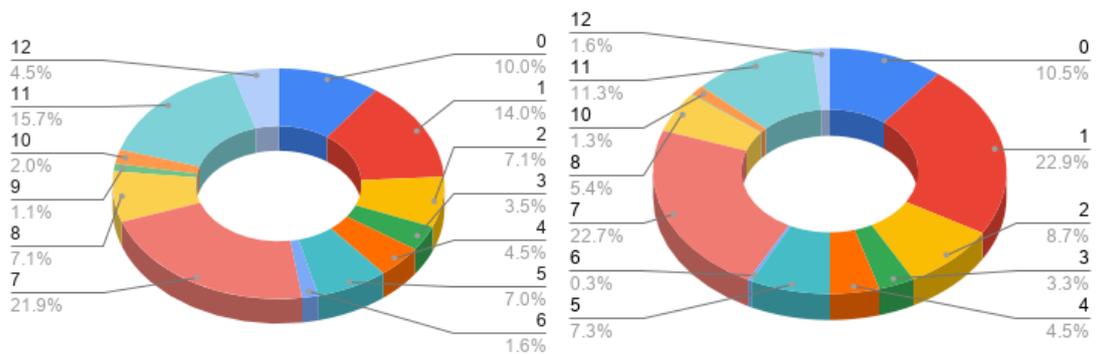
Figure 6a represents the confusion matrix of the predictions related to DATASET I. Cell $C_{i,j}$ represents the number of observations known to be in group i and misclassified to group j . For example, for the cell (12,1), it is seen that 95% of samples taken from class 12 for testing is wrongly classified to class 1. Additionally, it is also noticed that most number of vulnerability descriptions are being predicted wrongly to class 1, while the least number of vulnerability descriptions were incorrectly classified to classes 4,8,9,12.

Figure 5a represents the similarity graph for DATASET I derived from the confusion heat map



(a) Actual proportions of misclassified samples (b) Predicted proportions of misclassified samples

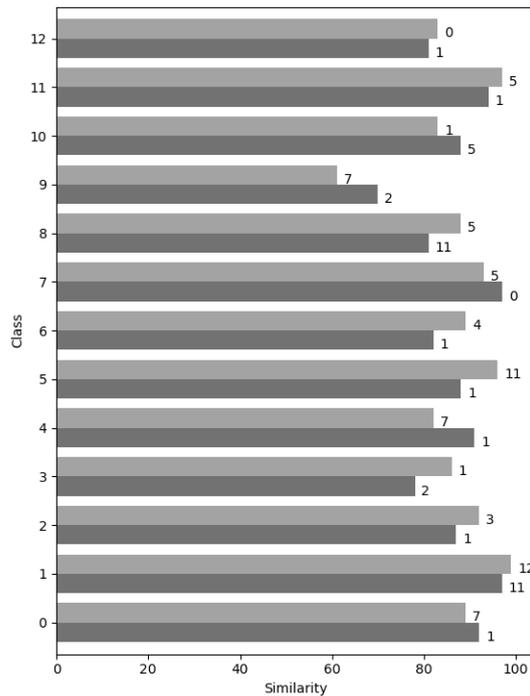
Figure 3: Proportions of misclassified samples of DATASET I: (a) Actual, (b) Predicted



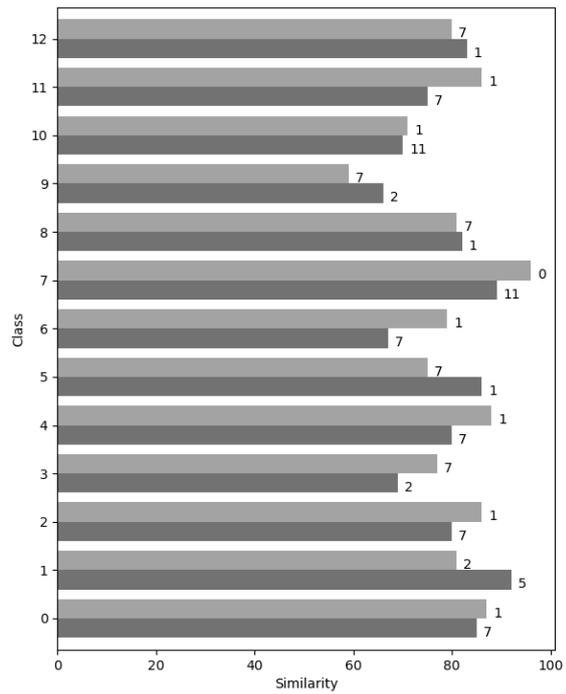
(a) Actual proportions of misclassified samples (b) Predicted proportions of misclassified samples

Figure 4: Proportions of misclassified samples of DATASET II: (a) Actual, (b) Predicted

of DATASET I (6a). Using cosine similarity, the similarity between the misclassified samples of the testing data and the data used for training the misclassified class is approximated. For this purpose, first, the required data from the respective training and testing classes are obtained and summarized. Extractive summarization using Natural Language Processing has been employed. This process consists of tokenizing the text first into sentences then to words. After this, the sentences are scored based on the scored assigned to the words in that particular sentence. Words are scored using the normalized frequency of each word. Finally, the summary consists of top n sentences. Approximately the size of a summary is half the size of the original text. The summarized text is then represented in a vector form which is used for finding the similarity measure. It can be observed that among the wrongly classified samples which actually belong to class 0, 29% were being mapped to class 1 and 24% were being mapped to class 7 (top 2 values from each row of the confusion heat map is taken into consideration). Thus it can be inferred that the 29% of samples from class 0 which is incorrectly classified to class 1 has high similarity to the samples used for training class 1, and 24% of the samples belonging to class 0 which



(a) The similarity graph for DATASET I



(b) The similarity graph for DATASET II

Figure 5: Intersection of misclassified descriptions obtained through various classifiers compared with misclassified instances of best classifiers similarity in percentage

is incorrectly classified to class 7 has high similarity with the vulnerability descriptions used for training class 7. Similarly, the similarity between the vulnerability descriptions used for testing and training for various classes has been depicted. Figure 6b represents the confusion matrix of the predictions related to DATASET II. Here it is observed that most of the samples were wrongly classified to Classes 1 and 7, almost equally. On the other hand, the least number of samples were predicted incorrectly to classes 9 and 10, respectively. Figure 5b depicts the similarity graph for DATASET II obtained from the confusion heat map of DATASET II (6b). It can be observed that among the wrongly classified samples which actually belong to class 0, 26% were being mapped to class 1 and 45% were being mapped to class 7 (top 2 values from each row of the confusion heat map is taken into consideration). Thus it can be concluded that the 26% of samples from class 0 has high similarity to the samples used for training class 1, and 45% of the samples belonging to class 0 has high similarity with the vulnerability descriptions used for training class 7. The similarity between the samples used for testing and training for different classes has been depicted.

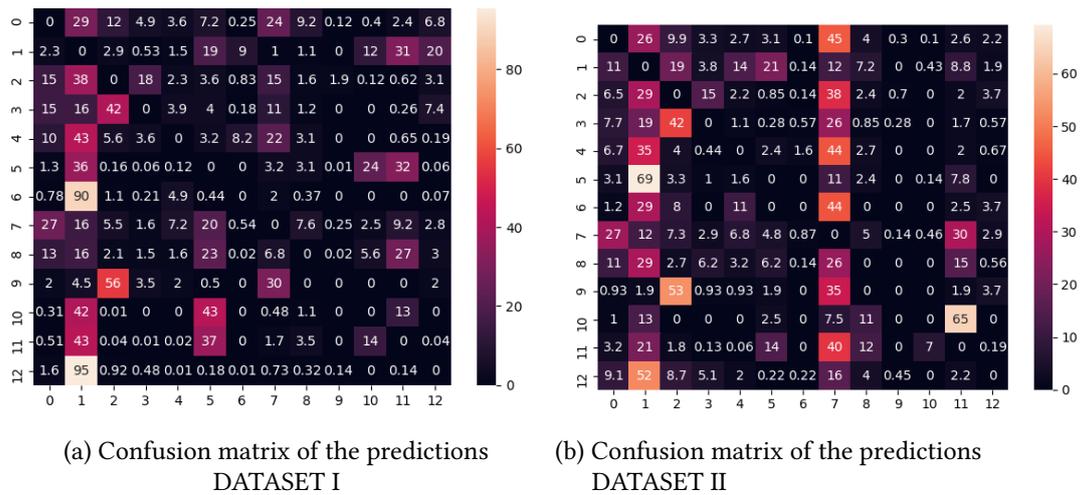


Figure 6: Information misclassification rate of vulnerability descriptions in 13 different classes: (a) DATASET I, (b) DATASET II

5. Conclusion

The growing number of vulnerabilities makes it urgent to have mechanisms for automatic classification of their descriptions. Since vulnerability reports are written in natural language, CVEs are usually classified manually which is a very time-consuming process. Machine and deep learning methods are increasingly adopted for automatically classifying vulnerabilities. Unfortunately, these approaches are prone to misclassification. In this paper, we provide an investigation over a dataset of 134,019 vulnerability descriptions extracted from the CVE repository for evaluating misclassification produced by different machine and deep learning algorithms. Our failure analysis showed that among the incorrectly classified samples most of the descriptions were being wrongly predicted to the classes “Code Execution”, “Denial of Service” and “Gain Information”. In the future, this work will be extended by evaluating the classification of newly discovered software vulnerabilities fetched from different websites where they are publicly announced. We will leverage our machine learning models for discovering new classes which can be included as the sub-class of the existing ones. This information would then be used to update our database which is then made publicly available. Since descriptions are too general, the code snippets attached with the vulnerability report can also be used to train our models so as to improve threat intelligence.

References

- [1] Skybox security, https://lp.skyboxsecurity.com/WICD-2020-02-Vulnerability\and-Threat-Trends-Report_01Reg.html, 2020. Accessed: 2021-03-10.
- [2] Common vulnerability scoring system (SIG), <https://www.first.org/cvss/>, 2020. Accessed: 2021-03-10.

- [3] Common vulnerabilities and exposures (CVE), <https://cve.mitre.org/>, 2020. Accessed: 2021-03-10.
- [4] S. Rehman, K. Mustafa, Software design level vulnerability classification model, *International Journal of Computer Science and Security (IJCSS)* 6 (2012) 238.
- [5] H. T. Le, P. K. K. Loh, Using natural language tool to assist vprg automated extraction from textual vulnerability description, in: *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, IEEE, 2011, pp. 586–592.
- [6] C. B. Simmons, S. Shiva, V. Phan, V. Shandilya, L. Simmons, Irs: An issue resolution system for cyber attack classification and management, SAM, Los Vegas (2012).
- [7] Y. Yamamoto, D. Miyamoto, M. Nakayama, Text-mining approach for estimating vulnerability score, in: *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, IEEE, 2015, pp. 67–73.
- [8] A. Khazaei, M. Ghasemzadeh, V. Derhami, An automatic method for cvss score prediction using vulnerabilities description, *Journal of Intelligent & Fuzzy Systems* 30 (2016) 89–96.
- [9] E. Aghaei, W. Shadid, E. Al-Shaer, Threatzoom: Hierarchical neural network for cves to cwes classification, in: *International Conference on Security and Privacy in Communication Systems*, Springer, 2020, pp. 23–41.
- [10] M. Bozorgi, L. K. Saul, S. Savage, G. M. Voelker, Beyond heuristics: learning to classify vulnerabilities and predict exploits, in: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 105–114.
- [11] M. F. Porter, et al., An algorithm for suffix stripping., *Program* 14 (1980) 130–137.
- [12] J. Han, J. Pei, M. Kamber, *Data mining: concepts and techniques*, Elsevier, 2011.
- [13] J. Cheng, L. Dong, M. Lapata, Long short-term memory-networks for machine reading, *arXiv preprint arXiv:1601.06733* (2016).
- [14] W. Wang, S. Hosseini, A. H. Awadallah, P. N. Bennett, C. Quirk, Context-aware intent identification in email conversations, in: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 585–594.
- [15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: *Advances in neural information processing systems*, 2017, pp. 5998–6008.