Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction

Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo

Abstract—In the research of mining software repositories, we need to label a large amount of data to construct a predictive model. The correctness of the labels will affect the performance of a model substantially. However, limited studies have been performed to investigate the impact of mislabeled instances on a predictive model. To bridge the gap, in this work, we perform a case study on the security bug report (SBR) prediction. We found five publicly available datasets for SBR prediction contains many mislabeled instances, which lead to the poor performance of SBR prediction models of recent studies (e.g., the work of Peters et al. and Shu et al.). Furthermore, it might mislead the research direction of SBR prediction. In this paper, we first improve the label correctness of these five datasets by manually analyzing each bug report, and we find 749 SBRs, which are originally mislabeled as Non-SBRs (NSBRs). We then evaluate the impacts of datasets label correctness by comparing the performance of the classification models on both the noisy (i.e., before our correction) and the clean (i.e., after our correction) datasets. The results show that the cleaned datasets result in improvement in the performance of classification models. The performance of the approaches proposed by Peters et al. and Shu et al. on the clean datasets is much better than on the noisy datasets. Furthermore, with the clean datasets, the simple text classification models could significantly outperform the security keywords-matrix-based approaches applied by Peters et al. and Shu et al.

Index Terms—Security bug report prediction, data quality, label correctness

1 Introduction

Mining software repository (MSR) has become an attractive research area to uncover interesting and actionable information about software systems and projects [1]. The basis of MSR is big data of software engineering, and a large amount of labeled data is required for constructing a prediction model [2]. The label correctness of data is critical for correctly assessing the effectiveness of a prediction model [3], [4], [8]. Furthermore, the incorrectly labeled data may translate to inaccurate results that may in turn mislead the research direction of the target problem [5], [6], [7], [8], [9], [10].

Currently, the studies that focus on the impact of inaccurate labels are limited, and the attention paid to label correctness is required when constructing a predictive model. This paper investigates the impacts of mislabeled instances for security bug report (SBR) prediction since identifying SBRs from a large-scale bug repository is critical for reducing the security risks of a software product. Many machine learning-based SBR prediction approaches have been proposed in recent years [11], [12], [13], [14], [15], [16], [17]. However, the performance (e.g., F1-score) of the two recent studies proposed by Peters et al. [11] and Shu et al. [12] is not ideal. For example, when considering F1-score, the work of Peters et al. only achieved 0.37 in their best cases. Shu et al. directly used the datasets processed by Peters et al. and improved the performance of prediction models by using a hyperparameter optimization approach. Their experiment results showed in the best case their approach could reach 0.86 and 0.25 in terms of pd (Recall) and pf (false positive rate), which means the number of the false positive item reaches 5,388 in Chromium as the NSBRs in the testing set of Chromium is 20,855 in their study. Such a high rate of false alarms is still unacceptable to deploy the tool into practice.

In this paper, we explore the reasons that lead to the poor performance of their studies and find one reason that can not be ignored is the quality of labels assigned to the datasets. The five publicly available datasets used by Peters et al. and Shu et al. are Chromium, Ambari, Camel, Derby, and Wicket. The first dataset is publicly shared during MSR 2011 [18] and contains 40,940 bug reports, while the remaining four datasets are manually labeled by Ohira et al. [19]. As mentioned by Peters et al., there are SBRs mislabeled as NSBRs in these five datasets. For example, in Chromium, bug reports related to *memory leakage* and *null pointer* problems are marked as NSBRs. However, they belong to classical vulnerability types since such issues are frequently exploited by hackers [20], [21], and are listed in the top 25 most dangerous CWE (Common Weakness Enumeration) types [22]. Figure 1 shows three examples of the mislabeled instances in Chromium. We mark the text that shows that the corresponding bug report is an SBR in red.

Additionally, the approaches applied by Peters et al. and

- *Xiaoxue Wu is with the School of Cyberspace Security, Northwestern Polytechnical University, Xi'an, China. E-mail: wuxiaoxue00@gmail.com*
- *Wei Zheng is with the School of Software, Northwestern Polytechnical University, Xi'an, China. E-mail: wzheng@nwpu.edu.cn*
- *Xin Xia is with Faculty of Information Technology, Monash University, Melbourne, Australia. E-mail: xin.xia@monash.edu*
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore. E-mail: davidlo@smu.edu.sg*

Manuscript received April 19, 2005; revised August 26, 2015.

ID &Title	Issue-1643: The memory leak in the root process	Issue 16036 : Alsa Audio Output leaks on shutdown	Issue 3795: Check: CrashForBitmapAllocationfailure()
Description	After several days of usage the root process allocated 1.6GiB of memory. And this process could not be restarted or selectively killed. **The root process must not leak the memory** or it should be easily restartable picking existing page processes after restart. Fro example page navigation could be done by the separate child process. That respawns if it dies. 1.6GiB is allocated.	Audio Output Stream's shutdown code does not correctly delete the output stream object after the thread is shutdown. **This causes a resource leak**. Because the data source has been modified to be non-blocking we should now remove the per-stream output thread and use a single message loop over all output streams for writing audio data.	I don't know what we can do other than look for **GDI leaks**. This is a common crash in 154.6 albeit with very little data so far. ... If we run out of GDI objects or some other error occurs we won't get a // bitmap here. **This will cause us to crash later because the data pointer is NULL.** To make sure that we can assign blame for those crashes ...

Fig. 1. Examples of mislabeled bug reports in Chromium. The red-colored texts are contents that help us to decide that a bug report is security-related (i.e., SBR).

Shu et al. are complex and time-consuming. For example, the hyperparameter tuning for the learner applied by Shu et al. costs around 5 hours to optimize the parameter of learner RF on dataset Chromium. According to the suggestions of Fu et al. [23] and Liu et al. [24], it is a good practice to explore simple but effective approach. Therefore, we investigate the effectiveness of simple text classification for SBR prediction.

Our study aims to answer the following two research questions:

RQ1: To what extent can dataset label correctness impact the performance of classification models?

We conduct an experimental evaluation of classification models' performance on both the clean datasets and the noisy datasets. We first correct the mislabeled records of the five datasets with manual annotation and obtain five clean datasets. As a result, we identified 749 SBRs that are mislabeled.

Following Peters et al. 's evaluation [11], we divided each of the five clean datasets into two equal parts, which are respectively training set and testing set. Then, we execute the approaches proposed by Peters et al. and Shu et al. on the clean datasets and the noisy datasets to obtain their performance values. Finally, we evaluate the impact of label correctness by comparing baseline approaches on the clean datasets with that of the noisy datasets. We find the performance of three baseline approaches on the clean datasets consistently outperforms that on the noisy datasets. On average, Recall, Precision, F1-score, and G-measure are increased by 191%, 46%, 147%, and 136%, respectively.

RQ2: How does simple text classification perform on the clean and the noisy datasets for SBR prediction?

We raise RQ2 because the main content of bug reports for SBR prediction is the *Description* described with text nature language [25], [26]. Moreover, much previous work focused on bug report analysis uses text classification combined with machine learning as text classification is direct and straightforward [13], [14], [15], [16], [17], [27].

We run all the five classification algorithms (i.e., Random Forest, Naive Bayes, K-Nearest Neighbour, Multilayer Perceptron, and Logistical Regression) applied by Peters et al. and Shu et al. with simple text classification on both the clean and noisy datasets. We find that the overall performance values (i.e., Recall, Precision, F1-score, and G-measure) on the clean datasets are much higher than those on the noisy datasets. Furthermore, on the clean datasets, the performance values (i.e., Recall, Precision, F1-score, and G-measure) of Random Forest using the text classification are much better than the performance of the security keywords-matrix-based baseline approaches.

Finally, we conduct further analysis of the reason behind Farsec's poor performance, the time cost of hyperparameter tuning approaches, and the implications of our study. We point out that data label correctness matters much for the performance of classification models, and the simple text classification is more effective than the carefully-designed matrix-based baseline approaches for SBR prediction while the data label correctness is improved.

This paper makes the following contributions:

- We manually annotate the label correctness of five publicly available datasets (i.e., Chromium, Ambari, Camel, Derby, and Wicket). We find there are no NSBRs mislabeled as SBRs; however, we identify a total of 749 SBRs that are mislabeled as NSBRs.
- To the best of our knowledge, we are the first to experimentally evaluate the impact of data label correctness for SBR prediction.
  We find that (1) for the same classifier, the performance on the clean datasets is much better than that of the noisy datasets; (2) simple text classification performs better when training is performed on the clean datasets than the noisy datasets; (3) with the clean datasets, simple text classification outperforms the three baseline approaches. These findings provide research clues and guidance for researchers and practitioners of SBR prediction.

The remainder of this paper is organized as follows. Section 2 introduces the related work of this study. Section 3 gives the background. Section 4 describes our data annotation method and results. Section 5 describes our experimental settings. Section 6 details our experimental results of each research question respectively. Section 7 discusses if the hyperparameter tuning method proposed by Shu et al. works for simple text classification, the patterns of mislabeled SBRs in noisy datasets, the performance levels of annotators, the implications of our study, and threats to the validity of our study. Section 8 summarizes the paper.

2 Related Work

2.1 SBR Prediction

The two recent works focused on SBR prediction are proposed by Peters et al. [11] and Shu et al. [12]. These two

work uses the same datasets and classification algorithms in their studies. Shu et al. improve the work of Peters et al. by applying the hyperparameter tuning to the learner (classifier) and oversampling approach SMOTE.

Before this, Wijayasekara et al. [28] analyzed bug reports of two well known open-source projects - Linux kernel and MySQL. They showed that 32% and 62% vulnerabilities of Linux kernel and MySQL in the time period of 2006 to 2011 were hidden vulnerabilities, which were reported in the bug tracking system without being identified as SBRs until disclosed to the public [29]. Their study also showed an increasing trend in the distribution of the hidden vulnerabilities in the following two years. Besides, the authors presented a method of identifying hidden vulnerabilities by using the long description (i.e., the field *Description*) and the short description (i.e., the field *Title*) of bug reports to generate features before applying a classification approach.

An earlier work focused on SBR prediction was proposed by Michael et al. [13] in 2010. The authors developed an automatic approach that uses text mining on bug report descriptions to train a statistical model with manually-labeled bug reports. This model is then used to identify SBRs that are manually-mislabeled as NSBRs in the bug tracking system. Their approach is evaluated on a Cisco software system.

2.2 The impact of data label correctness

In recent years, researchers have raised concerns about the quality of the dataset employed by machine learning-based classification models in the area of software engineering [4], [8], [30], [31], [32], [33], [34].

Tantithamthavorn et al. [4] investigates whether mislabeling is random in production. The authors conducted an analysis of 3,931 issue reports from Apache Jackrabbit and Lucene systems and summarized several findings. Such as the issue report mislabeling is not random; mislabeled issue reports rarely impact precision. Especially, their experiments showed the models trained on noisy data achieve 56%-68% of the Recall of models trained on clean data.

Kim et al. [8] experimentally evaluate the impact of noise for source code defect prediction. They conduct their study on both file-level and change-level by intentionally and randomly injecting false positives and false negatives into their datasets. Their results show that the performance of the defect prediction models are decreased significantly when 25% to 35% changes in the datasets are mislabeled.

Fan et al. [31] investigate the impact of the mislabeled changes on the performance and interpretation of just-in-time defect prediction models. The authors analyzed four SZZ [35] variants and then built prediction models with the labeled data by these four variants. Their experimental study on 126,526 changes from ten Apache projects showed the mislabeled changes by AG-SZZ cause a significant performance reduction. Considering the developers' inspection effort, the mislabeled changes of B-SZZ and AG-SZZ lead to 9% to 10% and 1% to 5% more wasted inspection effort.

Herzig et al. [33] observed that tangled changes impact the number of associated bugs for 16.6% of all source files. Since such a considerable number of files are affected with respect to the number of associated bugs, the noise induced by tangled changes significantly impacts the models which predict the number of bugs in source files.

Kochhar et al. [32] noted issue report mislabeling; namely, the reports labeled as bugs but actually referred to non-bug issues. The authors extract different feature values from bug reports and predict whether a bug report needs to be reclassified. They evaluated their approach with bug reports of projects HTTPClient, Jackrabbit, Lucene-Java, Rhino, and Tomcat5. The results show the performance of their approach ranges from 0.58 to 0.71, 0.61 to 0.72, and 0.57 to 0.71 for Precision, Recall, and F1-score, respectively.

Our study is orthogonal to the above studies. First, we improve the label correctness of five publicly-available datasets applied by two recent SBR prediction work proposed by Peters et al. and Shu et al. Second, we use the approaches proposed by Peters et al. and Shu et al. as baselines of this study. Third, we evaluate the impact of data label correctness by comparing baselines' performance on noisy datasets with that of clean datasets. We conduct an experimental evaluation of the impact of data label correctness by using the three baseline approaches. Finally, we apply the commonly used text classification approach to both the clean datasets and noisy datasets, and show the text classification is a simple but effective approach while the data labels are reliable.

3 Background

In this section, we recall the two recent SBR prediction work that inspired our study.

Farsec framework. Peters et al. [11] point out the mislabel problem of the five publicly available SBR prediction datasets: Chromium, Ambari, Camel, Derby, and Wicket. They design a framework named Farsec to improve SBR prediction by filtering out noisy data from NSBRs and extending the text mining approach with the security keywords matrix. The process of Farsec contains three major steps: *(1) Identifying security keywords and making data matrices.* They first tokenize SBRs of a dataset to terms. Then, each term's tf-idf value is calculated, and the top 100 terms with the highest tf-idf values are kept as security keywords. After that, they calculate the frequency of each security keyword in the *Description* of each bug report for both training set and testing set, and security-keywords matrices of the training set and testing set are generated respectively. *(2) Filtering NSBRs with security-related keywords.* The purpose of filtering in Farsec is to remove NSBRs with security related keywords. To achieve this, they designed seven different filters: farsec, farsecsq, farsectwo, clni, clnifarsec, clnifarsecssq, and clnifarsectwo. Table 1 provides a brief description of these filters. They apply each of these filters to the training set of each dataset. Therefore, seven new training sets are generated, and each of them is applied to fit the model independently. *(3) Ranking bug reports.* After predicting the SBRs, a list of ranked bug reports was generated based on ensemble learning. The actual SBRs in the prediction results appear closer to the top of the list.

Hyperparameter tuning. In machine learning, model parameters indicate the properties of training data. Hyperparameter optimization is the process of searching the most optimal values of the hyperparameters in the model [38].

TABLE 1
The seven filters applied by Peters et al. [11]

Filter	Description
farsec	Apply no support function.
farsecsq	Apply the support function proposed by Jalali et al. [36] to the frequency of words found in SBRs.
farsectwo	Apply the Graham version [37] of multiplying the frequency by two.
clni	Apply noise filter CLNI (Closet List Noise Identification).
clnifarsec	Apply CLNI to the farsec filtered data.
clnifarsecsq	Apply CLNI filter to the farsecsq filtered data.
clnifarsectwo	Apply CLNI filter to farsectwo filtered data.

Shu et al. [12] improve the performance of SBR prediction by applying a hyperparameter optimization approach. They conduct their study on the basis of Peters et al.'s work. A differential evolution algorithm is used to optimize the control parameters of a learner (i.e., classification algorithm) and the data pre-processing approach SMOTE [39], respectively. For example, they improve Random Forest by tuning the number of trees in the forest (i.e., the parameter $n\_estimators$); the default value of it is ten, and the tuning range is set as 10 to 150. They tune the number of neighbors (i.e., the parameter k) in SMOTE, the default value is 5, and the tuning range is set as 1 to 20. Table 2 gives hyperparameters and the setting of tuning classifier *Random Forest* and oversampling approach *SMOTE* in their study.

4 Methodology of SBR Data Annotation

Data annotation is an extremely time-consuming work [24], [40], [41]. It is never trivial to correctly label SBRs due to the demand of qualified expertise [42], [43], [44].

The five datasets applied by Peters et al. and Shu et al. are Chromium, Ambari, Camel, Derby, and Wicket. In this section, we introduce the annotation process of noisy datasets as well as our clean datasets.

4.1 Annotation on the noisy datasets

Here, we briefly introduce how the SBRs are identified in the five datasets we used in this paper. Moreover, we also analyze the reason why the original datasets contain mislabeled instances.

Chromium: Chromium[1] is an open-source browser project. The dataset of Chromium is shared by the 2011 mining challenge of the MSR conference [18], which contains 40,940 bug reports. The SBR labels are initially marked by the reporters, and many of them are correlated with CVE (Common Vulnerabilities and Exposures) entries [45]. For example, the records with identities Issue 34495 and Issue 34498 are correlated with CVE-2010-0048 and CVE-2010-0052, respectively. Therefore, these labels of SBRs in the dataset are reliable.

However, as Peters et al. [11] noted, there are still many hidden SBRs that are mislabeled as NSBRs in this dataset. This is common for open-source projects as many bug reports of such projects are submitted by end-users, who are rarely professional software security personnel.

1. https://www.chromium.org/Home, Jun. 2020.

Four Apache Datasets: The datasets of the four Apache projects Ambari[2], Camel [3], Derby [4], and Wicket [5] were initially collected by Ohira et al. [19] from JIRA, which is a widely applied issue tracking system [46]. There are a variety of types for an issue in JIRA, such as Bug, Improvement, Documentation, and Task. Ohira et al. randomly selected one thousand issues with types Bug or Improvement for each project. They manually reviewed and labeled these 4,000 issues by graduated students and faculty. They marked six high impact types for these issues (i.e., Blocking, Security, Performance, and Breakage bugs) with the following steps:

1) Given a project, one student and one faculty member manually label the bug reports independently.

2) And then they discuss their disagreements to reach a common decision.

As mentioned by Ohira et al. [19], some issues were judged differently for security, performance, and breakage categories because there were no established definitions of these categories. This should be a reason why there are SBRs mislabeled as NSBRs in these datasets. Another possible reason that leads to the mislabels should be the lack of software security knowledge of annotators. For example, if a software product throws an exception due to *nullpointer* reference, a bug reporter without sufficient security knowledge may report this bug as an NSBR. However, the *nullpointer* could be exploited by hackers, and the *nullpointer dereference* is one of the CWE top 25, which is a demonstrative list of the most widespread and critical weaknesses that can lead to serious vulnerabilities in software products [22].

4.2 Annotation on the clean datasets

Our data review aims to identify SBRs which are mislabeled as NSBRs from the five datasets (i.e., Chromium, Ambari, Camel, Derby, and Wicket). An SBR is a bug report describing one or more vulnerabilities of a software system [13]. A security vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network [47]. The occurrence of vulnerability may cause some unsafe consequences, such as information leakage and illegal privilege escalation.

To guarantee the quality of our manual annotation results, we not only arrange experienced software security experts but also develop a specific manual review process. We use software vulnerability types defined by CWE [48] as a basis and generate a codebook as guidelines of judging whether a bug report is an SBR.

4.2.1 Annotators

We have six annotators. Two of them are Ph.D. students, and the other four are employees of Huawei corporation. They have at least four years of software security-related work experience and are familiar with the projects from which the bug reports originated.

Table 3 provides the background of the six annotators from four dimensions: (1) their roles in the organization (i.e.,

2. http://ambari.apache.org/, Jun. 2020.
3. http://camel.apache.org/, Jun. 2020.
4. http://db.apache.org/derby/, Jun. 2020.
5. https://wicket.apache.org/, Jun. 2020.

TABLE 2
The setting of hyperparameter tuning for Random Forest and SMOTE. (Note: DE is short for differential evolution.)

Target	Parameters of Target			Parameters of DE			
	Parameters	Default	Tuning Range	NP	F	CR	ITER
Random Forest	$n\_estimators$	10	[10,150]	60	0.8	0.9	3,10
	$min\_samples\_leaf$	1	[1,20]				
	$min\_samples\_split$	2	[2,20]				
	max_leaf_nodes	None	[2,50]				
	$max\_features$	auto	[0.01,1]				
	max_depth	None	[1,10]				
SMOTE	k	5	[1,20]	30	0.8	0.9	10
	m	50%	[50,400]				
	r	2	[1,6]				

column Role); (2) their experience with practical software development and testing (i.e., column Software develop); (3) experience with the projects or similar projects (i.e., column Projects); (4) and experience with security bug analysis, such as source code vulnerability analysis, security bug reports analysis (i.e., column Security bug). To be objective, we use *year* (number of years of experience) to quantitatively measure the annotators' experience in each dimension. In addition, the last column (i.e., Note) of the table provides key supplementary information of annotators.

4.2.2 Software vulnerability categories

Software security researchers have created taxonomies of vulnerability types. According to Landwehr et al. [49], security vulnerabilities in software systems range from local implementation errors (e.g., use of the gets() function call in C/C++), through interprocedural interface errors (such as a race condition between an access-control check and a file operation), to much higher design-level mistakes (e.g., error handling and recovery systems that fail in insecure fashions or object-sharing systems that mistakenly include transitive trust issues).

CWE is a community-developed list of common software weaknesses that might result in systems being vulnerable to attack if left unaddressed [48]. It lists representative vulnerabilities from major operating systems vendors, commercial information security tool vendors, academia, government agencies, and research institutions. According to CWE definition, vulnerabilities that share a common characteristic are grouped as one category, and there are 40 top-level CWE categories and 417 subcategories for software product up to now [50]. For example, one of the top-level categories is CWE-1228, described as *API/function errors*. It is related to the use of built-in functions or external APIs. It includes seven subcategories like CWE-242 (Use of inherently dangerous function), CWE-477 (Use of obsolete function), and CWE-479 (Exposed dangerous method or function) [51].

To clarify the attributes of SBRs, annotators were asked to map each SBR to specific CWE categories (or subcategory) during the annotation process. CWE involves a broad and intensive knowledge base. However, for a given software system, its domain area and development languages involved are typically fixed, which dramatically reduces the number of CWE categories involved, thereby reducing the difficulty involved in manually matching SBRs to CWE categories.

4.2.3 Codebook generation

Like Viviani et al. [52], we develop a codebook to guide the annotation process before starting the manual review. This codebook is developed by reviewing the 351 originally labeled SBRs of the five noisy datasets. Each of the six annotators reviews these SBRs and creates a codebook independently. The codebook includes two elements:

a) Reason of being an SBR: annotators need to write down why the bug report is an SBR according to their experience and expertise knowledge. They are suggested to describe the possible security risks or consequences that SBR may cause once it occurs.

b) Evidence words or phrases: instead of copying all the content of the bug report description, annotators select the words or phrases that best show that the bug report is an SBR and record them.

After that, the same annotators merged the six codebooks and removed the duplicates. Figure 2 provides an example snippet of the codebook. The entire content of the codebook is available at https://github.com/NWPU-IST/sbrbench/tree/master/Codebook.

4.2.4 Manual annotation and card sorting

On the basis of CWE software vulnerability categories and the codebook, we conduct the manual review with the help of card sorting [53], [54] approach, namely, each dataset is reviewed and labeled by two corporation employees and one Ph.D. student independently. According to their experience, annotators A1, A4, and A6 work on the four small-sized datasets (i.e, Ambari, Camel, Derby, and Wicket) and A2, A3, and A5 work on Chromium. As a result, there are three cards for each bug report, and we get a total of 137,820 cards for all the five datasets. We compute the Fleiss's Kappa Coefficient [52], [55] to measure the agreement among the three annotators of each dataset.

4.3 Our data annotation results

Based on our manual annotation results, there are totally 118 inconsistent label results in the five datasets. Fleiss kappa is a group statistic that calculates the average agreement between the multiple annotators. The agreement level of Fleiss kappa for all the five datasets are *Substantial* or *Almost*

TABLE 3
Background of the six annotators. (Note: four of them are software engineerings of Huawei corporation, and the other two are Ph.D. students that focus on software security analysis.

Annotator	Role	Experience (Year)			Note
		Software develop	Projects	Security bug	
A1	Senior Developer	15	3	3	worked as Hadoop-related products developer. is familar with Apache products Ambari, Camel, Derby and Wicket.
A2	Senior Developer	8	4	1	is familar with web browser devemlpment; volunteer of Chromium project.
A3	Test Manager	11	5	6	worked as a security tester for around 5 years; in charge of bug reports analysis, and CVE and CWE data tracking and analysis.
A4	Security Tester	4	3	4	conducts security testing through fuzzing, penetration, etc.
A5	Ph.D. student	3	2	4	worked 3 years as a security tester of web application; current research is focused on security bug report analysis.
A6	Ph.D. student	0	2	4	focuses on vulnerability mining; received more $30,000 in prize money from security testing competition in recent 3 years.

Reason of being an SBR	Evidence words or phrases	Bug ID
sensitive data leakage or exposure	• OWASP has some suggestion on how to make it harder for an attacker to crack hashed passwords, service doesn't follow all the suggestions. It doesn't add a random salt and it only performs the hash operation once	Derby-5539
	• password leakage; a problem with the display password	Chromium-1758
	• you can consider adding the master password	Chromium-1785
	...	...
might cause system crash or denial of service attack	• memory corruption vulnerability in Chrome,track down the root cause as this seems to be exploitable	Chromium-11308
	• memory corruption on dragging file to a new tab	Chromium-12027
	• wrong cache causes no storage damage	Chromium-27509
	...	...
...	...	...

Fig. 2. A snippet of codebook list after merging.

perfect (the values of Fleiss kappa ranges from 0.73 to 0.87 across the five datasets) [6], which indicates a high agreement level among annotators [52]. The details of manual annotation results are shared at https://github.com/NWPU-IST/sbrbench/tree/master/ManualAnnotation.

For bug reports that have obtained three consistent labels, we directly use the labels of annotators as final results. However, for bug reports with inconsistent label results, the annotators organize a face-to-face review meeting to discuss until a consistent result is reached for each item. As a result, there are 749 NSBRs of noisy datasets confirmed as SBRs by our manual annotation. Specific to each dataset, there are 616, 27, 42, 91, and 37 newly identified SBRs for Chromium, Ambari, Camel, Derby, and Wicket, respectively. It is worth noting that no SBRs in noisy datasets have been confirmed as NSBRs, which indicates strong reliability of the SBRs in the noisy datasets. A possible reason is that the SBRs in bug tracking systems were usually reported by experienced security testers, who focused on or paid special attention to system security.

Most of the identified SBRs are obvious security issues that can be matched to a specific CWE category. Figure 3 lists four identified SBRs that belong to well-known vulnerability types (null pointer, improper memory handling, privacy leakage, improper permission, etc.). For example, the record Camel-286 is a *null pointer* related SBR. It belongs to CWE-465 (Pointer Issues), which occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit. Identifying the null pointer issue is critical to software safety [56], [57] because the null pointer dereference is just one step before crashing an application (best case scenario) or gaining privileged access to the computer by an attacker [58].

In comparison, our labeled datasets are cleaner than the

ID	Snippets of Title & Description
Camel -286	NullPointerException in CXF routes when there is an endpoint between router and service CXF endpoints. When an endpoint is added between a cxf router ...
Ambari-3135	Number of ExecutionCommandEntity objects keep growing and result in Out of memory on large cluster (100 nodes).Script to re-create the issue...
Chromium -8388	Privacy leak from using non-incognito history database to ... are likely outweighed by concern over sites' ability to use this information to track the user's history.
Chromium -29824	Heap stack and more contain read write executable permissions under linux ...These permissions allow for much easier memory corruption exploitation especially a rwx HEAP. An attacker can simply use javascript to perform a heap spray to defeat ...

Fig. 3. Examples of obvious SBRs that are identified in our data annotation. The red-colored texts are contents that help us to decide that a bug report is security-related (i.e., SBR).

6. Mapping of agreement level and the value of Fleiss kappa (K_p). Poor: K_p <0; Slight: 0.01 $\leq K_p \leq$ 0.20; Fair: 0.20 <$K_p \leq$ 0.40; Moderate: 0.40 <$K_p \leq$ 0.60; Substantial: 0.60 <$K_p \leq$ 0.80; Almost perfect: 0.80 <$K_p \leq$ 1.

original ones. Therefore, we call our labeled datasets as clean datasets and the original ones as noisy datasets.

Table 4 shows the distribution of the noisy datasets as well as our clean datasets. There is only a small group of bug reports labeled as SBRs in the five datasets for the noisy datasets. The percentages of SBRs range from 0.45% to 8.80%. The percentage of SBRs for clean datasets ranges from 1.93% to 17.90%. Especially for the large-scale dataset Chromium, we identified 616 SBRs, which were labeled as NSBRs in the noisy dataset, and the percentage of SBRs is increased by 320%.

5 Experimental Setup

In this section, we introduce the baseline approaches proposed by previous studies (i.e., the work of Peters et al. and Shu et al.). After that, we describe the setting of simple text classification, which is used for investigating RQ2. Finally, the classifiers, as well as the performance evaluation metrics of our study, are introduced.

5.1 Baseline approaches

We use the approaches proposed by Peters et al. and Shu et al., which we have introduced in Section 3, as baselines for validating the impact of data label correctness. We call the work of Peters et al. as Farsec by following the name of their work. For the work of Shu et al. [12], we obtained two baseline approaches from their work as they tune the hyperparameters of learner and SMOTE, respectively, in their study. Due to the fact that they directly use the keywords-matrix files of the training set and testing set built by Farsec in their study, we name the approach that tunes the parameters of learner as Farsec$^{Tuned}_{Learner}$, and the approach that tunes the parameters of SMOTE as Farsec$^{Tuned}_{Smote}$.

We describe the setting of these three baselines as follows:

- Farsec: a framework for filtering and ranking bug reports for reducing the presence of security-related keywords. Before fitting the prediction models, $Farsec$ generates security keywords matrices and removes non-security bug reports with security related keywords with different filters (shown in Table 1). We use filter *farsectwo* in our study as 80% of the best results across the five datasets are obtained with *farsectwo* in Peters et al.'s study. Farsec is a ranking approach. It first generates a keywords matrix based on the top X security-related keywords for each dataset. To select an optimal value for X, we conducted experiments (the inner-project experiments of Peters et al.'s work) with X = 50, 100, and 200. The results showed the average F1-score of the three settings is the same (0.15) while the G-measure achieved with setting 100 is the best (0.29). Therefore, in our study we use top 100, which is in line with the setting of Peters et al.'s study [11].
- Farsec$^{Tuned}_{Learner}$: using the processed security keywords matrix files as input (i.e., the training set and testing set), and tuning key parameters of learner with differential evolution algorithm [59].
- Farsec$^{Tuned}_{Smote}$: using the processed security keywords matrix files as input (i.e., the training set and testing set), and tuning the key parameters of SMOTE with differential evolution.

To be objective, we directly use the sourcecode shared by Shu et al. and keep all the settings of the parameters (i.e., key parameters, the default values, and the tuning range of each key parameter) as they are in their work. The three baselines apply the same classification algorithms (Random Forest, Naive Bayes, K-Nearest Neighbour, Multilayer Perceptron, and Logistical Regression) and datasets (Chromium, Ambar, Camel, Derby, and Wicket) in their studies.

5.2 Simple text classification

The three baseline approaches use the top 100 security keywords matrix to construct classification models. This is different from normal text classification. In this paper, we also use the simple text classification as another approach for our experimental evaluation as it is popular and applied by most bug report analysis [13], [14], [16], [17], [27]. We simply preprocess the text of bug report *Description* with basic text preprocess approach *CountVectorizer* and *SelectFromMode* integrated in machine learning package scikit-learn [60] for text tokenization and dimension reduction.

CountVectorizer converts the text of bug report *Description* to a matrix of token counts. Stop words like *a*, *the*, *and*, which are presumed to be uninformative in representing the content of a text, are removed to avoid them being constructed as a signal for prediction. After that, a sparse representation matrix of the token counts is produced. SelectFromModel is a feature selection approach. It evaluates feature importance and selects features based on importance weights. It is a meta-transformer that can be used along with any estimator that has a *coef* or *feature_importances* attribute, which denotes the weights assigned to the features. The features are considered unimportant and removed if the corresponding *coef* or *feature_importances* values are below the provided threshold value. Apart from specifying the threshold numerically, there are built-in heuristics for finding a threshold using a string argument. Available heuristics are *mean*, *median* and float multiples of these like *0.1*mean*. We use the default values for the parameters of CountVectorizer and SelectFromModel.

Although the pre-processing result of simple text classification (using CountVectorizer and SelectFromModel) is a matrix, this matrix is different from the security keywords-matrix generated in Farsec. The column of the security keywords-matrix of Farsec is the top 100 terms extracted from SBRs of the training set, while the row of matrix produced by simple text classification is generated with terms of all bug reports in the training set.

5.3 Classifiers and performance metrics

To make the evaluation results be objective, we use the classifiers and performance metrics applied by Peters et al. and Shu et al. in our study.

Classifiers. To be objective, we directly use the five classifiers applied by Peters et al. and Shu et al. in this study. Namely, Random Forest (RF), Naive Bayes (NB), K-Nearest Neighbor (KNN), Multilayer Perceptron (MLP), and

TABLE 4
Distribution of the five datasets on both the noisy and the clean version.

Dataset	# BR	SBR		NSBR	
		# Noisy (%)	# Clean (%)	# Noisy (%)	# Clean (%)
Chromium	41,940	192 (0.46%)	808 (1.93%)	41,748 (99.54%)	41,132 (98.07%)
Ambari	1,000	29 (2.90%)	56 (5.60%)	971 (97.10%)	944 (94.40%)
Camel	1,000	32 (3.20%)	74 (7.40%)	968 (96.80%)	926 (92.60%)
Derby	1,000	88 (8.80%)	179 (17.90%)	912 (91.20%)	821 (82.10%)
Wicket	1,000	10 (1.00%)	47 (4.70%)	990 (99.00%)	953 (95.30%)

Logistical Regression (LR). These classifiers are also widely applied in software data repository mining [61], [62]. However, to make the comparison clearer and easier, we only use classifier RF, combining with baseline approaches to answer RQ1. We choose RF [63] as the primary classifier because (1) RF is one of the best classifiers in experiment results of baseline Farsec; (2) RF is one of the most commonly used classification techniques and performs well for text classification of software engineering [52], [54].

Performance metrics. To avoid bias and give a comprehensive evaluation, we use all the performance evaluation metrics applied by the work of Peters et al. and Shu et al. These metrics include Recall (i.e., pd in their work), pf (probability of false alarm), Precision, F1-score, and G-measure. The first four are commonly used metrics in the area of empirical software engineering [61], [64], while G-measure is introduced as a major metric in Peters et al.'s study [11]. F1-score and G-measure are both harmonic means and G-measure considers the Recalls of both the majority and the minority classes [11].

As for each bug report, there are four possible outcomes of the prediction result:

- True Positive (TP): an SBR is predicted as SBR;
- False Negative (FN): an SBR is predicted as NSBR;
- True Negative (TN): an NSBR is predicted as NSBR;
- False Positive (FP): an NSBR is predicted as SBR.

Based on these outcomes, the performance metrics Recall, pf, Precision, F1-score, and G-measure can be calculated as below.

$$Recall = pd = \frac{TP}{TP + FN} \tag{1}$$

$$pf = FPR = \frac{FP}{FP + TN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{4}$$

$$G - measure = \frac{2 * Recall * (1 - pf)}{Recall + (1 - pf)} \tag{5}$$

Among these five performance metrics, Recall, Precision, F1-score, and G-measure are the higher, the better, while pf is the lower, the better.

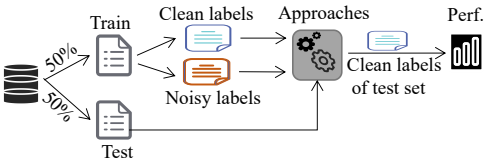

Fig. 4. Data setting and basic process of experiments.

5.4 Data setting

To ensure the fairness of comparison, we follow the way of baseline Farsec for the division of training set and testing set. That is, each dataset is sorted chronologically, and then it is divided into two equal parts (i.e., 50% and 50%). The first part is used as the training set, and the latter is used as the testing set, which is in line with the actual application scenario in production. The two baselines of Shu et al. (i.e., Farsec$^{Tuned}_{Learner}$ and Farsec$^{Tuned}_{Smote}$) also follow this dataset division method since they directly use the matrix files processed by Peters et al.

Figure 4 illustrates our experiment setting. We perform different classification approaches (i.e., Farsec, Farsec$^{Tuned}_{Learner}$, Farsec$^{Tuned}_{Smote}$, Text, Text$^{Tuned}_{Learner}$ and Text$^{Tuned}_{Smote}$) either on the noisy data or the clean data. However, we measure the performance of each approach with clean labels of the test set - including the model trained with noisy data. We do this because what really matters is whether the prediction results are correct or not in the sense of the truth, i.e., clean data.

6 Experiment Results

In the section, we present our experiment results by answering the two research questions.

6.1 Response to RQ1

RQ1: To what extent can dataset label correctness impact the performance of classification models?

To answer RQ1, we perform the three baseline approaches (i.e., Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$) with classifier RF on both the noisy datasets and the clean datasets.

Table 5 presents the performance results of the three baselines trained with both the noisy and the clean datasets. Training with the noisy datasets, average scores of Recall, Precision, F1-score, and G-measure across the five datasets are 0.15, 0.35, 0.16, and 0.22 in Farsec; 0.18, 0.25, 0.18, and 0.26 in Farsec$^{Tuned}_{Learner}$; and 0.23, 0.12, 0.14, and 0.32 in Farsec$^{Tuned}_{Smote}$. Trainning with the clean datasets, the average

TABLE 5
Performance results of the three baseline approaches with classifier RF on both the Noisy and Clean datasets. All labels of testing set are from the the Clean datasets. The value of Clean datasets is bold-faced if it is better than that of the Noisy dataset (Note: the higher the better for Recall, Precision, F1-score, and G-measure; the lower the better for pf).

Dataname	Approach	Recall		pf		Precision		F1-score		G-measure	
		Noisy	Clean	Noisy	Clean	Noisy	Clean	Noisy	Clean	Noisy	Clean
Chromium	Farsec	0.01	**0.66**	0.00	0.00	0.83	**0.95**	0.02	**0.78**	0.02	**0.79**
	Farsec$^{Tuned}_{Learner}$	0.03	**0.67**	0.00	0.00	0.39	**0.94**	0.05	**0.78**	0.05	**0.80**
	Farsec$^{Tuned}_{Smote}$	0.30	**0.80**	0.17	**0.07**	0.04	**0.22**	0.07	**0.35**	0.45	**0.86**
Ambari	Farsec	0.44	**0.50**	0.02	0.07	0.39	0.19	0.41	0.28	0.60	**0.65**
	Farsec$^{Tuned}_{Learner}$	0.38	**0.56**	0.02	0.06	0.35	0.23	0.36	0.33	0.54	**0.70**
	Farsec$^{Tuned}_{Smote}$	0.44	0.44	0.05	0.06	0.22	0.19	0.29	0.26	0.60	0.60
Camel	Farsec	0.07	**0.33**	0.02	0.12	0.27	0.21	0.11	**0.26**	0.12	**0.48**
	Farsec$^{Tuned}_{Learner}$	0.11	**0.24**	0.04	0.05	0.21	**0.33**	0.14	**0.28**	0.20	**0.38**
	Farsec$^{Tuned}_{Smote}$	0.07	**0.33**	0.06	0.13	0.11	**0.20**	0.08	**0.25**	0.12	**0.47**
Derby	Farsec	0.24	**0.84**	0.18	0.54	0.25	**0.27**	0.25	**0.41**	0.38	**0.60**
	Farsec$^{Tuned}_{Learner}$	0.38	**0.69**	0.22	0.44	0.30	0.27	0.33	**0.39**	0.51	**0.62**
	Farsec$^{Tuned}_{Smote}$	0.33	**0.60**	0.28	0.25	0.22	**0.37**	0.27	**0.45**	0.45	**0.67**
Wicket	Farsec	0.00	**0.52**	0.00	0.05	0.00	**0.34**	0.00	**0.41**	0.00	**0.67**
	Farsec$^{Tuned}_{Learner}$	0.00	**0.52**	0.00	0.08	0.00	**0.24**	0.00	**0.33**	0.00	**0.67**
	Farsec$^{Tuned}_{Smote}$	0.00	**0.39**	0.00	0.05	0.00	**0.26**	0.00	**0.32**	0.00	**0.55**
Average	Farsec	0.15	**0.57**	0.04	0.16	0.35	**0.40**	0.16	**0.43**	0.22	**0.64**
	Farsec$^{Tuned}_{Learner}$	0.18	**0.54**	0.06	0.13	0.25	**0.40**	0.18	**0.42**	0.26	**0.63**
	Farsec$^{Tuned}_{Smote}$	0.23	**0.51**	0.11	0.11	0.12	**0.25**	0.14	**0.33**	0.32	**0.63**
	All	0.19	**0.54**	0.07	0.13	0.24	**0.35**	0.16	**0.39**	0.27	**0.63**

values of Recall, Precision, F1-score, and G-measure across the five datasets reach 0.57, 0.40, 0.43, and 0.64 in Farsec; 0.54, 0.40, 0.42, and 0.63 in Farsec$^{Tuned}_{Learner}$; and 0.51, 0.25, 0.33, and 0.63 in Farsec$^{Tuned}_{Smote}$. Comparing the performance after training with clean and noisy datasets, we have the following observations:

(1) The maximum values of Recall, Precision, F1-score, and G-measure of the baselines on the clean datasets are higher than that on the noisy dataset.

(2) The average values of Recall, Precision, F1-score, and G-measure are increased by 191%, 46%, 147%, and 136%, respectively.

Finding I

For the same classifier, the performance on the clean datasets is much higher than that of the noisy datasets.

6.2 Response to RQ2

RQ2: How does simple text classification perform on the clean datasets for SBR prediction?

To answer RQ2, we perform the five classification algorithms (i.e., RF, NB, MLP, LR, and KNN) with simple text classification. We first train each of the five classification models with a noisy dataset and its corresponding clean dataset respectively. Then, we use the model to predict the testing set of the project. The performance metrics of the model are then calculated with the clean data labels.

Table 6 presents the performance results of the three baselines trained with both the noisy and the clean datasets. The performance scores corresponding to the clean dataset is highlighted in boldface if it is better than that of the noisy dataset. Note that the overall values of Recall, Precision, F1-score, and G-measure on the clean version of the five datasets are much higher than that of the noisy version. The average values of performance metrics Recall, Precision, F1-score, and G-measure are 0.08, 0.29, 0.11, and 0.14 for models trained on the noisy datasets, and 0.37, 0.54, 0.42, and 0.51 for models trained on the clean datasets, which are 362%, 86%, 281%, and 264% higher than that of the noisy datasets. In particular, the improvement on the large-scale dataset Chromium is substantial. The average F1-score with the five classifiers trained on the clean Chromium dataset is 654% higher than the average F1-score on the noisy Chromium dataset.

Finding II

Simple text classification performs better on the clean datasets than the noisy datasets.

For Precision, our experiment results for the simple text classification contradicts Tantithamthavorn et al.'s conclusion as in their work, the mislabeling rarely impacts the Precision of prediction models [4]. We further explored the reasons and found that in Tantithamthavorn's work, they rebalanced the training data to combat class imbalance, which

TABLE 6
Performance results of the five classifiers with simple text classification on both the Noisy and Clean datasets. All labels of testing set are from the the Clean datasets. The value of Clean datasets is bold-faced if it is better than that of the Noisy datasets (Note: the higher the better for Recall, Precision, F1-score, and G-measure; the lower the better for pf).

Dataname	Learner	Recall		pf		Precision		F1-score		G-measure	
		Noisy	Clean	Noisy	Clean	Noisy	Clean	Noisy	Clean	Noisy	Clean
Chromium	RF	0.01	**0.72**	0.00	0.00	0.99	0.92	0.02	**0.81**	0.02	**0.84**
	NB	0.15	**0.66**	0.02	0.04	0.17	**0.29**	0.16	**0.40**	0.26	**0.79**
	MLP	0.06	**0.49**	0.00	0.00	0.33	**0.71**	0.10	**0.58**	0.11	**0.66**
	LR	0.04	**0.58**	0.00	0.00	0.37	**0.84**	0.07	**0.69**	0.08	**0.73**
	KNN	0.04	**0.58**	0.00	0.00	0.37	**0.84**	0.07	**0.69**	0.08	**0.73**
Ambari	RF	0.13	**0.25**	0.02	0.02	0.20	**0.31**	0.15	**0.28**	0.22	**0.40**
	NB	0.13	**0.44**	0.02	0.02	0.15	**0.39**	0.14	**0.41**	0.22	**0.60**
	MLP	0.31	**0.38**	0.03	**0.02**	0.28	**0.38**	0.29	**0.38**	0.47	**0.54**
	LR	0.19	**0.25**	0.01	0.01	0.38	**0.40**	0.25	**0.31**	0.32	**0.40**
	KNN	0.19	**0.25**	0.01	0.01	0.38	**0.40**	0.25	**0.31**	0.32	**0.40**
Camel	RF	0.00	**0.37**	0.00	0.00	0.00	**0.89**	0.00	**0.52**	0.00	**0.54**
	NB	0.04	**0.30**	0.05	0.05	0.09	**0.37**	0.06	**0.33**	0.08	**0.46**
	MLP	0.00	**0.20**	0.02	0.03	0.00	**0.43**	0.00	**0.27**	0.00	**0.33**
	LR	0.02	**0.15**	0.00	0.02	0.33	**0.39**	0.04	**0.22**	0.04	**0.26**
	KNN	0.02	**0.15**	0.00	0.02	0.33	**0.39**	0.04	**0.22**	0.04	**0.26**
Derby	RF	0.10	**0.46**	0.02	**0.01**	0.59	**0.90**	0.18	**0.61**	0.19	**0.63**
	NB	0.16	**0.53**	0.06	0.13	0.39	**0.50**	0.23	**0.51**	0.28	**0.66**
	MLP	0.14	**0.39**	0.03	0.07	0.50	**0.56**	0.22	**0.46**	0.25	**0.55**
	LR	0.12	**0.39**	0.01	0.06	0.67	0.60	0.21	**0.48**	0.22	**0.55**
	KNN	0.12	**0.39**	0.01	0.06	0.67	0.60	0.21	**0.48**	0.22	**0.55**
Wicket	RF	0.00	**0.48**	0.00	0.00	0.00	**0.92**	0.00	**0.63**	0.00	**0.65**
	NB	0.00	**0.30**	0.00	0.03	0.00	**0.30**	0.00	**0.30**	0.00	**0.46**
	MLP	0.00	**0.22**	0.00	0.02	0.00	**0.36**	0.00	**0.27**	0.00	**0.36**
	LR	0.00	**0.13**	0.00	0.01	0.00	**0.38**	0.00	**0.19**	0.00	**0.23**
	KNN	0.00	**0.13**	0.00	0.01	0.00	**0.38**	0.00	**0.19**	0.00	**0.23**
Average	RF	0.05	**0.46**	0.01	0.01	0.36	**0.79**	0.07	**0.57**	0.09	**0.61**
	NB	0.10	**0.45**	0.03	0.05	0.16	**0.37**	0.12	**0.39**	0.17	**0.59**
	MLP	0.10	**0.33**	0.02	0.03	0.22	**0.49**	0.12	**0.39**	0.17	**0.49**
	LR	0.07	**0.30**	0.01	0.02	0.35	**0.52**	0.11	**0.38**	0.13	**0.44**
	KNN	0.07	**0.30**	0.01	0.02	0.35	**0.52**	0.11	**0.38**	0.13	**0.44**
	All	0.08	**0.37**	0.01	0.03	0.29	**0.54**	0.11	**0.42**	0.14	**0.51**

will impact the Precision of prediction models [65], [66]. In fact, their conclusion is close to the results of our baseline approaches (i.e., Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$), and these baselines alleviate class imbalance by filtering out noise from the majority category or applying SMOTE.

Figure 5 shows the boxplots of Recall, Precision, F1-score, and G-measure of approaches Text, Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$ trained on the clean and noisy datasets. The scores for the clean datasets are shown as red boxes, while the scores for the noisy datasets are shown as blue boxes. From the plots of Precision, we can observe the Precision of the three baseline approaches on the clean data and noisy data are similar, which is in line with the conclusion of Tantithamthavorn et al. 's work [4].

Comparing the performance of simple text classification with the three baselines (i.e., Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$) trained on the clean datasets, the Precision is significantly improved with an increase of 125% on average. However, the three baselines perform better in terms of Recall. In the context of SBR prediction, Recall measures the percentage of true SBRs identified from total SBRs of target project, while Precision measures the percentage of true SBRs of the total identified SBRs. Recall and Precision are both important metrics for SBR prediction. Therefore, we use F1-score as the most important evaluation metric to avoid bias. F1-score is the harmonic mean of Recall and Precision - it evaluates if an increase in Precision (Recall) outweighs a reduction in Recall (Precision) [67]. In this paper, we use F1-score as the most important evaluation metric to avoid bias. From the boxplots of F1-score, on the clean datasets (red), the box of simple text classification (txt) is much higher than those of the three baselines (fsc, ftl, and fts). On average, Simple text classification is 46% higher than the three baselines in terms of F1-score.

Finding III

With the clean datasets, simple text classification outperforms the three baseline approaches.

7 Discussion

In this section, we discuss if the hyperparameter tuning method proposed by Shu et al. works while combining simple text classification, the patterns of mislabeling in the original datasets, performance levels of different annotators (engineers versus Ph.D. students), the implications of our study, and threats to the validity of our study.

7.1 Do the hyperparameter tuning approaches proposed by Shu et al. work on simple text classification?

We investigate whether the hyperparameter tuning approach proposed by Shu et al. works when combining with the simple text classification techniques proposed in RQ2.

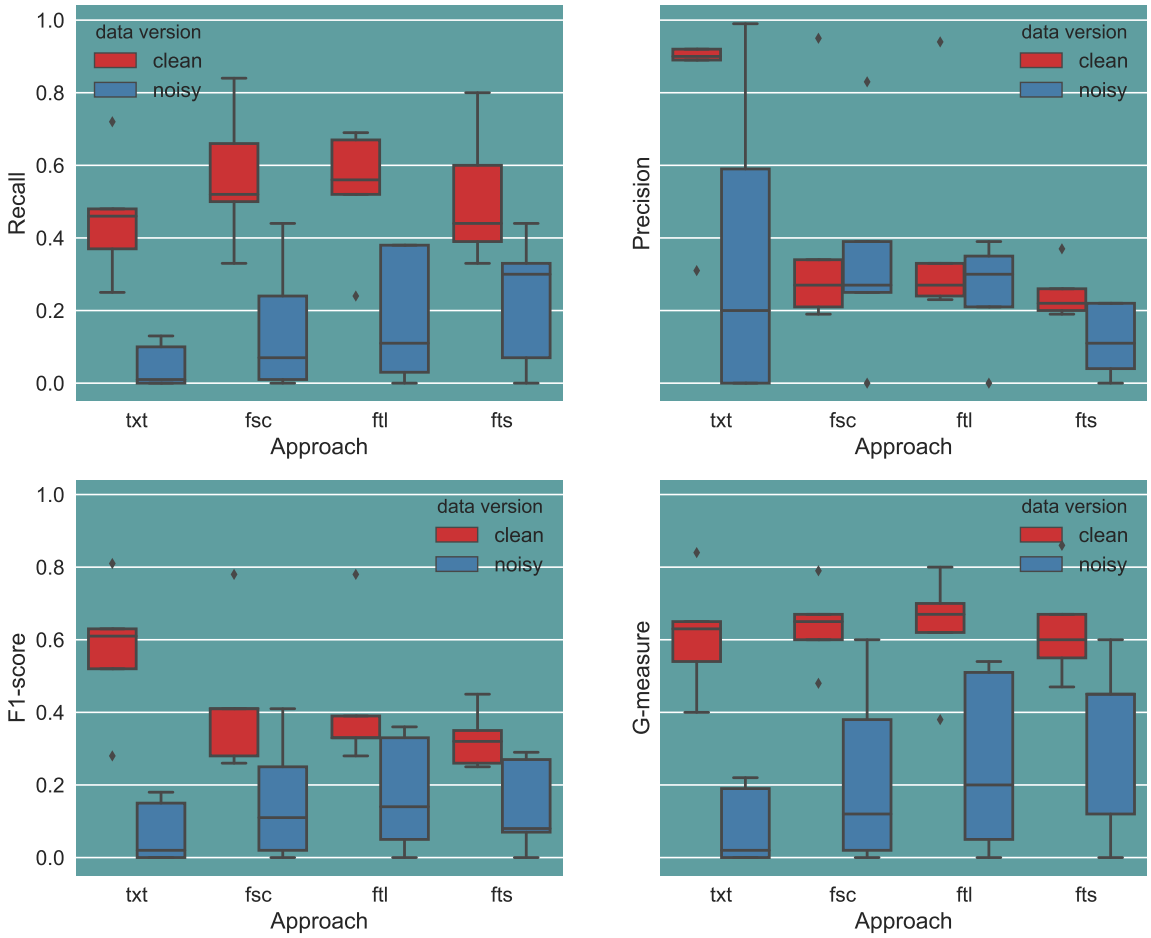

Fig. 5. Performance comparison between the clean datasets (red) and noisy datasets (blue) for simple text classification and the three baselines (i.e., Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$). (Note that the **txt, fsc, ftl, and fts** are short for Text, Farsec, Farsec$^{Tuned}_{Learner}$, and Farsec$^{Tuned}_{Smote}$, respectively).

Here, we apply the two hyperparameter tuning approaches proposed by Shu et al. with RF. Similar to the naming rule of our baseline approaches, we use Farsec$^{Tuned}_{Learner}$ and Text$^{Tuned}_{Smote}$ to indicate the combination of text classification with the learner tuning (tuning the key parameters of learner RF) and the combination of text classification with the Smote tuning (tuning the key parameters of Smote), respectively.

Table 7 shows the results of combining text classification with hyperparameter tuning (i.e., tune the learner and SMOTE respectively). The best value of Recall, Precision, F1-score, and G-measure achieves 0.76, 0.94, 0.84, and 0.89 across the five datasets; and the average of Recall, Precision, F1-score, and G-measure are 0.56, 0.58, 0.52, and 0.68 respectively. Farsec$^{Tuned}_{Learner}$ (combination of simple text classification and hyperparameter tuning of learner) substantially improves Precision, while Farsec$^{Tuned}_{Smote}$ (combination of simple text classification and hyperparameter tuning of SMOTE) is more effective for improving Recall. Considering the importance of harmonic metric F1-score, Farsec$^{Tuned}_{Learner}$ outperforms Farsec$^{Tuned}_{Smote}$ on average.

To further compare the effectiveness of Farsec$^{Tuned}_{Learner}$ and Farsec$^{Tuned}_{Smote}$ with the four approaches involved in answering RQ1 and RQ2, we use boxplots to show the performance values of the six approaches across the five clean datasets, as shown in Figure 6. Paired color is used to show each group, namely, the blue pair represent two basic approaches (Farsec and Text), the green pair are approaches with hyperparameter tuning of learner (Farsec$^{Tuned}_{Learner}$ and Text$^{Tuned}_{Learner}$), and the red pair are approaches with hyperparameter tuning of SMOTE (Farsec$^{Tuned}_{Smote}$ and Text$^{Tuned}_{Smote}$). We can observe that Farsec$^{Tuned}_{Learner}$ (green) is much better than Farsec$^{Tuned}_{Learner}$ (light green) in terms of Precision and F1-score, which is in line with comparing Text (blue) with Farsec (light blue). However, Text$^{Tuned}_{Smote}$ (red) consistently outperforms Farsec$^{Tuned}_{Smote}$ (light red) for all the four

TABLE 7
Effectiveness of hyperparameter optimization approaches proposed by Shu et al. combined with simple text classification.

Dataset	Approach	Recall	pf	Precision	F1-score	G-measure
Chromium	$\text{Text}^{Tuned}_{Learner}$	0.76	0.00	0.94	0.84	0.87
	$\text{Text}^{Tuned}_{Smote}$	0.86	0.08	0.17	0.28	0.89
Ambari	$\text{Text}^{Tuned}_{Learner}$	0.25	0.02	0.31	0.28	0.40
	$\text{Text}^{Tuned}_{Smote}$	0.50	0.02	0.50	0.50	0.66
Camel	$\text{Text}^{Tuned}_{Learner}$	0.39	0.00	0.95	0.55	0.56
	$\text{Text}^{Tuned}_{Smote}$	0.41	0.09	0.32	0.36	0.57
Derby	$\text{Text}^{Tuned}_{Learner}$	0.52	0.01	0.93	0.66	0.68
	$\text{Text}^{Tuned}_{Smote}$	0.68	0.25	0.40	0.50	0.71
Wicket	$\text{Text}^{Tuned}_{Learner}$	0.52	0.01	0.71	0.60	0.68
	$\text{Text}^{Tuned}_{Smote}$	0.70	0.03	0.57	0.63	0.81
Average	$\text{Text}^{Tuned}_{Learner}$	0.49	0.01	0.77	0.59	0.64
	$\text{Text}^{Tuned}_{Smote}$	0.63	0.09	0.39	0.45	0.73
	All	0.56	0.05	0.58	0.52	0.68

performance indicators (Recall, Precision, F1-score, and G-measure). When comes to the key performance indicator F1-score, $\text{Text}^{Tuned}_{Learner}$ is slightly higher than Text and becomes the best of the six approaches.

7.2 Patterns of mislabeling in the original datasets

Manual data annotation is an exhausting task. To shed light on further automated data cleaning and data labeling work, this section tries to summarize the patterns of mislabeling based on analyzing the manual identified SBRs.

7.2.1 Prevalence

There are totally 749 SBRs identified from the mislabeling of noisy datasets by the manual review. One or multiple CWE tag(s) are assigned to each record. These tags allowed us to understand the distribution of mislabeling items.

More than 50 CWE categories are involved in the 749 SBRs. However, to extract the most common characteristics from these SBRs, we group them from the 40 top-level CWE categories of software development. Finally, CWE-1218 (Memory buffer error), CWE-199 (Information management error), and CWE-465 (Pointer Issues) are the top three categories with the largest number of SBRs. They take up about 75% of all SBRs. Amongst this, around 90% records of CWE-1218 and CWE-199 are contributed by Chromium, while 95% records of CWE-465 are from the four small-sized datasets.

7.2.2 Patterns of description

Bug description usually describes the observed behavior (OB) and/or the expected behavior (EB) with natural language. Chaparro et al. [26] summarized the most common patterns of OB and EB:

- **OB:** *([subject]) [negative aux. verb] [verb] [complement].* Here, [negative aux. verb] $\in$ {are not, can not, does not, did not, etc. }. For example, [Audio Output Stream's shutdown code] [does not] [correctly delete the output stream object after the thread is shutdown...] (from Chromium Issue 16036).
- **EB:** *[subject] should/shall (not) [complement].* For example, [It] shouldn't [bother to call back to the UI thread since such a call won't actually do anything at this point.](from Chromium Issue 41547).

TABLE 8
Keywords summarized from mislabeling SBRs of the top-3 categories.

Category	Keywords	
	[Subject]	[verb]/[complement]
CWE-1218	memory, heap, stack, cache, buffer, pool, cpu, loop, size, range, index, array, file, path, data, exception, bound, consumption	out, comsume, leak, handle, corrupt, null, exceed, overflow, crash, uncontrol, dereference, use-after-free, out-of-bounds
CWE-119	password, cookie, log, cache, credential, user, username, session, profile, sandbox, privacy, security, license, proxy, host, certificate	leak, exposure, mask, storage, transfer, log, sensitive, clear, hash, permit, allow, invalid, malicious
CWE-465	pointer	dereference, reference, release, incorrect, improper, null, outside, invalid, exception, uninitialized, initialize, out-of-range, expired, handle

When it comes to the SBRs, there are some security-related keywords that appear in the *[subject]* and *[verb]/[complement]* of SBRs' description. Based on analyzing the 749 identified SBRs, we summarize some high-frequency words for the top-3 CWE categories (CWE-1218, CWE-199, and CWE-464), as shown in Table 8. SBRs belonging to each category always include a combination of words from both *[subject]* and *[verb]/[complement]*, or words that are with the same roots as these words.

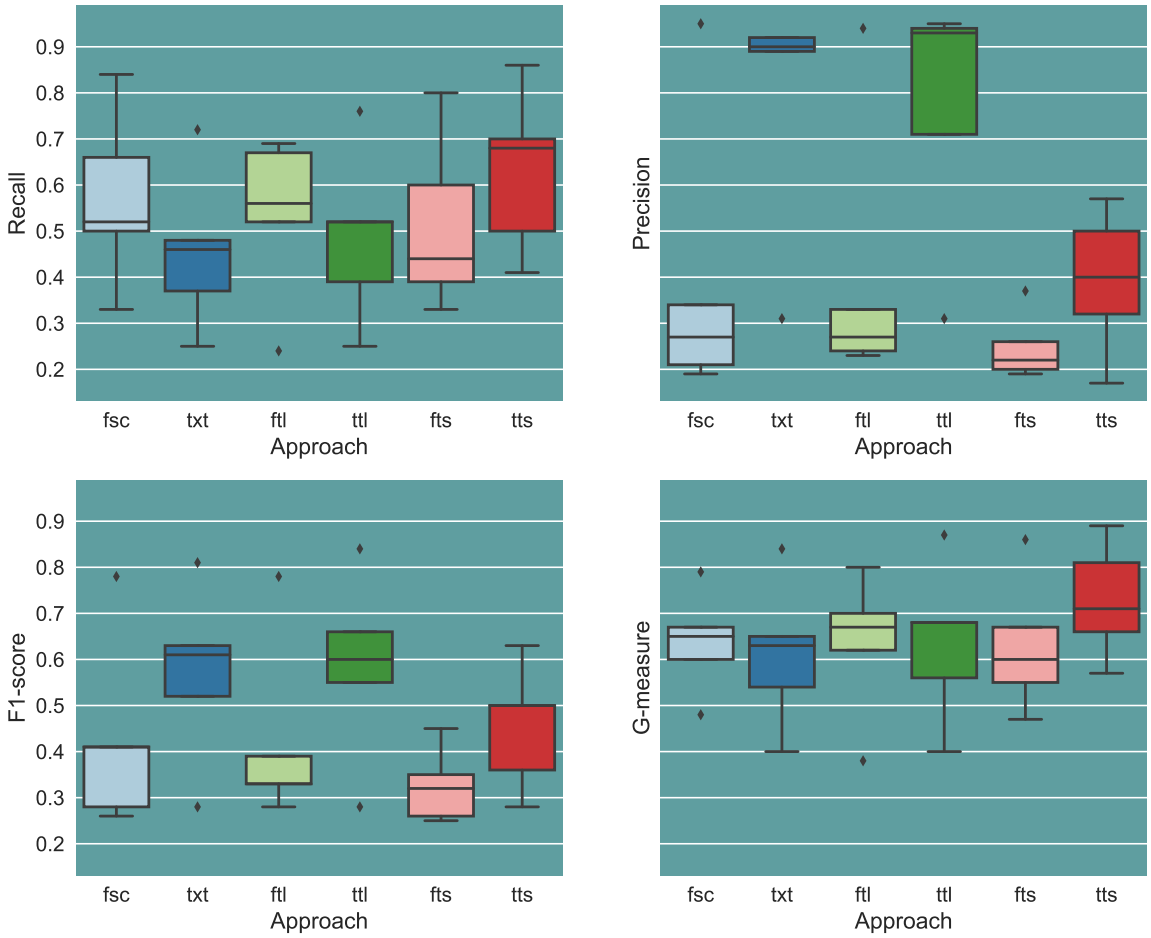

Fig. 6. Boxplots with *Paired* color for the performance of the six involved classification approaches on the clean datasets. The blue pair represent two basic approaches (Farsec and Text), the green pair are approaches with hyperparameter tuning of learner (Farsec$^{Tuned}_{Learner}$ and Text$^{Tuned}_{Learner}$), and the red pair are approaches with hyperparameter tuning of SMOTE (Farsec$^{Tuned}_{Smote}$ and Text$^{Tuned}_{Smote}$). (Note that the **fsc, txt, ftl, ttl, fts, and tts** are short for approaches Farsec, Text, Farsec$^{Tuned}_{Learner}$, Text$^{Tuned}_{Learner}$, Farsec$^{Tuned}_{Smote}$, and Text$^{Tuned}_{Smote}$, respectively).

7.3 Performance levels of annotators: engineers vs. Ph.D. students

Our clean datasets annotation involves six annotators, including four software engineers from Huawei corporation and two Ph.D. students. This section discusses the performance levels of annotators from corporation employees and university students. The annotation results are shared at https://github.com/NWPU-IST/sbrbench/tree/master/ManualAnnotation.

Based on each annotation result and the final labels of clean datasets, the overall accuracy of corporation annotators is better than that of Ph.D. students. However, Ph.D. student A5, who has three years of software testing experience and currently focuses on bug reports analysis, outperforms corporation annotator A3, who has eight years of software development experience and is familiar with project Chromium. That is to say, annotators with practical development and testing experience are more likely to give correct labels; however, experience on software security-related work contributes more to correct labelling.

7.4 Implications

From this work, we distill some general suggestions which are beyond the specific task and approaches.

Data quality matters. According to the investigation result of Kim et al. [68], data quality is the first challenge of data scientists in Microsoft. They point out that the data quality issue makes it difficult for data scientists to have high confidence about the correctness of their work.

As shown by our experiment results of SBR prediction, the effectiveness of SBR prediction models is strongly correlated with data label correctness. The performance of a model fit with clean data is much better than the model fit

with noisy data. Researchers have conducted similar validation in the area source code defect prediction [31], [68], valid bug prediction [32], clone detection [69] of software engineering. For example, Kim et al. [68] investigated the impact of data labels on defect prediction. Their evaluation results show that the prediction model's performance is clearly decreased while the mislabels over 25%. Tantithamthavorn et al. [4] showed the prediction models trained on noisy data achieve 56%-68% of the Recall of models trained on clean data.

Collective effort is required. The high-quality dataset is vital to proper experimental evaluation of approaches for SBR prediction. For SBR prediction dataset labeling, manual effort is required because the current automatic approach can not handle this well. Although manual data labeling is difficult and expensive, it can still be overcome with collective effort. There are many successful cases of constructing high-quality datasets with collective efforts. For example, Svajlenko et al. [69], [70] built a large-scale benchmark for clone detection by mining and then manually checking clones of ten common functionalities. The benchmark contains six million true positive clones of different clone types: Type-1, Type-2, Type-3, and Type-4. These clones were found by three judges over 216 hours of manual validation efforts. These datasets are widely-used by researchers [71], [72], [73].

In the area of SBR prediction, collective effort is required for correctly labeling data, and construct high-quality publicly available datasets. Our work is one of the steps towards achieving that goal. We have released our datasets and manual annotation files publicly for others to verify and extend[7].

7.5 Threats to validity

Internal Validity. A threat to the internal validity of this study is the potential mislabels in the clean datasets. We take several measures to reduce this threat. However, it is still challenging to guarantee that there are no false positives and false negatives in the clean datasets.

As Ohira et al. [19] noted, it is hard to judge whether a bug report is an SBR or NSBR when there is no clear definition for SBR. We use the definitions of CWE, which is the most authoritative organization of vulnerability management, as a basis to judge whether a bug report is an SBR. Furthermore, all six annotators conducting the manual review have rich practical experience and a deep understanding of different vulnerability types and characteristics.

Besides, Viviani et al. [52] present a good practice of data annotation; we follow their approach to generate a codebook by reviewing 351 known SBRs of the five projects. This codebook record the reasons why a bug report is labeled as SBR and the exact sentences and phrases that support this decision. We use the codebook as guidance of the following review process, and the card sorting approach is also applied to guarantee the correctness of label results further.

Another threat to the internal validity is the impact of tuning for the findings. To mitigate this threat, a set of turning approaches have been considered in our experiments, including selecting optimal values for Farsec (i.e., use filter farsectwo and top 100 security-related keywords), tuning the key parameters of classifiers (i.e., $\text{Farsec}^{Tuned}_{Learner}$ and $\text{Text}^{Tuned}_{Learner}$), and tuning parameters of over-sampling approach SMOTE (i.e., $\text{Farsec}^{Tuned}_{Smote}$ and $\text{Text}^{Tuned}_{Smote}$). Experiment results show these tuning approaches have little impact on the findings. However, other tuning approaches like feature selection, under-sampling approaches could be considered in the future [74].

External Validity. Threats to external validity are concerned with the generality of our findings. In this study, we evaluate the impact of data label correctness by comparing the performance of the same SBR prediction models on the noisy datasets with that of the clean datasets. The three baseline methods extracted from two recent SBR prediction work proposed by Peters et al. and Shu et al., and the simple text classification models are applied. Moreover, a set of performance indicators (all the performance indicators involved in their studies), including Recall, pf, Precision, F1-score, and G-measure, are used for the evaluation.

Another external threat is the generalization of the findings. This study investigates the impact of data quality on SBR prediction. We do not claim that our findings can be generalized to all software analytics tasks. However, In the area of MSR, data annotation is involved in many tasks. For example, sourcecode defect prediction [4], [8], [31], high-impact bug report prediction [27], [32], [34], and assessing software productivity and quality [68]. Data quality is one of the key factors determining a model's effectiveness for prediction model-involved tasks [64]. Most previous studies show that the label correctness of training data impacts the performance of classification models [31], [33], [68], which is in line with our findings. We think the findings of this paper are possible to be generalized to MSR scenarios with similar characteristics as SBR prediction. These characteristics include but are not limited to: (1) the scenario involves significant class imbalance; (2) the problem-solving approach is based on text mining. High-impact defect report prediction is the closest scenario. For example, performance bug report prediction [75] and configuration bug report prediction [76] - these are important tasks that can influence software quality.

8 Summary

In this paper, we improved the label correctness of five publicly available SBR prediction datasets and conducted an extensive experimental evaluation of the impacts of data label correctness for classification algorithms. The results show: (1) with the same prediction model (e.g., the approaches proposed by Peters et al. and Shu et al.), the performance on the clean datasets is much higher than that of the noisy datasets. (2) Simple text classification performs much better on the clean datasets than the noisy datasets. (3) With the clean datasets, simple text classification outperforms the baseline approaches proposed by Peters et al. and Shu et al.

7. https://github.com/NWPU-IST/sbrbench.

ACKNOWLEDGMENTS

The authors would like to thank Sensen Guo, Zhong Liu, Weiqiang Fu, Fengyu Liu, and Manqing Zhang for their contributions to our datasets annotation. We would also like to thank the authors of *Better Security Bug Report Classification via Hyperparameter Optimization* and *Text Filtering and Ranking for Security Bug Report Prediction* for sharing their datasets and scripts.

This research is done with support from the Innovation Foundation for Doctor Dissertation of Northwestern Polytechnical University (CX202067) and the Key Laboratory of Advanced Perception and Intelligent Control of High-end Equipment, Ministry of Education (GDSC202006).

REFERENCES

[1] "Mining software repository," 2020, http://www.msrconf.org/.
[2] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.
[3] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Software Engineering*, vol. 21, no. 6, pp. 2298–2323, 2016.
[4] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 812–823.
[5] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon, and M. Harman, "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 695–705.
[6] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *In Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE, 2015, pp. 789–800.
[7] J. Debattista, S. Auer, and C. Lange, "Luzzu—a methodology and framework for linked data quality assessment," *Journal of Data and Information Quality (JDIQ)*, vol. 8, no. 1, pp. 1–32, 2016.
[8] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 481–490.
[9] O. Alonso, "Challenges with label quality for supervised learning," *Journal of Data and Information Quality (JDIQ)*, vol. 6, no. 1, pp. 1–3, 2015.
[10] Z. Liu, X. Xia, C. Treude, D. Lo, and S. Li, "Automatic generation of pull request descriptions," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 176–188.
[11] F. Peters, T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.
[12] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better Security Bug Report Classification via Hyperparameter Optimization," in *preprint arXiv:1905.06872*, 2019.
[13] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Proceedings of the 7th International Working Conference on Mining Software Repositories, Cape Town, South Africa, 2-3 May 2010*. IEEE, 2010, pp. 11–20.
[14] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *Proceedings of International Conference on Optimization, Reliabilty, and Information Technology, Faridabad, India, 6-8 Feb. 2014*. IEEE, 2014.
[15] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *Proceedings of the 8th working conference on mining software repositories*, 2011, pp. 93–102.
[16] P. Kamongi and K. Kavi, "VULCAN: Vulnerability Assessment Framework for Cloud Computing," in *Proceedings of the 7th International Conference on Software Security and Reliability, Gaithersburg, MD, USA, 18-20 June 2013*. IEEE, 2013.
[17] Z. Thomas, N. Nachiappan, and W. Laurie A., "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation, Paris, France, April 7-9, 2010*. IEEE, 2010, pp. 421–428.
[18] "Mining Software Repository Challenge," 2011, http://2011.msrconf.org/msr-challenge.html.
[19] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, "A dataset of high impact bugs: Manually-classified issue reports," in *Mining Software Repositories*, 2015.
[20] Q. Gao, Y. Xiong, Y. Mi, L. Zhang, W. Yang, Z. Zhou, B. Xie, and H. Mei, "Safe memory-leak fixing for c programs," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 459–470.
[21] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 65–76.
[22] "2019 TOP 25 CWEs," 2019, https://cwe.mitre.org/top25/.
[23] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 49–60.
[24] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.
[25] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 86–96.
[26] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 396–407.
[27] X. L. Yang, D. Lo, X. Xia, Q. Huang, and J.-L. Sun, "High-impact bug report identification with imbalanced learning strategies," *Journal of Computer Science and Technology*, vol. 32, no. 1, pp. 181–198, 2017.
[28] W. J. Wijayasekara D, Manic M, "Mining Bug Databases for Unidentified Software Vulnerabilities," in *Proceedings of the 5th International Conference on Human System Interactions, erth, WA, Australia, 6-8 June 2012*, 2012, pp. 89–96.
[29] J. Arnold, A. Tim, D. Waseem, P. Gregory, E. Nelson, T. Geoffrey, and A. Kaseorg, "Security Impact Ratings Considered Harmful," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*. Usenix, 2009.
[30] S. Kim, T. Zimmermann, K. Pan, E. James Jr *et al.*, "Automatic identification of bug-introducing changes," in *21st IEEE/ACM international conference on automated software engineering (ASE'06)*. IEEE, 2006, pp. 81–90.
[31] Y. Fan, X. Xia, D. A. da Costa, D. Lo, A. E. Hassan, and S. Li, "The impact of changes mislabeled by szz on just-in-time defect prediction," *IEEE Transactions on Software Engineering*, 2019.
[32] P. S. Kochhar, F. Thung, and D. Lo, "Automatic fine-grained issue report reclassification," in *2014 19th International Conference on Engineering of Complex Computer Systems*. IEEE, 2014, pp. 126–135.
[33] S. J. Herzig Kim and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401.
[34] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou, "Automated configuration bug report prediction using text mining," in *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 2014, pp. 107–116.
[35] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM sigsoft software engineering notes*, vol. 30, no. 4, pp. 1–5, 2005.
[36] O. Jalali, T. Menzies, and M. Feather, "Optimizing requirements decisions with keys," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, 2008, pp. 79–86.
[37] Graham and Paul, *Hackers and painters : big ideas from the computer age*. O'Reilly, 2004.

[38] M. Feurer and F. Hutter, "Hyperparameter optimization," *Automated Machine Learning*, pp. 3–33, 2019.

[39] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[40] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, 2018.

[41] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.

[42] B. Liu, G. Meng, W. Zou, Q. Gong, F. Li, M. Lin, D. Sun, W. Huo, and C. Zhang, "A large-scale empirical study on vulnerability distribution within projects and the lessons learned," in *2020 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2011.

[43] K. Ku, T. E. Hart, M. Chechik, and D. Lie, "A buffer overflow benchmark for software model checkers," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 389–392.

[44] I. Pashchenko, S. Dashevskyi, and F. Massacci, "Delta-bench: differential benchmark for static analysis security testing tools," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 163–168.

[45] "CVE," 2020, https://www.cvedetails.com/.

[46] "JIRA," 2018, https://www.atlassian.com/software/jira.

[47] B. Potter and G. McGraw, "Software security testing," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 81–85, 2004.

[48] "CWE," 2020, https://cwe.mitre.org/.

[49] C. E. Landwehr, B. Alan R., M. John P., and C. William S., "A taxonomy of computer program security," *ACM Computing Surveys*, vol. 26, no. 3, pp. 211–254, 1994.

[50] "Software CWEs," 2020, https://cwe.mitre.org/data/definitions/699.html

[51] "CWE 1228," 2020, https://cwe.mitre.org/data/definitions/1228.html/.

[52] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, and G. C. Murphy, "Locating latent design information in developer discussions: A study on pull requests," *IEEE Transactions on Software Engineering*, 2019.

[53] D. Spencer, "Card sorting: Designing usable categories," *Rosenfeld Media*, 2009.

[54] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[55] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological Bulletin*, vol. 76, no. 5, pp. 378–382, 1971.

[56] B. Jack, "Vector rewrite attack: Exploitable null pointer vulnerabilities on arm and xscale architectures," *White paper, Juniper Networks*, 2007.

[57] T. Mandt, "Kernel attacks through user-mode callbacks," *BlackHat USA, Aug*, 2011.

[58] A. Kogtenkov, "Void safety," Ph.D. dissertation, ETH Zurich, 2017.

[59] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2010.

[60] "Scikit learn," 2020, https://scikit-learn.org/stable/index.html.

[61] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network-based detection of self-admitted technical debt: From performance to explainability," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 3, pp. 1–45, 2019.

[62] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, pp. 125–136, 2019.

[63] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[64] Y. Yang, X. Xia, D. Lo, T. Bi, J. Grundy, and X. Yang, "Predictive models in software engineering: Challenges and opportunities," in *preprint arXiv:2008.03656*, 2020.

[65] P. Hulot, D. Aloise, and S. D. Jena, "Towards station-level demand prediction for effective rebalancing in bike-sharing systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 378–386.

[66] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models," *IEEE Transactions on Software Engineering*, 2018.

[67] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, "Automatic, high accuracy prediction of reopened bugs," *Automated Software Engineering*, vol. 22, no. 1, pp. 75–109, 2015.

[68] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "Data scientists in software teams: State of the art and challenges," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1024–1038, 2017.

[69] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480.

[70] J. Svajlenko and C. K. Roy, "Evaluating clone detection tools with bigclonebench," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 131–140.

[71] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1157–1168.

[72] J. Svajlenko and C. K. Roy, "Bigcloneeval: A clone detection tool evaluation framework with bigclonebench," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 596–600.

[73] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 70–80.

[74] Y. Fan, X. Xin, D. Lo, and A. E. Hassan, "Chaff from the wheat: Characterizing and determining valid bug reports," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2018.

[75] X. Han, T. Yu, and D. Lo, "Perflearner: learning from bug reports to understand and generate performance test frames," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 17–28.

[76] W. Wen, T. Yu, and J. H. Hayes, "Colua: Automatically predicting configuration bug reports and extracting configuration options," in *2016 IEEE 27Th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 150–161.