2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and
2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)

Security and Performance Bug Reports Identification with Class-Imbalance Sampling and Feature Selection

Dipok Chandra Das
Institute of Information Technology
University of Dhaka, Bangladesh
bsse0501@iit.du.ac.bd

Md. Rayhanur Rahman
Institute of Information Technology
University of Dhaka, Bangladesh
rayhan@du.ac.bd

***Abstract*—Nowadays, software projects receive a huge number of bug reports daily. Among them, security and performance bug reports are higher priority to software developers and users. So, rapid identification of security and performance bug reports as soon as these are reported is mandatory. But bug tracking systems do not provide any mechanism to isolate them from the collection of bug reports. In this paper, we have proposed a learning based approach to identify security and performance bug reports addressing class-bias and feature-skew phenomenon. We have proposed two separate classification models namely *Sec-Model* and *Perf-Model*, where the former classifies a bug report as security or non-security bug report and the latter classifies as performance or non-performance bug report. We have experimented our approach on four datasets of bug reports of four software projects- Ambari, Camel, Derby and Wicket. We have evaluated the performance of our two models in terms of area under curve receiver operating characteristics curve (AUC). The average AUC values of *Sec-Model* and *Perf-Model* are 0.67 and 0.71 respectively.**

***Keywords*—Bug Report Classification, Security Bug, Performance Bug, Class-Imbalance Learning, Feature Selection**

I. Introduction

Software projects often manage bug tracking systems to track the bugs in their software. These bugs are reported by testers, users or developers from all over the world. A triager inspects the reported bug report and take actions i.e. assigning priority, developer for this bug report. Among these bug reports, the triager has to isolate high impact security and performance bug reports as soon as these are reported. Since the usage of Internet devices are increasing, the security issues are of great concern among people [1]. A security bug can cause unauthorized access to the software. Performance bug causes performance degradation i.e. low user experience, waste of computational resources, lower throughput, less responsiveness. A performance bug can force the users to switch to competitive providers [2]. So, delay in identifying or unnoticed security or performance bug reports can cause serious loss to systems as well as users. But in a large-scale software system, a huge amount of bugs is reported daily [3]. So, it becomes a tiresome and time-consuming task for a triager to inspect each bug report and to isolate security and performance bug reports. Manual inspection process can also introduce human-error.

The bug tracking tools do not provide any mechanism to isolate security and performance bug reports from huge number of bug reports [4]. The triager use *grep* based searching to identify security and performance bug reports. *grep* takes the advantage of string matching property. For security bug, the triager search with security-vulnerability terms e.g. “authorization”, “login” ,“attack”, etc., and for performance bug reports, search with performance-related terms e.g. “perf”, “performance”, “memory”, etc. But the *grep* based approach does not perform well in real scenario. Because both the summary and description of a bug report written in free-form natural language text are incomplete and imprecise [5]. It also requires domain knowledge about software security and performance. So, to overcome time-consumption, human-error and low-performance of *grep*, we propose a *learning* based approach along with text-mining in order to identify security and performance bug reports. The *learning* based approach builds a classifier model (i.e. statistical model) using text-mining techniques from historically labeled bug reports to identify security and performance bug reports. But in the *learning* approach, there are two problems due to imbalanced data namely class-bias [6] and feature-skew [3]. Class-bias problem refers the bias to the majority sampled class in a classification model due to imbalanced data in a binary setting. In our case, the number of security bug reports and performance bug reports are smaller than other types of bug reports. So, the learning model gets biased to other kinds of bug reports. Feature-skew problem refers to positive features outnumbering negative features in selected features by a two-sided feature selection metric e.g. CHI-Square, Mutual Information [7], or its derivatives [8], etc. due to imbalanced data [3]. So, the negative features are absent from the selected features resulting in misclassifying samples of negative class. But correct classification of samples of negative class is also important as negative samples dominate in number.

In order to address the class-bias problem, we apply *class-imbalance sampling* which balances the samples of considered classes to some extent before training the model. There are popular sampling techniques namely under-sampling, over-sampling and SMOTE [6]. We apply Random Under-sampling (RUS) in our study, a variant of under-sampling because of its effectiveness [10]. In order to address feature-skew problem, we apply a feature selection proposed by Zheng *et al.* [3] which selects positive and negative features separately and explicitly combines them as selected features.

In this paper, we propose two separate *binary single*

978-1-5386-5163-6/18/$31.00 ©2018 IEEE

label classification models namely *Sec-Model* and *Perf-Model*, where the former has two labels, security (i.e. positive) and non-security (i.e. negative) bug report and the latter has two labels, performance (i.e. positive) and non-performance (i.e. negative) bug report. We propose two separate classification models instead of one single classification model for two reasons. Firstly, it is due to multi-class imbalanced classification. In multi-class imbalanced classification, the relations among classes are not straightforward. One class is minor compared to second class but is major to third class. There exist data-level difficulties like overlapping and class noise and ill-defined class boundaries [6]. So, the overall performance of a single multi-class imbalance classification model degrades. Secondly, correct identification of security and performance bug reports is essential because of their high impact nature. Each category (i.e. security or performance) will get more focus in separate classification models than one single classification model. Separate classification models will take more time and resources but will help to identify the security and performance bug reports accurately. Our proposed approach is equivalent to *one-versus-all* approach in multi-class imbalanced classification [9]. We have used Naïve Bayes Multinomial (NBM) as a machine learning algorithm in the classification models because of its effectiveness [8], [10]. We have experimented our approach on four datasets of bug reports created by Ohira *et al.* [1]. We use popular technique Receiver Operating Characteristic (ROC) to evaluate the effectiveness of our approach which is popular in fraud detection, medical science, etc [11]. We compare the performance of each classification model in terms of area under ROC curve (AUC).

The reminder of this paper is organized as follows. Section II presents the related work. Section III describes the proposed approach. Section IV illustrates the experiments and results. Section V contains threats to validity. Section VI contains conclusion and future work.

II. Related Work

We classify the related works into three parts. The first part contains studies about security and performance bug reports. The second part is about bug report categorization and the third part is about studies on class-imbalance and feature selection techniques applied in the field of bug report.

A. Security and Performance Bug Reports

The recent work close to ours is conducted by Zhou *et al.* [12]. They propose machine learning approach in order to find security vulnerabilities from bug reports in open source projects. They apply the ensemble method, Stacking [13] in order to resolve imbalanced data problem. First, our study focuses on security and performance bug report. Second, our study is more interested on feature selection and class-imbalance sampling. Because feature selection and class-imbalance sampling are relatively more important than machine learning algorithm in imbalanced situations [14]. Gegick *et al.* [15] propose a text mining approach to identify security bugs. They have investigated their proposed approach on Cisco Software Systems[1] to identify security bug reports. They have found that the security bugs are mis-labeled by security engineers too. They do not consider the imbalanced data phenomenon.

Zaman *et al.* [16] conduct a case study on security and performance bug reports of Mozilla Firefox[2]. They have found that the security and performance bug reports require more time, developers to fix than other types of bug reports. Next year, Zaman *et al.* [4] conduct a qualitative study on performance bug reports. They have collected 400 performance and non-performance bug reports from Mozilla Firefox[2] and Google Chrome[3]. They have found that performance bug reports contain performance specific terms (cpu, memory, disk), improvement suggestion, test cases, etc.

Kashiwa *et al.* conduct a pilot study on the diversity of high impact bug reports including security and performance bug reports [2]. Ohira *et al.* [1] create four datasets of high impact bugs of four well-known software projects each containing one thousand bug reports. They have found that the characteristics of security and performance bug reports are quite different from other types of bug in terms of number, fixing time and information source provided by reporters and required by developers to identify and fix them.

Those above mentioned studies confirm that security and performance bug reports are different from other types of bug reports. We are interested in identifying security and performance bug reports more accurately from the collection of huge bug reports. So, we rather build classification model to identify newly arrived security and performance bug reports.

B. Bug Report Categorization

The popular bug report categorization works are bug triage [5], duplicate bug report detection [17], bug and non-bug report identification [18], priority prediction [19], severity prediction [20], etc. These are also known as text categorization. In these text categorization, the number of samples of each category is roughly similar. But in security and performance bug reports, imbalanced data exists. Considering severity and priority, Kashiwa *et al.* introduce six types of high impact bug reports. We focus on two high impact product bugs, security and performance bug [2].

C. Class-imbalance and Feature Selection on Bug Reports

Yang *et al.* [10] introduce class-imbalance sampling in order to identify surprise bug reports. They use datasets created by Ohira *et al.* [1] to measure the effectiveness of the class-imbalance sampling. While surprise bugs are process bugs, security and performance bugs are product bugs. Menzies *et al.* [20] perform feature selection techniques *InfoGain* for predicting the severity of an upcoming bug report. Sharmin *et al.* [8] select most indicative features applying combination of *Mutual Information* and *Chi-Square* for bug report severity prediction. Sharma *et al.* [21] apply *InfoGain* and *Chi-Square* in order to select top-125 features i.e. terms for severity prediction on bug reports. But these conventional feature selection metrics i.e. Mutual Information, InfoGain, Chi-Square, etc. do not perform well in imbalanced data scenario [3].

[1] https://www.cisco.com/

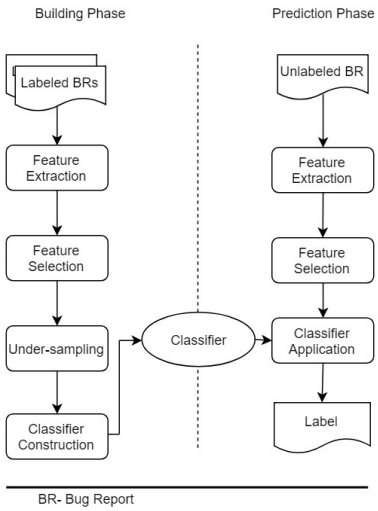

Fig. 1: Overall Framework of Generalized Classification Model

III. Proposed Approach

Our approach builds a classification model (i.e. statistical model) from historically labeled bug reports to identify security and performance bug reports. We build two separate binary single-label models namely *Sec-Model* and *Perf-Model*. The generalized model is shown in Figure 1 and described in this section. Firstly, we extract features from bug reports. Secondly, we perform feature selection on extracted features and class-imbalance sampling on labeled data. Thirdly, we apply Naive Bayes Multinomial (NBM) as a machine learning algorithm in order to build our classification model. Finally, we evaluate our prediction model on some real data.

A. Feature Extraction

We consider the textual information (i.e. *summary* and *description*) of bug reports as source of our feature extraction of our classification model. We apply text mining approach to extract features from the textual information. First, we use *tokenization* to extract terms in the textual information and also to filter out the punctuation, numbers, etc. Applying Stop-words removal, we filter out conjunctions, prepositions and terms e.g. "the", "in", etc., as those terms do not carry much specific information. Then, we use Porter Stemmer [22] to convert each term to its basic form as same term expressed in different forms carry same information. The extracted terms are the used as extracted features. We represent a bug report using the feature vector as $BR = \langle f_1, f_2, f_3, f_4, ..., f_n \rangle$ where n is the total number of features i.e. terms and the value of f_i is the *tf-idf* value of the feature in the bug report.

B. Feature Selection

Bug reports classification is considered as text categorization problem. The dimension of feature space on text categorization is high. Because of high-dimensionality, the performance of classification model degrades [7]. Feature selection method selects the most indicative features filtering out noisy and redundant features. Feature selection also reduces training and testing time as well as the chance of over-fitting. The features can be positive and negative referring to indicative of membership and non-membership of a specific category respectively. One-sided feature selection metric e.g. Log ODD Ratio, Coefficient Correlation [3], etc. selects either positive or negative but not both features. Positive features selected by one-sided metric can predict the positive samples rightly but fail to correctly classify the negative samples [3]. But the correct classification of the negative samples is also important as in our context, the negative samples outnumbers positive samples significantly. Two-sided feature selection metric e.g. Chi-Square, Mutual Information, [7] etc. selects features combining positive and negative features ignoring the sign of the feature. Features selected by two-sided metric on imbalanced data ignore the negative features as their score is not significant due to huge number of negative samples. So, the samples of negative class are misclassified. Forman *et al.* argue that feature selection is more important than machine learning algorithm in imbalanced data [14]. So, correct combination of positive and negative features is needed.

Zheng *et al.* [3] propose a feature selection technique for text categorization for imbalanced data which selects positive and negative features using one-sided metric separately and explicitly combines the positive and negative features as selected features. The proposed feature selection outperforms the existing both one-sided feature selection metrics and two-sided feature selection metrics on imbalanced data. The proposed framework has two parameters, α, percentage of positive features of total features and *l*, total number of features. The feature selection framework is described below:

- Choose a one-sided feature selection metric $\xi(t, C)$ and empirically select two parameter α and l
- Select positive feature set, F^+ of $l_1 = \alpha * l$ features with highest $\xi(t, C)$ scores
- Select negative feature set, F^- of $l_2 = l - l_1$ features with lowest $\xi(t, C)$ scores
- Compute the union of positive and negative features set to find final feature set F, $F = F^+ \cup F^-$

In this proposed framework, any one-sided feature selection metric can be used. Likewise Zheng *et al.* [3], we use LOR, Log Odd Ratio, as one-sided feature selection metric. In a binary setting of positive and negative class, odds ratio measures the odds of the term occurring in the positive class normalized by that of the negative class. The basic idea is that the distribution of features on the positive class is different from the distribution of features on the negative class. LOR is computed by (1).

$$LOR(t, C_i) = \log \frac{P(t|C_i)[1 - P(t|\bar{C}_i)]}{P(t|\bar{C}_i)[1 - P(t|C_i)]} \quad (1)$$

In (1), t and C_i refer to term and class respectively.

C. Random Under-Sampling

Due to class-bias, the performance of classification model degrades. Class-imbalance sampling overcomes the bias by balancing the number of samples of considered classes before training. There are three popular sampling techniques namely under-sampling, oversampling and SMOTE [23]. We use under-sampling as this technique performs better in most

[2]https://bugzilla.mozilla.org/
[3]https://www.chromium.org

cases [10]. Under-sampling continuously deletes the samples from majority class until a predefined ratio of the number of samples belonging to majority class to the number of samples of all classes is reached. Under-sampling repeats the following two steps until a predefined ratio of majority samples to all samples reaches to r:

Step1: Sample Selection: Select a sample from the original data belonging to majority class. The sample is selected randomly or applying KNN[4]-like approach.

Step2: Sample Deletion: The selected sample is deleted from training data.

In our experiment, we select samples belonging to majority class randomly, namely Random Under-sampling (RUS). We use Random Under-sampling (RUS) because of its effectiveness [10]. We set r as 0.5 stating that the number of samples of positive and negative class is equal in the training data.

D. Naïve Bayes Multinomial Model

Naive Bayes (NB) classifier is a popular machine learning algorithm in text categorization. Naive Bayes assumes that the features are independently and identically (i.i.d) distributed. Also, all the features are binomial. It states that each feature has value either 0 or 1; a feature is present or absent in an instance. Despite unrealistic assumption, the classifier performs surprisingly well. Naive Bayes Multinomial (NBM) is a derivation of Naive Bayes. The value of each feature in NBM is any non-negative number. NBM outperforms NB because NBM carries more information than NB [8] [10]. A bug report is represented as a vector $BR_i = \langle f_{i1}, f_{i2}, f_{i3}, ..., f_{in} \rangle$ where n is the size of the feature set and each $f_{ij} \in [0, \infty)$ indicates the *tf-idf* value of each term in bug report BR_i. The conditional probability of class, c_j given bug report, BR_i is computed using Bayesian rule (2).

$$p(c_j|BR_i) = \frac{p(c_j) \times p(BR_i|c_j)}{p(BR_i)} \tag{2}$$

The probability $p(BR_i|c_j)$ is calculated under Naive Bayes' assumption of independence of features by (3).

$$p(BR_i|c_j) = \prod_{k=1}^{n} p(t_k|c_j)^{w(t_k, BR_i)} \tag{3}$$

In (3), $w(t_k, BR_i)$ is *tf-idf* value of term t_k in BR_i. $p(t_k|c_j)$ is estimated from training data with known classes, using maximum likelihood Laplacean prior:

$$p(t_k|c_j) = \frac{1 + \sum_{BR_i \in c_j} w(t_k, BR_i)}{n + \sum_{k=1}^{n} \sum_{BR_i \in c_j} w(t_k, BR_i)} \tag{4}$$

As prior, $p(c_j)$ is uniform among classes and $p(BR_i)$ is equal to 1 for all classes, the Multinomial Naive Bayes Classifier predicts the label $\hat{c}$ of a Bug Report using the (5).

$$\hat{c} = \arg\max_j \prod_{k=1}^{n} p(t_k|c_j)^{w(t_k, BR_i)} \tag{5}$$

[4]K-nearest neighbour

TABLE I: Ohira Dataset [1]

Project	Total BR	Sec BR	Perf BR	IR in *Sec-Model*	IR in *Perf-Model*
Ambari	1000	29	41	33.48	23.39
Camel	1000	32	93	30.25	9.75
Derby	1000	88	101	10.36	8.90
Wicket	1000	10	83	99	11.05

BR-Bug Report, Sec-Security, Perf-Performance, IR-Imbalance Ratio

IV. Experiment and Results

In our study, the experimental environment is Intel(R) Core(TM) i3450 3.10 GHz CPU, 8GB RAM desktop running Windows 10 (64-bit). In our evaluations, we intend to answer the two research questions:

- RQ1-How the proposed approach affect the performance of the classifier?
- RQ2-How the classifier performs with respect to training with description of bug report?

A. Data Collection

We have collected four datasets of bug reports (i.e. issue reports) of four software projects namely Ambari[5], Camel[6], Derby[7] and Wicket[8]. Ohira *et al.* [1] have created these datasets and manually labeled these four thousand bug reports in order to study the characteristics of high impact bugs including security and performance bug reports. The details of four datasets are shown in Table I. Each dataset contains one thousand bug reports referring to one of four projects. The proportion of security and performance bug reports is smaller across all four datasets (i.e. projects) resulting in imbalanced data. The ratio of number of negative samples to positive samples is called imbalance ratio, IR. In both *Sec-Model* and *Perf-Model*, the negative samples outnumbers positive samples significantly across all projects. *Sec-Model* experiences highly imbalanced data compared to *Perf-Model*. The average IRs in *Sec-Model* (i.e., non-security:security) and *Perf-Model*(i.e., non-performance:performance) is 43.27 and 13.27 respectively. The IR of project Wicket is 99, resulting in highest imbalanced data.

B. Experimental Settings

In order to answer the research question, **RQ1**, we build each classification model in three different approaches namely ***Proposed-SFS***, ***RUS*** and ***FS***. In ***RUS***, we do not address the *feature-skew* problem, and it similar to the approach proposed by Yang *et al.* [10] for surprise bug reports identification and in ***FS***, we do not address the *class-bias* problem, and it is similar to approach proposed by Zheng *et al.* [3] for text categorization on imbalanced data. In order to answer the research question, **RQ2**, we conduct feature extraction (see Section III-A) in two ways. One way, we extract features only from the *summary* of bug report excluding *description* is named ***without-desc*** and other way, we extract features from both *summary* and *description* is named ***with-desc*** (see Table IIa and Table IIb). We use popular ten-fold cross-validation

[5]https://issues.apache.org/jira/projects/AMBARI/
[6]https://issues.apache.org/jira/projects/CAMEL/
[7]https://issues.apache.org/jira/projects/DERBY/
[8]https://issues.apache.org/jira/projects/WICKET/

TABLE II: Experimental Results of *Sec-Model* and *Perf-Model*

(a) AUC values of *Sec-Model*

FEx	Method	Ambari	Camel	Derby	Wicket	AVG
without-desc	*RUS*	0.6460	0.6223	0.7309	0.5141	0.6283
without-desc	*FS*	0.5391	0.6206	0.7598	0.5260	0.6114
without-desc	*Proposed-SFS*	0.7071	0.6200	0.8065	0.5414	**0.6688**
with-desc	*RUS*	0.6260	0.5859	0.7238	0.5535	0.6223
with-desc	*FS*	0.5986	0.6646	0.7527	0.5189	0.6337
with-desc	*Proposed-SFS*	0.5933	0.6725	0.8291	0.5331	**0.6570**

(b) AUC values of *Perf-Model*

FEx	Method	Ambari	Camel	Derby	Wicket	AVG
without-desc	*RUS*	0.7071	0.6430	0.6884	0.5759	0.6536
without-desc	*FS*	0.5980	0.6525	0.6843	0.6532	0.6469
without-desc	*Proposed-SFS*	0.6275	0.6590	0.7686	0.6729	**0.6820**
with-desc	*RUS*	0.6645	0.6477	0.6846	0.6532	0.6625
with-desc	*FS*	0.6391	0.6379	0.7117	0.6912	0.6700
with-desc	*Proposed-SFS*	0.7067	0.6540	0.7322	0.7401	**0.7083**

FEx:Feature Extraction, *with-desc*: *summary+description*, *without-desc*: *summary*, AVG: Average AUC value

TABLE III: Confusion Matrix

	p	n
Y	TP	FP
N	FN	TN

technique for evaluating the effectiveness of our approach. In ten-fold cross-validation, we split the training data into ten folds. We perform ten evaluation round; in each round nine folds are used for training and rest one fold is used for testing. We sum up the ten round's result to report the overall performance. We conduct the whole experiment ten times and report the average results. We use popular Python (v-3.5) libraries, *nltk*[9] for text mining, *sklearn*[10] for machine learning and *imbalanced-learn*[11] for Random Under-sampling. We use the default values of parameters of aforementioned Python libraries. The parameters in feature selection approach, l and α are set to $\{0.15, 0.25, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0\}$ and $\{0.5\}$ respectively, where values of l represent the percentage of total features.

C. Evaluation Metrics

Popular metrics measure the performance of classification models in terms of true positives, false positives, false negatives and true negatives. Let in a binary classification model, there are two kinds of samples namely positives, p and negatives, n. The prediction of an instance has two values, Y and N denoting positive and negative class. Then, *TP*, *FP*, *FN* and *TN* denote true positives, false positives, false negatives and true negatives respectively in Table III. Popular measures, F-measure and Accuracy are not appropriate in evaluating the performance in our experiment because class distribution is not uniform. We use performance measurement approach named Receiver operating characteristic (ROC). ROC graphs are invariant with class and error cost distribution [11]. ROC uses two metrics True Positive Rate, TPR and False Positive Rate, FPR. TPR is defined by the true positives divided by actual positive samples (6). TPR is known as *hit rate*, equivalent to recall.

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

[9]https://www.nltk.org

[10]http://scikit-learn.org/

[11]http://contrib.scikit-learn.org/imbalanced-learn

FPR is defined by the false positive divided by actual negative samples (7). FPR is known as *false alarm rate*.

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

ROC graphs are two-dimensional graphs in which TPR is plotted on the Y axis and FPR is plotted on the X axis. An ROC graph depicts relative trade-offs between benefits (true positives) and costs (false positives). A scalar metric named area under an ROC curve (AUC) is used to compare the performance of different classifiers. AUC calculates the area under the ROC curve constructed by plotting the score of the classifier i.e. (FPR,TPR), in ROC graph. The AUC value has a statistical property that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [11]. The value of AUC ranges from 0.0 to 1.0. The AUC of random classification is 0.5. The performance of a classifier is considered better if it scores higher AUC value.

D. Experimental Results

We illustrate the experimental results of *Sec-Model* and *Perf-Model* in this section. Table IIa and Table IIb show the experiment results of *Sec-Model* and *Perf-Model* respectively on different projects (i.e., datasets) in terms of AUC value. We also calculate the average of AUC scores of four different projects.

1) Answer to Research Question, ***RQ1****:* In *Sec-Model*, from the Table IIa, we can see that our proposed approach, *Proposed-SFS* of *without-desc* consistently outperforms *RUS* and *FS* across four projects in terms of AUC value. The average AUC values of *Proposed-SFS*, *RUS* and *FS* are 0.6688, 0.6283 and 0.6114 respectively. In *with-desc*, the proposed approach fails to outperform consistently. The proposed approach scores 5% lower in Ambari and 3.5% in Wicket but 10% higher in Derby than best of *RUS* and *FS*. In *Perf-Model*, from the Table IIb, in *without-desc*, the proposed approach outperforms *RUS* and *FS* across all projects but Ambari. In *with-desc*, the proposed approach outperforms RUS and FS consistently across all projects. The average AUC values of *Proposed-SFS*, *RUS* and *FS* are 0.7083, 0.670 and 0.6625 respectively. So, our proposed approach of combining feature selection and under-sampling constructs a synergy.

2) Answer to Research Question, ***RQ2****:* In *Sec-Model* (Table IIa), our proposed approach of *without-desc* performs better than of *with-desc* in terms of average AUC, namely 0.6688 in *without-desc* and 0.6570 in *with-desc*. But this

improvement is not impressive. In *Perf-Model* (Table IIb), the proposed approach of *with-desc* outperforms of *without-desc* by 3.8%. In *Sec-Model*, the inclusion of *description* in feature extraction does not increase the performance of the classifier rather degrades the performance. But in *Perf-Model*, the overall performance of the classifier increases with the inclusion of *description* in feature extraction.

V. Threats to Validity

Threats to construct validity concern the suitability of our evaluation metrics. We have used area under of ROC curve (AUC) metric in order to compare the performance of the approaches. ROC graphs are popular in medical science, fraud detection, etc., whose are akin to our case and ROC graphs are invariant with respect to class and cost error distribution [11]. So, there is no threat to construct validity. Threats to internal validity concern human-error in experiments or bias to randomization. We have used default values of parameters in Python's machine learning libraries in order to reduce internal threat. We have also conducted each experiment ten times and reported the average results. So, there is little threat to internal validity. Threats to external validity concern the generalization of our results. We have conducted our experimented on four datasets of bug reports of four software projects created by Ohira *et al.* [1]. The datasets contain bug reports of four well-known software projects. So, usage of standard datasets reduces the threats to external validity to some extent.

VII. Conclusion and Future Work

Security and Performance bug reports are of great concern of software providers and users. Security and Performance bug reports classification is a class-imbalance problem. We have proposed two separate models namely *Sec-Model* and *Perf-Model* to identify security and performance bug reports separately. The generalized approach (see Section III) of combining feature selection and under-sampling outperforms both individual under-sampling and feature selection . We have also found that features extracted from both *description* and *summary* help in identifying security and performance bug reports more accurately than alone *summary*. In the future, we will focus more on feature extraction sub-module. The inclusion of structured information like author, component and keywords, may lead to better performance of the learning based approach in identifying security and performance bug reports.

V. Acknowledgments

This research is supported by the fellowship from ICT Division, Ministry of Posts, Telecommunications and Information Technology, Bangladesh, No-56.00.0000.028.33.094.18-168, dated 03 May, 2018.

References

[1] 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 *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, 2015, pp. 518–521.

[2] Y. Kashiwa, H. Yoshiyuki, Y. Kukita, and M. Ohira, "A pilot study of diversity in high impact bugs," in *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, 2014, pp. 536–540.

[3] Z. Zheng, X. Wu, and R. K. Srihari, "Feature selection for text categorization on imbalanced data," *SIGKDD Explorations*, vol. 6, no. 1, pp. 80–89, 2004.

[4] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings*, 2011, pp. 93–102.

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, 2006, pp. 361–370.

[6] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[7] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Icml*, vol. 97, 1997, pp. 412–420.

[8] S. Sharmin, F. Aktar, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "Bfsp: A feature selection method for bug severity classification," in *Humanitarian Technology Conference (R10-HTC), 2017 IEEE Region 10*. IEEE, 2017, pp. 750–754.

[9] A. FernáNdez, V. LóPez, M. Galar, M. J. Del Jesus, and F. Herrera, "Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches," *Knowledge-based systems*, vol. 42, pp. 97–110, 2013.

[10] X. Yang, D. Lo, Q. Huang, X. Xia, and J. Sun, "Automated identification of high impact bug reports leveraging imbalanced learning strategies," in *40th IEEE Annual Computer Software and Applications Conference, COMPSAC 2016, Atlanta, GA, USA, June 10-14, 2016*, 2016, pp. 227–232.

[11] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[12] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, 2017, pp. 914–919.

[13] Z.-H. Zhou, "Ensemble learning," *Encyclopedia of biometrics*, pp. 411–416, 2015.

[14] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1289–1305, 2003.

[15] 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, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, 2010, pp. 11–20.

[16] 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*. ACM, 2011, pp. 93–102.

[17] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, 2008, pp. 461–470.

[18] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada*, 2008, p. 23.

[19] Y. Tian, D. Lo, and C. Sun, "DRONE: predicting priority of reported bugs by multi-factor analysis," in *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*, 2013, pp. 200–209.

[20] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China*, 2008, pp. 346–355.

[21] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Computer Science*, vol. 70, pp. 632–639, 2015.

[22] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[23] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.