Research Article

Maryyam Said, Rizwan Bin Faiz, Mohammad Aljaidi*, and Muteb Alshammari

Comparative analysis of impact of classification algorithms on security and performance bug reports

https://doi.org/10.1515/jisys-2024-0045
received January 12, 2024; accepted June 30, 2024

Abstract: Identification and classification of bugs, e.g., security and performance are a preemptive and fundamental practice which contributes to the development of secure and efficient software. Software Quality Assurance (SQA) needs to classify bugs into relevant categories, e.g., security and performance bugs since one type of bug may have a higher preference over another, thus facilitating software evolution and maintenance. In addition to classification, it would be ideal for the SQA manager to prioritize security and performance bugs based on the level of perseverance, severity, or impact to assign relevant developers whose expertise is aligned with the identification of such bugs, thus facilitating triaging. The aim of this research is to compare and analyze the prediction accuracy of machine learning algorithms, i.e., Artificial neural network (ANN), Support vector machine (SVM), Naïve Bayes (NB), Decision tree (DT), Logistic regression (LR), and K-nearest neighbor (KNN) to identify security and performance bugs from the bug repository. We first label the existing dataset from the Bugzilla repository with the help of a software security expert to train the algorithms. Our research type is explanatory, and our research method is controlled experimentation, in which the independent variable is prediction accuracy and the dependent variables are ANN, SVM, NB, DT, LR, and KNN. First, we applied preprocessing, Term Frequency-Inverse Document Frequency feature extraction methods, and then applied classification algorithms. The results were measured through accuracy, precision, recall, and *F*-measure and then the results were compared and validated through the ten-fold cross-validation technique. Comparative analysis reveals that two algorithms (SVM and LR) perform better in terms of precision (0.99) for performance bugs and three algorithms (SVM, ANN, and LR) perform better in terms of *F*1 score for security bugs as compared to other classification algorithms which are essentially due to the linear dataset and extensive number of features in the dataset.

Keywords: bug classification, security bug, performance bug, text mining, bug prediction

1 Introduction

An error, defect, or fault in software is known as a bug. A bug is identified in the software after the testing phase. The developer allows users to report bug by bug tracking systems such as Bugzilla, Eclipse, etc. [1]. Software bug management involves the identification, classification, prediction, storage, prevention, and

*** Corresponding author: Mohammad Aljaidi,** Department of Computer Science, Faculty of Information Technology, Zarqa University, Zarqa, 13116, Jordan, e-mail: mjaidi@zu.edu.jo
Maryyam Said: Faculty of Computing Riphah International University, Islamabad, 46000, Pakistan, e-mail: maryamsaid880@yahoo.com, maryamsaid880@gmail.com
Rizwan Bin Faiz: Faculty of Computing Riphah International University, Islamabad, 46000, Pakistan, e-mail: rizwan.faiz@riphah.edu.pk
Muteb Alshammari: Department of Information Technology, Faculty of Computing and Information Technology Northern Border University, Rafha, 91431, Saudi Arabia, e-mail: muteb.alshammari@nbu.edu.sa

reporting of bugs, these tasks are used to find and remove the bugs [2]. Software bug classification is the task of categorizing the bug into various predefined categories. A software bug has several attributes like bug summary, bug description, bug-id, bug type, assigned-to, product, component, etc. [3]. When we focus on quality attributes and bugs at the same time, we categorize software bugs as security bugs, performance bugs, reliability bugs, usability bugs, etc. [4]. This research focuses on the classification of security and performance bugs that have a significant impact on the cost of system development [5].

The most crucial aspect of software quality is its security. The goal of security is to keep the system safe from things like weak authentication, virus injection, buffer overflows, XSS vulnerabilities, unauthorized access, encryption mistakes, and information loss. Performance defects are found during performance testing and are related to the speed, stability, response time, and resource consumption of the software. When a program malfunctions and is unable to use its allotted resources for specific tasks, it is considered a performance bug [6]. Usability flaws make an application difficult to use, which reduces the user's enjoyment of the program. Test engineers and UX designers compare software to usability requirements in order to find usability flaws. Bug classification improves the overall efficiency of the testing and development processes, expedites the fixing of defects, and has an impact on the assessment of the effectiveness of the development process based on defect severity and priority levels.

Text mining is the task of mining meaningful information from a large volume of text [7]. Text mining includes preprocessing, feature extraction, feature selection, and classification steps [8,9]. Preprocessing includes the following steps: Data cleaning, stop words removal, tokenization, stemming, lemmatization, spelling correction, etc. [10,11]. Preprocessed data are given to machine learning (ML) algorithms as input that uses statistical analysis to predict an output. We have various types of bug tracking tools available in software testing that help to track the bug, which is related to the software. Bugzilla is an open-source bug-tracking tool, which is most widely used by many software organizations to track bugs. Bugzilla supports various operating systems such as Windows, Linux, and Mac. ML is broadly categorized into supervised learning and unsupervised learning. Supervised learning is performed on labelled data. Unsupervised learning performs on unlabeled data [12,13].

Developers enable users to report bugs through bug tracking programs like Eclipse, Bugzilla, and so forth. When a software system is being maintained or developed, a triager frequently needs to speak with a bug report type. The objective is to identify the areas of a project that require additional attention because they are not functioning well. When a new bug is reported, a triager (such as the manager) tries to categorize it so that expertise can be used to fix or resolve it. When there are not many bug reports, the task is small. But as time goes on, more bug reports are filed, and triagers find it increasingly difficult to sift through such a large volume of reports [14].

1.1 Research contribution

The purpose of our experiment is the comparative analysis of supervised classification algorithms on security and performance-related bugs. Improving the bug report prediction accuracy can be done by removing the noise, selecting the important features, adding stop word and start word lists, and using Term Frequency-Inverse Document Frequency (TF-IDF) and *N*-gram. In this research, we compare and analyze the prediction accuracy of ML algorithms while predicting security and performance bugs through text mining/Natural Language Processing (NLP). Our contribution includes:

1. Creation of a new dataset of 1,000 instances from the Bugzilla repository for identification and then labelling security and performance keywords.
2. Preprocess labelled data by:
   a. Removing start and stop words through TD-IDF and *N*-gram.
   b. Standardizing the distribution gap between the source and target dataset.
   c. Class balancing through CTGAN.
3. Extraction of significant features and identification of security and performance bugs through ML algorithms by application of Support vector machine (SVM), Artificial neural network (ANN), Logistic regression (LR), K-nearest neighbor (KNN), and Naïve Bayes (NB).

4. Compare and analyze classification accuracy of performance and security bugs through various classification algorithms using standard metrics, i.e., accuracy, precision, recall, and *F*1.

This study is divided into four sections. Section 1 describes the Introduction, Section 2 gives the background knowledge, Section 3 presents the related work of bug report classification, and Section 4 introduces the proposed approach. Section 5 explains the experiments and results of the proposed approach. The conclusion and future work are discussed in Section 6.

1.2 Background

The goal of our work is to classify security and performance bug(s). We therefore select supervised classification algorithms, e.g., SVM, NB, ANN, Radial basis function (RBF), KNN, and LR. Each algorithm choice is justified based upon its suitability for the dataset.

SVM is a supervised algorithm used for both classification and regression purposes. The main principle of SVM is to determine separators in the search space that can separate the two classes. The SVM finds hyperplane using support vectors and margins [9,15]. SVM kernels are used for different classification tasks, some well-known kernel functions are: linear, polynomial, radial basis, and sigmoid. The SVM classifier is useful in large-scale scenarios in which a large amount of unlabeled data and a small amount of labelled data are available. It enhances classification performance by eliminating unrelated features [16]. SVM is well suited to deal with learning tasks where the number of features is large with the number of training instances. It takes less computational time. The SVM method is widely used in many applications such as pattern recognition, face detection, spam filtering, and text categorization [17,18].

Naïve Bayes is a probability-based classifier that applies the Bayesian theorem [19]. In this algorithm, all attributes are equally important and independent of each other. It assumes that the probability of an event is based on previous knowledge and the history of events [3]. When the number of features is high, the probability calculation is difficult, so feature filtering is an effective way. There are two classes C = S (security bug report, SBR) and C = B (performance bug report) in the bug report analysis [20].

$$\mathrm{P(C|B)} = \frac{(\mathrm{P(B|C)P(C)})}{\mathrm{P(B)} \propto \mathrm{P(B|C)P(C)}}.$$

Naïve Bayes can either have the Bernoulli Model in which a document is represented by a vector of binary features and the frequency of words is ignored. If the vocabulary size is low, Bernoulli performs outclass or multinomial model to capture the word frequencies in a document by representing the document as a bag of words. The multinomial model is always good for large vocabulary sizes [18,21]. The precision rate decreases if the amount of dataset is less. Naïve Bayes is mostly used for text classification applications [9].

A neural network is made up of connected artificial neurons which work like a human brain. The output of each neuron is determined by using an activation function such as sigmoid, logistic, and Tan. In supervised learning, neural networks (NN) are trained with a pattern of known classes [15]. The most frequently used feed-forward neural networks of classification are multilayer perception and RBF [3]. In multi-layer perception (MLP), neurons are placed in different layers that are connected through certain weights. MLP contains input, hidden, and output layers [18]. ANN performs more efficiently when the number of input data is large [17]. ANN is mostly used in text classification and pattern recognition.

KNN is introduced by the nearest neighbor algorithm which is designed to find the nearest point of the observed object. The main idea of KNN algorithms is to find the *K*-nearest points [22]. KNN is a lazy ML algorithm. It works well with a small number of input variables and does not perform well on imbalanced data. KNN inherently has no capability of dealing with a missing value. The performance of the algorithm depends on the number of input sizes. It has wide applications in different fields such as pattern recognition, text classification, analysis of image database clusters, etc.

Logistic regression is among the oldest classification techniques. A linear classifier having a decision boundary is called LR. Instead of predicting classes, it predicts probabilities. It is effective to use a logistic regression classifier to predict categorical outcomes. Nevertheless, the prediction necessitates the independence of every data point [23]. LR is simple to use, has low processing resource requirements, and no preprocessing of input features are needed. Survey analysis, marketing, medicine, and credit scoring issues are among the fields that use LR.

2 Literature review

Software defect classification plays an important role in improving the quality of software. It can be used to reduce efforts and costs related to resolving the bugs. Software bug classification helps in effective team management, cost-effectiveness, faster bug identification, and identifying the strengths and weaknesses of software components. Bug classification can help in Triaging, i.e., assigning bugs to related developers, which saves time and effort [24]. Classification and identification help software quality engineers and managers to measure how software projects evolve.

Nagwani and Verma (2014) compared ten standard algorithms for classifying software bugs into two main classes: bug and non-bug. They divided the process into three categories: (1) identify bug dataset, (2) preprocessing, and (3) classification. Experiments were performed using Java and Weka API. They found that classifier accuracy decreases when the number of classes increases. These algorithms are implemented on four open sources: Android, JBoss-Seam, Mozilla, and MySQL. They concluded from the results that four algorithms, namely, SVM, bagging, RBF, and regression perform better, their accuracy is 90.1% [3].

Behl et al. (2014, February) identified SBRs through NB and TF-IDF. They pre-processed the existing dataset of 10,000 security bug reports from the Bugzilla repository through text mining. They identified start and stop word lists to use in classification algorithms for predicting. Their result shows that TF-IDF had a high success rate of 93.989% [20].

Kukkar and Mohana (2018) used TF-IDF, bigram, and K-NN classifiers to identify bug reports as bug or non-bug. They proposed four fields (reporter, severity, priority, and component) to be added to bug reports to increase the performance of a classifier. They observed that the performance of the KNN classifier changed according to the dataset and the bigram method improved the performance. They used five datasets of bug repertories, that are JBoss, Firefox, Open FOAM, Mozilla, and Eclipse. The performance of the algorithm is 89% for the bug report and 94% for the non-bug report [5].

Zou et al. (2018) identified bug reports as SBR and non-security bug report (NSBR) through NLP processing and ML techniques on the Bugzilla dataset. For classification, meta features (time, severity, and priority) and textual features (the text in summary fields) with the SVM classifier were used. The experimental results show that SBR with imbalanced data processing can successfully identify SBRs with 99.4% higher accuracy and 79.9% recall compared to existing work. The experimental findings indicate that SBR with imbalanced data processing can successfully identify the SBRs with a higher precision of 99.4% and recall of 79.9% [25].

Zhou et al. (2016) presented a hybrid approach of combining text mining and data mining techniques to identify corrective defect reports by combining multinomial NB and Bayesian Net, for better textual classifiers. Their work was evaluated on five bug datasets, Mozilla, Eclipse, JBoss, Firefox, and OpenFOAM by using precision and recall rates. Comparative experiments were performed with previous studies and the results were found to be better than the previous studies 73.8 to 81.7% for Mozilla. They used the WEKA toolset to automate the frequency calculation and validate through ten-fold cross-validation [8].

Otoom et al. (2019, August) classified software bug reports as perfective and corrective using classification algorithms NB, SVM, and random trees (RT). They evaluated the result on three different open-source projects: AspectJ, Tomcat, and SWT projects. They used the WEKA tool and carried out five-fold cross-validation that achieved higher accuracy (93.1%) using the SVM classifier [26].

An extensive study on ML methods that have been effectively applied to predict software bugs – both bug and non-bug – is presented by Khleel and Nehéz (2021). The supervised ML algorithms DT, NB, Random forest

(RF), and LR are the foundation of the software bug prediction model they present. For this experiment, four NASA datasets were used, and model performance was examined. According to the experimental results, DT and RF classifiers outperform other classifiers with an accuracy of 99% in prediction [27].

Choudhary (2017) predicted bug priority using MLP and Naïve Bayes classification algorithm and showed results through receiver-operating characteristic curve (ROC) and *F*-measure. The four main processes performed in experiments are dataset acquisition, pre-processing, feature selection, and classification. They preprocess text by tokenization, stop word removal, and stemming. They performed on Eclipse bug reports of three components (JDT, PDE, and Platform) with the precision of MLP and NB being 82%, which is better than the previous values [1].

Gegick et al. (2010, May) predicted SBRs from Cisco software. They introduced an industrial text mining tool that evaluated the natural-language models on a large Cisco software system called SAS Text Miner5. Their approach is made up of three main steps. To begin with, a labelled dataset must be obtained. Creating three text mining configuration files – a start list, a stop list, and a synonym list – is the second step. The training, validation, and testing phase is the third. 78% of the SBRs were successfully classified by their model, which also predicted that they would be classified as SBRs with a probability of 0.98% [28].

Zaman et al. (2012, June) used Mozilla and Chrome bug reports and classified the performance bug reports by feature selection and found out that the "step to reproduce" is important for the performance of software [29].

Goseva-Popstojanova and Tyo (2018, July) employed both supervised and unsupervised techniques to automatically categorize software bug reports as security or non-security issues. Three different feature vector types were used: Term Frequency (TF), Binary Bag-of-Words Frequency (BF), and TF-IDF. These were combined with a variety of learning algorithms, including RF, NB, KNN, Bayesian network, and SVM. For this experiment, they examined the performance of three NASA datasets. Compared to other classifiers, RF classifiers have a prediction accuracy of 82%, according to the experimental results [30].

Zheng et al. (2021) conducted research on six real-world projects with more than 100,000 bug reports, varying in size. They first examined how the class imbalance problem affects SBR prediction and verify that it has a detrimental effect on prediction performance. After that, they compared six class rebalancing strategies with five well-known classification algorithms for SBR prediction. They come to the conclusion that the best performing model can be created by combining the ROSE and RF classification algorithms, which improves performance by an average of 75% and, in the best case, 267% in terms of the *F*1 score [31].

Yadav and Pal (2015) integrated both technique classification and clustering for error detection based on time and error, and they found the K-Means and Bayes Net are better than others (J48GRAFT, LAD TREE, and BAYESNET) [32].

Odera and Odiaga (2023) classified spam and ham SMS using Recurrent neural network (RNN) and compared their results with SVM. They used UCI SMS spam Dataset v.1 and preprocessing data including case conversion, punctuation mark removal, abbreviation expansion, tokenization, stemming, and stop words removal. They evaluated the results using area under the curve (AUC) (Accuracy loss). According to the results, SVM's false positive rate is somewhat lower than that of RNN, which has a slightly higher training and validation accuracy of 0.98 compared to 0.94 [33].

Aljedaani et al. (2022, April) classified the bug reports into accessibility-related and non-accessibility-related bugs. They used two popular issue-tracking systems Bugzilla and Monorail repositories, to clean up data using preprocessing steps, i.e., tokenization, special character removal, and lemmatization. They evaluated five different ML algorithms; DT, RF, decision jungle (DJ), SVM, and NN to observe which one offers the most successful outcomes for the classification of accessibility bug reports. They conclude that DT's *F*1-score of 93% outperforms all other classifiers in terms of evaluation parameters [34].

Alqahtani (2022) used ML classifiers FASTTEXT, RF, NB, KNN, MLP, and LR to classify SBR and NSBR. He used datasets from five Java projects which include Chromium, Amabri, Camel, Derby, and Wicket. They preprocessed a line of words using stop words, tokenization, and derive bag of words, to evaluate results including recall (R), probability of false alarm (pf), precision (P), *F*1-score (*F*1), and *G*-measure (*G*). They found that the fasttext classifier can effectively improve the classification of SBRs with an average *F*1-score of 0.81. It also achieved an average *G*-measure of 0.80 [13] (Table 1).

Table 1: Literature review summary

Reference no.	Repository	Classification algorithm	Bug type		Pre-processing						Validation	
			Security	Functional requirement	Stemming/ start list	Label	Token	Stopword	N-gram	TF-IDF	K- Fold Cross	Results
[1]	JDT, PDE	NB, MLP	✗	✗	✓		✓	✓	✗	✗	✓	82%
[3]	Android, JBoss, Mozilla, MySQL	SVM, NB, ANN, DT	✗	✓	✓	✗	✗	✓	✗	✓	✓	90%
[5]	JBoss, Firefox, Mozilla, Eclipse	KNN	✗	✓	✓	✓	✓	✓	✓	✓	✗	94%
[8]	Mozilla, Eclipse, JBoss, Firefox	MNB, BN	✗	✓	✓	✓	✓	✓	✗	✗	✓	81%
[13]	Chromium, Amabri, Camel, Derby, Wicket	Fasttext, KNN, MLP, LR, NB, RF	✓	✗		✓	✓	✓	✗	✗	✓	81%
[20]	Bugzilla	NB	✓	✗	✓	✗	✗	✓	✗	✓	✗	93%
[25]	Bugzilla	SVM	✓	✗	✗	✓	✓	✓	✗	✗	✓	99%
[26]	AspectJ, Tomcat, and SWT	NB, SVM, RT	✓	✗	✗	✗	✗	✗	✗	✗	✓	93%
[27]	NASA	DT, RF, LR, NB	✓	✗	✓	✓	✓	✓	✗	✗	✗	99%
[28]	Cisco	SAS Text Miner5	✓	✗	✗	✓	✗	✓	✗	✗	✗	78%
[30]	NASA	BN, KNN, NB, *NBM*, RF, SVM	✓	✗	✓	✓	✗	✓	✗	✓	✗	82%
[34]	Bugzilla, Monorail	DT, RF, DJ, RF, NN	✗	✗		✓	✓	✓	✗	✗	✓	93%

2.1 Research gap

It is evident from the literature [1,3,5,8,20,25–28] that SVM, ANN, NB, RBF, KNN, LR, DT, FR, TF-IDF, and N-gram have been used as classification techniques upon various bug repositories, e.g., Bugzilla, Android, Eclipse, JBoss, Firefox, OpenFOAM, and NASA to classify bugs for functional requirements [30,34]. However, the classification of bugs into performance and security is yet to be done. Besides a comparative analysis of existing ML algorithms upon the Bugzilla repository, which will facilitate Software Quality Assurance managers to choose relevant algorithms in future classification, is also missing in the literature. Such comparative analysis of classification algorithms for the identification of security and performance bugs can not only help in triaging but will also facilitate software evolution and maintenance.

3 Materials and methods

In this study, we employed the explanatory research design. The goal of explanatory research is to elucidate a problem's causes and effects. The optimal bug classification algorithm will be identified if the problem is well-defined. In order to establish the cause-and-effect relationships between the variables for accurate identification, this research paper will employ experimental research methods to manipulate and control the variables. Our experiment is designed to identify defects earlier using developed, mature projects as a basis. By using mature source projects to train the classifier, we will be able to identify defects earlier and classify them in the target projects. Prediction accuracy (precision, recall, and *F*1) is the dependent variable in this research paper, while the algorithms are the independent variables. The design of this study is 1 factor 1 treatment. Furthermore, results are validated through cross-validation, e.g., K fold, etc., since it enables us to compare different models and parameters, providing a consistent and fair way of evaluation. Our research question is:

RQ 1: What is the impact of classification algorithms, i.e., SVM, ANN, LR, KNN, and NB upon identification accuracy of security and performance bugs?

Ho: There is a significant impact upon identification accuracy of security and performance bugs through any classification algorithm, i.e., SVM, ANN, LR, KNN, and NB.

H1: There is no significant impact upon identification accuracy of security and performance bugs through any classification algorithms, i.e., SVM, ANN, LR, KNN, and NB.

3.1 Proposed classification process

In this section, we proposed a classification process. We train our algorithms and predict the defect's prediction accuracy. By following this approach, we will detect/classify the defects earlier, and allocate bugs to the related triager and debugger with efficient use of time and resources. We identified the bug reports from the Bugzilla bug repository and manually labelled data into two classes using the keyword-matching technique.

After labelling, we perform pre-processing steps including lower casing, tokenization, stop word removal, TF-IDF, and *n*-gram and then apply classification steps and compare the results. Figure 1 shows the detailed methodology of the proposed research.

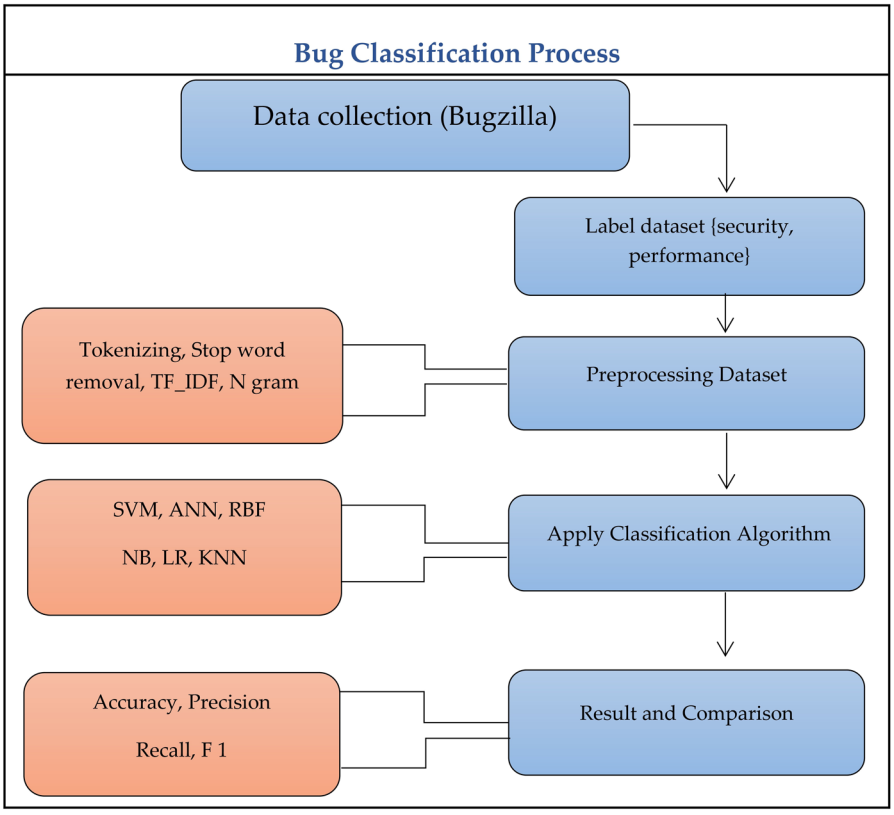
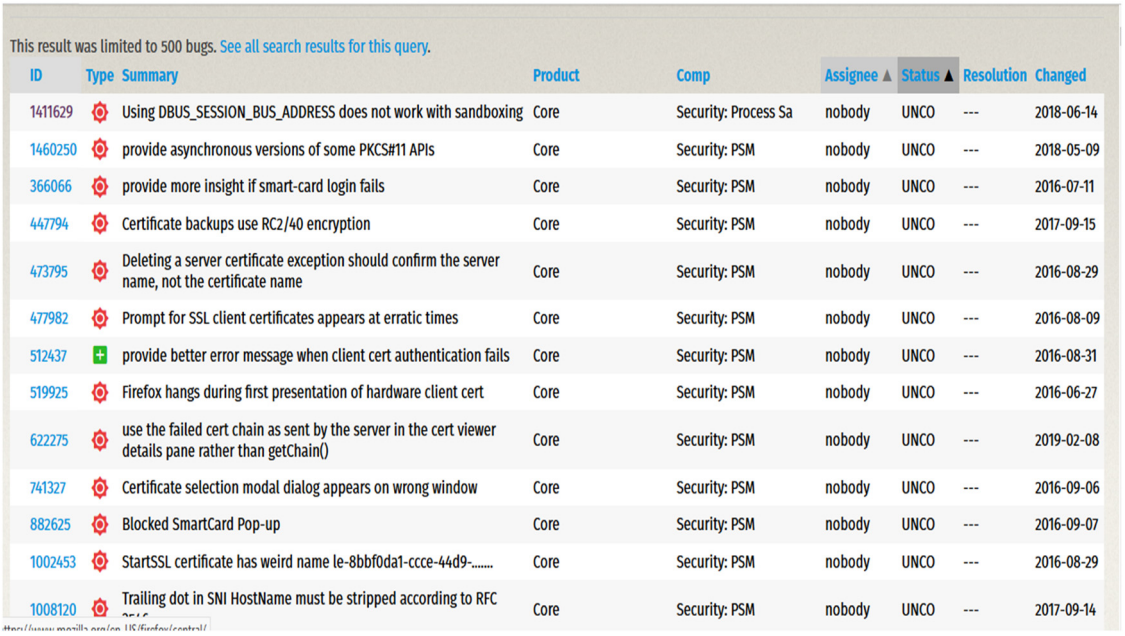


Figure 1: Blug classification process.

3.1.1 EDA

We performed Exploratory Data Analysis (EDA) on the Bugzilla repository. The purpose of EDA is to extract the most important features, remove unnecessary features, and map and understand the structure of the data. We used the Bugzilla repository, which has nine field attributes that show the bug information, i.e., bug Id, bug summary, bug type, product, component, assignee, status, change, and description. We extract features from the BUG SUMMARY field [21]. Bugzilla bug report contains the details of a software issue; the information is organized into several fields as follows:

- **BUG ID:** The number ID of a bug that is unique for the bug tracking system.
- **TYPE:** Describe the type, whether it is a bug or new requirement.
- **SUMMARY:** One line short description of a bug containing a few keywords which is defined as the bug also known as the title of a bug report.
- **PRODUCT:** This term describes which among all the products e.g., thunderbird, firefox, seamonkey etc has a bug.
- **COMPONENT:** This term describes which among all the components e.g., Security, Performance, general etc. has a bug.
- **ASSIGNEE:** The name of the person who assigns this bug.
- **STATUS:** The current state of a reported bug. Two types of bugs are there: unconfirmed and confirmed; one is an open bug and the other is a closed bug.
- **CHANGED:** The date when the bug status changed.
- **DESCRIPTION:** Detailed description of a bug. This comprises the reproduction steps: simplified, simple instructions that will cause the bug. Make sure to include any special setup steps, if applicable. Real Results: The actions taken by the application following the aforementioned procedures. Expected Outcomes: the user's expectations following the aforementioned action [2].



This result was limited to 500 bugs. See all search results for this query.

ID	Type	Summary	Product	Comp	Assignee ▲	Status ▲	Resolution	Changed
1411629		Using DBUS_SESSION_BUS_ADDRESS does not work with sandboxing	Core	Security: Process Sa	nobody	UNCO	---	2018-06-14
1460250		provide asynchronous versions of some PKCS#11 APIs	Core	Security: PSM	nobody	UNCO	---	2018-05-09
366066		provide more insight if smart-card login fails	Core	Security: PSM	nobody	UNCO	---	2016-07-11
447794		Certificate backups use RC2/40 encryption	Core	Security: PSM	nobody	UNCO	---	2017-09-15
473795		Deleting a server certificate exception should confirm the server name, not the certificate name	Core	Security: PSM	nobody	UNCO	---	2016-08-29
477982		Prompt for SSL client certificates appears at erratic times	Core	Security: PSM	nobody	UNCO	---	2016-08-09
512437		provide better error message when client cert authentication fails	Core	Security: PSM	nobody	UNCO	---	2016-08-31
519925		Firefox hangs during first presentation of hardware client cert	Core	Security: PSM	nobody	UNCO	---	2016-06-27
622275		use the failed cert chain as sent by the server in the cert viewer details pane rather than getChain()	Core	Security: PSM	nobody	UNCO	---	2019-02-08
741327		Certificate selection modal dialog appears on wrong window	Core	Security: PSM	nobody	UNCO	---	2016-09-06
882625		Blocked SmartCard Pop-up	Core	Security: PSM	nobody	UNCO	---	2016-09-07
1002453		StartSSL certificate has weird name le-8bbf0da1-ccce-44d9-.....	Core	Security: PSM	nobody	UNCO	---	2016-08-29
1008120		Trailing dot in SNI HostName must be stripped according to RFC	Core	Security: PSM	nobody	UNCO	---	2017-09-14

Figure 2: Bugzilla bug repository from 2019.

Figure 2 shows a bug report of an open-source bug repository, i.e., Mozilla/Bugzilla. The vertical columns describe the field that consists of nine attributes of the Bugzilla, and the horizontal rows describe the instance/record of the bugs. After the EDA, we came to know Bugzilla repository has multiple attributes and most of them are unnecessary and affect prediction accuracy. So, we removed the noise by removing the redundant data, dropped all the unimportant columns such as the STATUS and PRODUCT, removed missing instances and extracted important features like SUMMARY. That efficiently trained our models to classify bugs with higher accuracy. We collected 1,000 instances of bug reports and labelled them manually into two classes; 550 securities and 450 performances (Table 2).

Table 2: Bugzilla dataset detail

Projects	Bug reports	Security	Performance
Bugzilla	1,000	500	450

3.2 Pre-processing

Data preprocessing describes any type of processing performed on raw data to prepare it for another processing procedure [35]. Preprocessing includes the following steps. Data Cleaning: Tokenization, stop words removal, *n*-gram, TF-IDF, steaming, lemmatization, and also include spelling correction [20]. Text preprocessing is a prerequisite step that prepares the data for further processing and analysis. Both these techniques are important in NLP and are often used together in text-based applications.

3.2.1 Data cleaning

In data cleaning, we clean data from noise by removing redundant data and missing values and taking useful attributes. We used only one column for feature extraction and we removed the other columns.

3.2.2 Label dataset

First, we identified more than 200 security and 60 performance-related keywords with the help of security experts and from literature. We made two lists: A (security) and B (performance), which are shown in Table 3.

Table 3: Keywords of security and performance

Type	Keyword
Security	Confidentiality, security, PKCS, virus, authorization, audit, biometric, validation, encryption, verify, key, password, alarm, encryption, noise, certificates, verification, trust, sign, SHA, smartcard, decryption, SSL, fingerprint, smartcard, decryption, OCSP, rights, TLS, blacklist, cryptography, Hash function, SIMME, auth, biometrics, rule, validation, access control, scam, MD5, restrict, cookies, hijacking, proxy, SMPT, squid, token, permission, code, secure, defense, Trojan, untrusted, 2f, warning, privacy, privileges, hookworm, authentication, and restrict.
Performance	Execute storage, dynamic, throughput, peak, load, stress, volume, capacity, mean, space, time, response, memory index, runtime, reduce, fixing, early, offset late, completeness, compress, uncompressed, and perform.

We match/find the keyword in the bug summary field. If any word matches with list A, we label bug summary as security. If any word matches with list B, we label bug summary as performance (Figure 3).

	A	B	C
1	BUG-ID	SUMMARY	CLASS
2	1288988	changed server not reflected in saved password list	security
3	487269	allow selection of a bugzilla group with testopia's access control list	security
4	530748	when selecting product dashboard in testopia - i get not authorised error - same in ff	security
5	559576	the test run assignee should be able to update test runs w/o requiring write perm to	security
6	580311	user can see all test runs and cases not only those for which products, he has access	security
7	593817	secreview: make tab strip async	security
8	769052	feature request: a list of mail addresses to which smime signatures should not be ser	security
9	227405	request for a secure site tray/taskbar type notice/advisory screen and notification ico	security
10	296249	allow for advanced challenge/response authentication (optical flickering)	performance
11	296598	ns_error_dom_security_err for menulist.xml	security
12	315494	support time-stamp protocol (tsp) as per rfc3161	security
13	327493	emails flagged as possible email scam should be rendered as simple html (similar to r	security
14	332550	restrict ajax/javascript scope to dom element.	security
15	344945	startup reports 'could not initialize the browser's security component'	security

Figure 3: Dataset after EDA.

3.2.3 Tokenization

The process of tokenization involves using lexical analysis to divide a sentence into a collection of tokens or words based on delimiters like spaces and punctuation marks. In order to obtain a large corpus of tokens, special characters should be eliminated and the tokens should be converted to lower case during the tokenization process. We use the word2vec (Bag of words) tool, which is commonly used in text mining, to map these tokens into vectors. For tokenizing, we also used TF-IDF and *N*-gram [4,11,18] (Table 4).

Table 4: Tokenization

Firefox account sends an authorization code even if a password is incorrect	Security
org.mozilla.jss.ssl.sslsocket leaks memory on instantiation	Performance
After tokenization	
"Firefox," "account," "send," "authorization," " code," "even," "if," "password," "is," "incorrect,"	"Security"
"Org," "Mozilla," "jss," "ssl," "sslsocket," "leaks," "memory," "on," "instantiation"	"Performance"

3.2.4 *N*-gram

N-gram is an alternative to the word tokenizer. *N*-gram uses more than one word as a token. These are *n*-word word sequences that follow one another. Bigrams, e.g., are tokens made up of two words that are placed next to each other; a unigram is a single word. An *N*-gram is an *N*-character of a longer string *N* would be ($N = 1$) that is called a unigram, ($N = 2$) is a bigram and ($N = 3$) is a trigram. We used this technique in tokenizing, to get better and more meaningful results [18,23,36,37] (Table 5).

Table 5: *N*-gram

Firefox account sends authorization code even if password is incorrect	Security
org.mozilla.jss.ssl.sslsocket leaks memory on instantiation	Performance
After applying *N*-gram = 2	
"Firefox account," "send authorization," " code even," "if password," "is incorrect"	"Security"
"Org Mozilla," "jss ssl," "ssl sslsocket," "leaks memory," "on instantiation"	"Performance"

3.2.5 Stop words removal

The text uses a lot of words, many of which do not refer to any important information. Stop words include conjunctions (e.g., and, but, then), pronouns (e.g., he, she, it), and articles (e.g., a, an, the). Because these stop words affect the identification algorithm's performance, they must be eliminated from the set of tokens created in the preceding step [11] (Table 6).

Table 6: Stop word removal

Firefox account sends authorization code even if password is incorrect	Security
org.mozilla.jss.ssl.sslsocket leaks memory on instantiation	Performance
After removing stop word	
"Firefox," "account," "send," "authorization," " code," "even," "password," "incorrect"	"Security"
"Org," "Mozilla," "jss," "ssl," "sslsocket," "leaks," "memory," "instantiation"	"Performance"

3.2.6 TF-IDF/Doc2vec

The features play a major role in constructing a predictive model that significantly affects performance overall. Therefore, before training the model, it is crucial to identify the best features that contribute to high classification accuracy. TF-IDF is a helpful weighting scheme used in text mining and information retrieval. To identify the best feature in the dataset, we apply the TF-IDF feature extraction technique. The term's significance in a document within a corpus is indicated by TF-IDF. It is a feature weighting scheme used to represent the importance of words in documents, while text preprocessing involves tasks performed on raw text data to clean and standardize it before analysis or modeling. TF-IDF can be considered as a part of the

feature engineering process which can be used for training ML algorithms. bag of words (BoW) is a method for preparing text for input (features) in classification algorithms. The advanced term of BoW is TF-IDF which generates a word vocabulary and converts it into a vector, and calculates the term frequency of all words [30,23]. It creates a vocabulary of all the unique words occurring in all the documents in the training set. The use TF-IDF feature extraction method helps in finding the occurrence feature in the document. While tokenizing, tokens should also be changed to lowercase, and special characters should be removed. We get a large vocabulary of all the distinctive words that occur in all the training data [5,20].

TF (w) = (Number of times term w appears in a document)/(Total number of terms in the document)

IDF (w) = log (Total numbers of documents/Number of documents with term w in it)

$$\mathrm{TF} - \mathrm{IDF}(\mathrm{w}) = \mathrm{TF} * \mathrm{IDF}$$

To normalize data, noise is first removed from the dataset. The distribution gap between the source and target datasets is then closed. Finally, we use CTGAN synthesizer to resolve the class imbalance issue. After all of these processes, we have normalized the data, which we can use for more experiments. We will go into great detail about each stage of the data preprocessing process. Compared to the Bugzilla repository (2020), which has been the subject of prior research, the updated repository (2024) has different feature values [16].

3.3 Model tuning

We have generated our results after tuning algorithm values and parameters. We get better classification accuracy after modifying training and testing size, by adjusting hidden layers, weights, epochs, and other parameters.

3.4 Performance measurement

The commonly used measures are accuracy, precision, recall, *F*1 (*F*-measure), ROC, and AUC. The most used performance metric is accuracy, which is closely followed by recall, precision, and *F*1-score. These measures are derived from the confusion matrix. We validate our result by applying ten-fold cross-validation with 70% by 30% training and testing [15,16]. The confusion matrix has 4 values:

TP "stands for true positives, which indicates the positive values that the system has predicted as positives."

TN "is true negatives that are negative values the system identifies as negatives."

FP "is false positives, negative values the system identifies as positives."

FN "is false negatives, positive values that the system predicted as negative. By using their values we measure Accuracy, Recall, *F*1, and Precision."

Accuracy (correct classification rate) is defined as the correctly predicted number of bugs to the total number of bugs [2].

$$\text{Accuracy} = \frac{\text{True positive} \ + \ \text{True negatives}}{\text{True positive} + \text{True negatives} \ + \ \text{False positives} + \text{False negatives}}.$$

Precision "is the percentage of the number of correctly classified bug reports that are predicted positive" [23].

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} \ + \ \text{False positives}}.$$

Recall "is the percentage of the number of correctly classified bugs that are Actual positive bugs" [23].

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} \ + \ \text{False negatives}}.$$

***F*1-measure/*F*-score** "considers both precision and recall equally important by taking their harmonic mean. The higher value of the *F*-measure indicates a higher quality of the classifiers." *F*-measure is calculated as follows [20]:

$$F \text{ measure} = 2\,(\text{precision} \times \text{recall})/(\text{precision} + \text{recall}).$$

3.5 The proposed algorithm

Input: CSV file of bug reports
Output: Classify bug report as security and performance bug report and the results are shown as accuracy, precision, recall, and *F*1 measure.
Step 1: Input CSV file only with the bug summary part that is in text data
import pandas as pd
data = pd.read_csv('bugzilladataset.csv');
Step 2: Spilt dataset 70 by 30, 70% for training and 30% for testing.
from sklearn.cross_validation import train_test_split
X_train_raw, X_test_raw, y_train, y_test = train_test_split (data[1],data[0],test_size = 0.30)
Step 3: Give text data for preprocessing {Tokenization → Stop word removal → TF_IDF → Ngram}
from nltk.corpus import stopwords
Step 4: Apply TF-IDF and N-gram and extract features. Also called bag of word vocabulary (V).
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')
Step 5: Apply classification algorithm (SVM, ANN, NB, KNN, LR)
from sklearn import svm
svm = svm.SVC(kernel = 'linear')
Step 6: Apply ten-fold cross validation
from sklearn.model_selection import cross_val_score
accuracy = cross_val_score(svm, X_train, y_train, cv = 10).mean()
Step 7: Apply classification confusion matrix
from sklearn.metrics import classification_report,confusion_matrix
print (classification_report(y_test,predictions))

---

4 Result

In this section, we will show the result of all algorithms that we apply in this study, which helps to answer the research questions. Our process broadly falls in the explanatory research, explaining the causes and consequences of a problem, i may be taken as an experiment first, we show algorithm results one by one and then we compare the results.

4.1 NB

NB is mostly used for text classification that uses the Bayesian theorem that calculates the frequency of attributes in an instance and gives the class a high probability weight. When the number of features is large the calculation probability is difficult so filtering the feature is more efficient. The precision of algorithms decreases if the number of datasets is less [22]. Because of its ease of use and efficiency, NB is widely employed in text classification applications [31]. Bernoulli Naïve Bayes (BNB), Multinomial Naïve Bayes (MNB), and Gaussian Naïve Bayes (GNB) are the three models that NB offers. We used the function of BNB model and performed classification tasks (Table 7).

Table 7: Specified model tuning parameters of NB

Algorithm	Alpha	Binarize	class_prior	fit_prior	random_state	min_df	test_size	cv
BNB	1.9	0.0	None	True	1	3	0.30	10

NB model is good at making true positives by reducing the number of false positives and yielding high precision for performance bugs. It means that the classifier is good at correctly identifying bugs while minimizing false alarms. This could be due to unique bug-related features, clean data, imbalanced classes, or effective features.

4.2 ANN

NN is made up of connected artificial neurons. The neurons' interconnection link spreads a certain weight. In case NN are trained with training patterns of known classes, these are called supervised learning [15]. The most frequently used feed-forward NNs for classification include multilayer perceptron and RBF network [3]. In the case of MLP, neurons are connected in a network. Mostly three layers of MLP are used which contain the input, hidden, and output layers [18] (Table 8).

Table 8: Specified model tuning parameters of ANN

Algorithm	Activation	Batch size	Hidden layer	Random_state	min_df	Test_size	cv
MLP classifier	Relu	Auto	(5,2)	1	3	0.25	10

After tunning various parameters, we achieved a high precision rate of 0.96% for performance bug data which is higher than the security precision rate of 0.94%. ANN are capable of handling nonlinear data relationships, performing automatic feature extraction, and have a large parameter space that allows them to closely fit the training data. We also achieved a security bug's $F1$ score of 0.95%, which is greater than the performance bugs F-measure of 0.93%. Artificial neural network (ANN) are used when there are only a few parameters to tune. Moreover, their learning efficiency improves as the volume of training data increases [17]. ANN has various techniques for classification, such as MLP, SLP, RBF, and deep learning; we use MLP. MLP has several attributes that can be modified, including hidden layer, learning rate, and maximum iterations. Mostly we use default values; by changing values, we get better results but it is a time taking work.

4.3 LR

One method of classification is LR. LR is a linear classifier with a decision boundary. LR is used in medicine, advertising and survey analysis, credit scores, public health, and other apps. Where the data are noisy and non-separable, LR converges far more quickly and reliably. We used default values of the attribute and modified penalty = "l2" and "liblinear" solver value that affects the accuracy result. LR achieves high precision and $F1$ scores for performance bug data over the security bugs data because of class imbalance, data quality, feature importance, threshold settings, and more predictable or clearer patterns (Table 9).

Table 9: Specified LR model tuning of parameter

Algorithm	Penalty	C	random_state	test_size	min_df	cv
liblinear	"l2"	3	1	0.30	3	10

4.4 KNN

KNN algorithm is used to classify by finding the K (int) nearest matches in training data. Use similarity scores among texts and identify the KNN. KNN works well with a small number of input variables. It is strong for noisy training data. KNN does not perform well on imbalanced data. KNN does not deal with the missing value problem. We modify the k value to find the neighbor values and then calculate the distance and similarity score (Table 10).

Table 10: Specified model tuning parameters of KNN

Algorithm	n_neighbors	random_state	test_size	min_df	n-gram_range	cv
KNN classifier	50	0	0.30	4	(1, 1)	10

We used the default attribute and changed the value of k to achieve better results. We set the value of k 50 near the neighbor and got 92.4% accuracy. When we decrease the value of k, the accuracy decreases. We achieve a high $F1$_measure of 0.94% for the security bug dataset and a high precision of 0.97% for performance bug dataset as shown in Figure 4. KNN gets high precision for performance class. This can be due to the class's data points being well-separated, the K value is changed and features are highly relevant for that class.

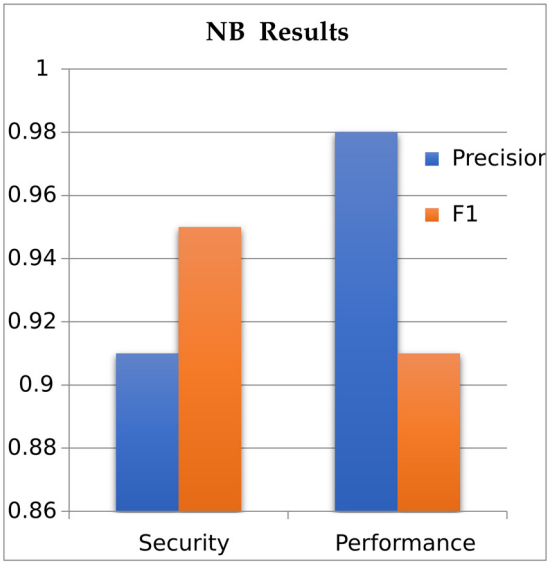


Figure 4: Presents the result of KNNs for the identification of security and performance problems.

4.5 SVM

Support vector machine is a widely used supervised learning technique for classification. This innovative method of learning is primarily applied to binary classification [7,27]. SVM uses kernels to solve classification

problems. SVM kernels are employed in various classification applications. We employed the linear function, but other well-known kernel functions include polynomial, RBF, sigmoid, and linear. It improves the performance of classification by removing irrelevant features. When learning tasks involve a high number of features in some training instances, SVM is a good fit. SVM contains numerous "linear," "RBF," and "poly" kernels. After using them all, we discover that linear kernels work best with our dataset (Figure 5) (Table 11).

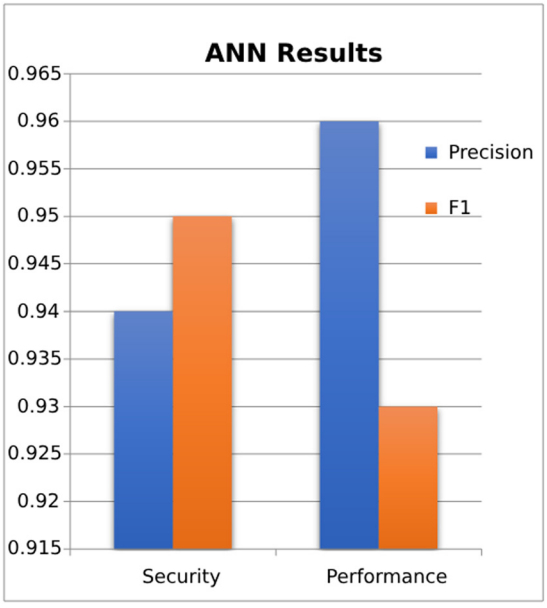


Figure 5: Displays the precision and *F*1 results of the NB model in classify security and performance bug reports. BNB classifier has a higher precision rate for performance bugs (0.98) compared to security bugs (0.91). However, the *F*1 score is higher for security bugs (0.95) than for performance bugs (0.91), with a difference of 0.4%.

Table 11: Specified model tuning parameters of SVM

Algorithm	Kernel	C	Class_weight	Verbose	gamma	Test_size	Min_df	Random_state	cv
SVM.SVC	linear	1	None	False	6	0.30	4	2	10

We used *F*1 and precision for comparison that shows the predicted results of the algorithm. SVM is well-suited for learning tasks involving a large number of features and a small number of training instances. SVM gets high precision and recall values for the performance bug dataset because SVMs handle imbalanced datasets by adjusting the class weights and *C* parameters. This flexibility enables them to prioritize precision over recall. SVMs handle nonlinear relationships in the data using kernel functions (e.g., polynomial, RBF). It also generalizes well with small sample sizes, making them applicable in situations where limited data are available. It focuses on support vectors, which are data points near the decision boundary. This helps the model to concentrate on the most challenging instances, contributing to better precision and recall.

5 Findings and comparative analysis of classification algorithms

In this section, the impact of classification algorithms, i.e., SVM, ANN, LR, KNN, and NB on identification accuracy of security and performance bugs is analyzed. Five supervised ML algorithms are analyzed and evaluated in this study, which are SVM, NB, ANN, KNN, and LR. These algorithms were chosen based on the LR of previous studies. After applying the algorithms (NB, ANN, SVM, LR, and KNN) on the performance bug repository, we measure the results through precision and *F*1 measure.

Figure 6 shows that the precision rate of algorithms is higher as compared to the $F1$ score for performance bugs as we see two algorithms (SVM and LR) perform better in terms of precision (0.99%) as compared to other algorithms (NB, KNN, and ANN). High precision means that the model is good at making positive predictions, and when it classifies a data point as positive, it is often correct. This suggests that there are relatively few false positives in the model's predictions. Here the precision rate is high while the $F1$ score is low for the performance dataset, which typically occurs when one of these metrics is significantly lower than the other. In this case, the low $F1$ score is likely due to low recall. Several factors contribute to this low recall, including the existence of an imbalanced dataset, potential biases in the model favoring the majority class, the use of harsh decision thresholds demanding high confidence for positive predictions, and inherent difficulties in distinguishing between classes when they share significant similarities. After applying algorithms on the security bug repository, we compare results through precision and $F1$ measure.

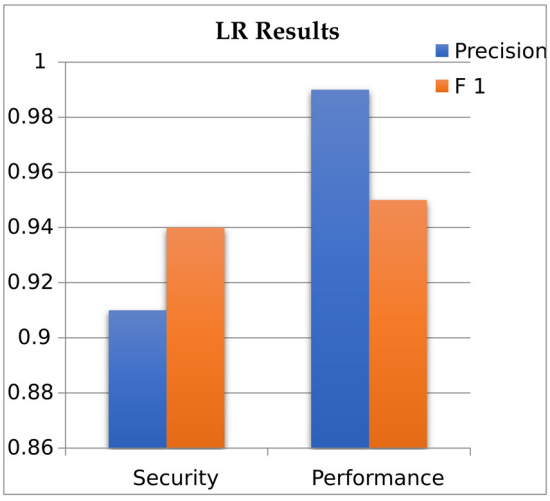


Figure 6: Comparative analysis of security and performance bug through SVM.

Figure 7 shows that the $F1$ score of all algorithms is higher as compared to the precision score for security bugs as we see three algorithms (SVM, ANN, and LR) perform better in terms of $F1$ score as compared to other algorithms (NB and KNN). A high $F1$ score means high recall which indicates that the model is good at

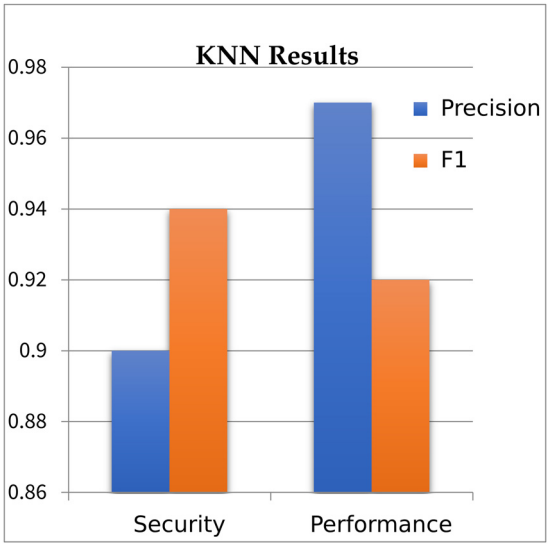


Figure 7: Algorithm result of SBR.

identifying a large proportion of the actual positive instances in the dataset. This suggests that there are relatively few false negatives (instances that are positive but predicted as negative). When precision is low but the *F*1 score is high for a dataset, it typically signifies a situation where the classification model prioritizes recall over precision. This commonly occurs in scenarios such as imbalanced datasets, where the model classifies the minority class more generously, leading to higher recall but lower precision due to an increased number of false positives.

Additionally, high sensitivity, threshold adjustments, the emphasis on reducing false negatives, and specific business requirements can contribute to this pattern. The balance between precision and recall depends on the application's goals, and in cases where minimizing false negatives is critical, a lower precision but higher *F*1 score may be preferred. After applying NB, ANN, SVM, LR, and KNN algorithms on the security and performance bug repository, we compare the results through accuracy, precision, and *F*1-measure. Table 12 shows the average accuracy of performance and security bug classification. As seen in Figure 8, algorithms get higher precision scores compared to *F*1 for performance bugs and Figure 9 shows that algorithms get higher *F*1 scores as compared to precision scores for security Bugs, but in Figure 10, we get an average of precision, *F*1, and accuracy rate for both security and performance Bugs.

Table 12: Classification algorithm result of security and performance bugs

Evaluators	SVM	ANN	NB	LR	KNN
F measure	0.961	0.954	0.956	0.947	0.934
Accuracy	0.957	0.950	0.951	0940	0.927
Precision	0.951	0.946	0.933	0.909	0.916

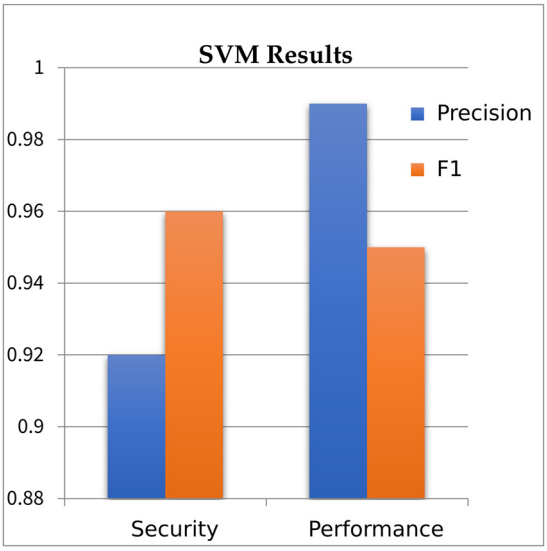


Figure 8: Presents comparative analysis of security and performance bug through SVM. We achieved a performance precision score of 0.99, which is 0.7% greater than the security precision score (0.92). The *F*1 measure shows the average of precision and recall which is 0.95% for performance and 0.96% for SBRs.

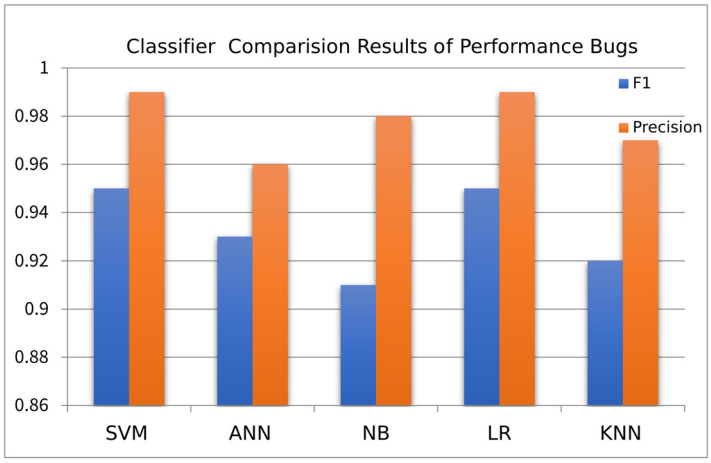


Figure 9: Presents the precision and *F*1 result of ANN for the identification of security and performance bug.

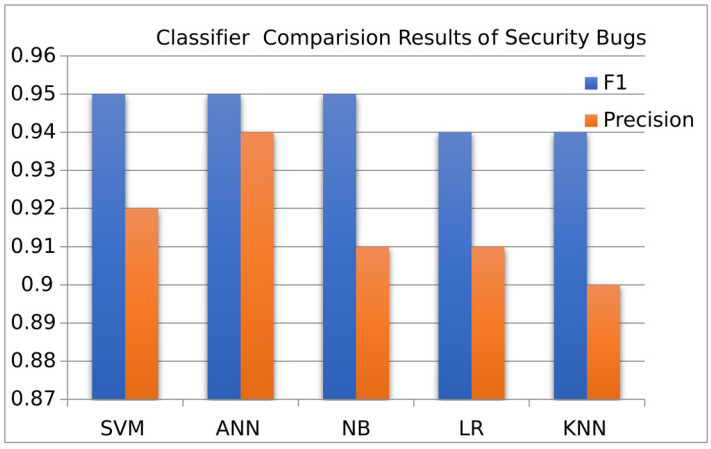


Figure 10: Presents the results of LR for the classification of security and performance problems. LR performs better for performance bugs. For performance, LR achieves a 0.99 precision score and 0.95 *F*1-measure score.

So, we can conclude that SVM performs better as compared to other algorithms for security and performance bug datasets. SVM accuracy rate is 0.95%, the *F*1-score is 0.96%, and the precision score is 0.95% on the Bugzilla dataset. The addition of the TF-IDF method improves the performance of classifiers. Results are shown in Figure 11.

Figure 11 shows the results of all classification algorithms which we applied to the dataset and found that the SVM has the highest *F*1 measure of 95.7% among all classification algorithms.

SVM outperforms LR, KNN, NB, and ANN on security and performance datasets due to their ability to handle both linear and nonlinear data effectively. SVMs are robust to outliers and noisy data; they can identify the most critical instances (bugs) and create a well-defined decision boundary, minimizing the impact of outliers on the model's performance. SVM is a binary classifier by default, which is well-suited for security and performance bug datasets. SVM aims to maximize the margin between classes, which leads to a better separation of data points. This margin maximization can help distinguish between security bugs and performance bugs more effectively. Further tuning hyper parameters such as selecting the right kernel function and balancing values of other parameters have a significant impact on SVM's performance. SVMs can leverage well-engineered features to improve their ability to discriminate between security and performance bugs. Imbalanced datasets can lead to biased classifiers, but SVMs can handle balanced datasets effectively. The *F*1 score, which balances precision and recall, is a suitable evaluation metric for security

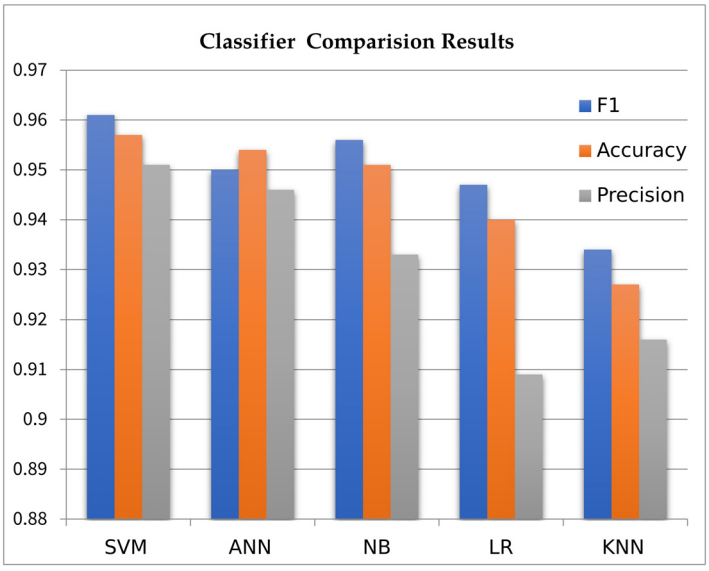


Figure 11: Comparison of the algorithms.

bug data, especially when both false positives and false negatives have significant implications. SVMs can achieve a high *F*1 score by striking the right balance between identifying true security bugs (recall) and minimizing false alarms (precision). Thus, it is evident from our research that there is a significant impact on identification accuracy of security and performance bugs through any classification algorithms, i.e., SVM, ANN, LR, KNN, and NB.

5.1 Result validation

Validating the results of a text classification algorithm involves assessing its performance to ensure that it generalizes well to unseen data. Here are some common techniques for validating text classification algorithms: Error analysis, evaluation metrics, stratified sampling, validation set, cross-domain validation, and train-test split.

Instead of using a single train-test split, we employed cross-validation methods such as k-fold cross-validation. To do this, the dataset is divided into k subsets. The model is then trained on various combinations of $k - 1$ subsets k-times, and its performance is assessed on the remaining subset on each evaluation. As a result, the model's performance estimate is more reliable. Ten-fold cross-validation was employed [18]. The dataset is divided into ten distinct subsections by the cross-validation test, with nine of them being used for training and the tenth one for testing. After ten iterations of the algorithm, the average accuracy over all folds is determined [3,4].

To ensure that classification models are neither under fitting nor over fitting generally used these strategies, including Train-validation-test split, cross-validation, learning curves, regularization, model complexity, validation set performance and early stopping. We used train and test spilt, parameter tuning [18], and cross-validation techniques, such as k-fold cross-validation, to train and evaluate the model on multiple subsets of the data. Our models perform consistently well across different folds, it is less likely to be overfitting [1,21].

5.2 Threat to validity

There are certain risks associated with this research that could compromise the reliability of the findings and inferences made. In an attempt to improve classification performance, we experimented with various model

parameter combinations and algorithms. Experimenting with every classification algorithm and hyper parameter combination is not feasible, though.

Second, we meticulously adhered to the original papers in order to implement the baselines. We have not worked on the source code of the dataset of the old paper [3] because these related comparison works do not provide the source codes of their works. We have worked on a significant amount of source code and made every effort to ensure that the implementation is accurate. Open-source software project data are also utilized for validation, in addition to the Bugzilla datasets. Our suggested method may perform better or worse depending on whether it is applied to different kinds of software projects created in various environments. In addition, the fact that we only used 1,000 instances of the dataset for testing and training poses another risk, potentially leading to variations in the current results.

6 Conclusion and future work

This research compared and analyzed the impact of classification algorithms, i.e., SVM, ANN, LR, KNN, and NB on identification accuracy of security and performance bugs. We have applied SVM, ANN, NB, KNN, and LR on the Bugzilla bug repository to classify security and performance bugs. We analyzed and found out the best algorithm. The best algorithm in the proposed methodology is the SVM with 0.96% *F*1 measure. This research indicates that SVM has higher prediction accuracy after applying the pre-processing step, TF-IDF. Two algorithms SVM and LR perform better in terms of precision (0.99%) for performance bugs and three algorithms SVM, ANN, and LR perform better in terms of *F*1 score for security bugs. The main contribution of this research includes the creation of a new data set of 1,000 instances, identification and comparative analysis of the classification of security and performance bugs from the Bugzilla repository. It was evident from our research that there is a significant impact on the identification accuracy of security and performance bugs through any classification algorithms, i.e., SVM, ANN, LR, KNN, and NB.

In future, we plan to extend our comparative analysis among deep neural networks by extending our dataset of security and performance bugs. Besides, we may improve the "precision-recall trade-off" issue where the precision rate is high while the *F*1 score is low, through resampling techniques, threshold adjustment, etc.

Acknowledgements: The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number "NBU-FFR-2024-1580-05."

Funding information: The research is funded by the Deanship of Scientific Research at Northern Border University, Arar, KSA.

Author contributions: Inception of research: M.S. and R.B.F.; data curation: M.S. and R.B.F; gap analysis: M.S. and R.B.F.; funding acquisition: M.A.; results and analysis: M.S. and R.B.F.; proposed methodology: M.S.; experimental setup: M.S.; result validation: R.B.F. and M.A.; writing – original draft: M.S.; review and editing: R.B.F and M.A. All authors have read and agreed to the published version of the manuscript.

Conflict of interest: The authors declare no conflict of interest.

Data availability statement: The data repository used for experimentation is Bugzilla 2019 which is publicly available at https://bugzilla-dev.allizom.org/home, https://bugzilladev.allizom.org/buglist.cgi?quicksearch=performance, https://bugzilla-dev.allizom.org/buglist.cgi?quicksearch=security.

References

[1] Choudhary P. Neural network based bug priority prediction model using text classification techniques. Int J Adv Res Computer Sci. 2017;8:1315–9. doi: 10.26483/ijarcs.v8i5.3559.

[2] Immaculate SD, Begam MF, Floramary M. Software bug prediction using supervised machine learning algorithms. In 2019 International Conference on Data Science and Communication (IconDSC). IEEE; 2019 Mar 1. p. 1–7. doi: 10.1109/IconDSC.2019.8816965.

[3] Nagwani NK, Verma S. A comparative study of bug classification algorithms. Int J Softw Eng Knowl Eng. 2014;24:111–38. doi: 10.1142/S0218194014500053.

[4] Ezami S. Extracting non-functional requirements from unstructured text. Master's thesis. University of Waterloo; 2018.

[5] Kukkar A, Mohana R. A supervised bug report classification with incorporate and textual field knowledge. Procedia Computer Sci. 2018;132:352–61. doi: 10.1016/j.procs.2018.05.194.

[6] Naik K, Tripathy P. Software testing and quality assurance: theory and practice. Hoboken, NJ, United States: John Wiley & Sons; 2011.

[7] Rashwan A, Ormandjieva O, Witte R. Ontology-based classification of non-functional requirements in software specifications: A new corpus and SVM-based classifier. In 2013 IEEE 37th Annual Computer Software and Applications Conference. IEEE; 2013 Jul 22. p. 381–6. doi: 10.1109/COMPSAC.2013.64.

[8] Zhou Y, Tong Y, Gu R, Gall H. Combining text mining and data mining for bug report classification. J Software Evol Process. 2016;28(3):150–76. doi: 10.1002/smr.1770.

[9] Allahyari M, Pouriyeh S, Assefi M, Safaei S, Trippe ED, Gutierrez JB, et al. A brief survey of text mining: Classification, clustering and extraction techniques. arXiv preprint arXiv:1707.02919. 2017 Jul 10. doi: 10.48550/arXiv.1707.02919.

[10] Cleland-Huang J, Settimi R, Zou X, Solc P. The detection and classification of non-functional requirements with application to early aspects. In 14th IEEE International Requirements Engineering Conference (RE'06). IEEE; 2006. p. 39–48. doi: 10.1109/RE.2006.65.

[11] Anandarajan M, Hill C, Nolan T. Practical text analytics. Maximizing the Value of Text Data. (Advances in Analytics and Data Science. Vol. 2.). Switzerland AG: Springer Nature; 2019. p. 45–59. doi: 10.1007/978-3-319-95663-3.

[12] Patil S. Concept-based classification of software defect reports. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE; 2017 May 20. p. 182–6. doi: 10.1109/MSR.2017.20.

[13] Alqahtani SS. Security bug reports classification using fasttext. Int J Inf Secur. 2024;23(2):1347–58. doi: 10.1007/s10207-023-00793-w.

[14] Yeasmin S. Analysis and Interactive Visualization of Software Bug Reports. PhD diss. University of Saskatchewan; 2014. Corpus ID: 28792891.

[15] Allahyari M, Pouriyeh S, Assefi M, Safaei S, Trippe ED, Gutierrez JB, et al. A brief survey of text mining: Classification, clustering and extraction techniques. arXiv preprint arXiv:1707.02919; 2017. doi: 10.1145/3357729.3357740.

[16] Stamp M. Introduction to machine learning with applications in information security. New York: Chapman and Hall/CRC; 2022.

[17] Nikam SS. A comparative study of classification techniques in data mining algorithms. Orient J Computer Sci Technol. 2015;8(1):13–9.

[18] Russell SJ, Norvig P. Artificial intelligence a modern approach. London: Pearson; 2016.

[19] Pratama BY, Sarno R. Personality classification based on Twitter text using Naive Bayes, KNN and SVM. In 2015 international conference on data and software engineering (ICoDSE). IEEE; 2015 Nov 25. p. 170–4. doi: 10.1109/ICODSE.2015.7436992.

[20] Behl D, Handa S, Arora A. A bug mining tool to identify and analyze security bugs using Naive Bayes and TF-IDF. 2014 International Conference on Reliability Optimization and Information Technology (ICROIT); 2014. p. 294–9. doi: 10.1109/ICROIT.2014.6798341.

[21] Aggarwal CC, Zhai C. A survey of text classification algorithms in mining text data, New York, NY, USA: Springer; 2012. p. 163–222. doi: 10.1007/978-1-4614-3223-4_6.

[22] Gajjala A. Multi faceted text classification using supervised machine learning models. Master's thesis, San José State University, San Jose, CA, USA, 2016; p. 482.

[23] Kowsari K, Jafari Meimandi K, Heidarysafa M, Mendu S, Barnes L, Brown D. Text classification algorithms: A survey. Information. 2019 Apr;10(4):150. doi: 10.3390/info10040150.

[24] Imseis J, Nachuma C, Arifuzzaman S, Zibran M, Bhuiyan ZA. On the assessment of security and performance bugs in chromium open-source project. In Dependability in Sensor, Cloud, and Big Data Systems and Applications: 5th International Conference, DependSys 2019, Guangzhou, China, November 12–15, 2019, Proceedings 5. Springer Singapore; 2019. p. 145–57.

[25] Zou D, Deng Z, Li Z, Jin H. Automatically identifying security bug reports via multitype features analysis. In Information Security and Privacy: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings 23. p. 619–33. doi: 10.1007/978-3-319-93638-3_35.

[26] Otoom AF, Al-jdaeh S, Hammad M. Automated classification of software bug reports. In Proceedings of the 9th International Conference on Information Communication and Management; 2019 Aug 23. p. 17–21. doi: 10.1145/3357419.3357424.

[27] Khleel NA, Nehéz K. Comprehensive study on machine learning techniques for software bug prediction. Int J Adv Computer Sci Appl. 2021;12(8):726–35. doi: 10.14569/IJACSA.2021.0120884.

[28] Gegick M, Rotella P, Xie T. Identifying security bug reports via text mining: An industrial case study. 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa; 2010. p. 11–20. doi: 10.1109/MSR.2010.5463340.

[29] Zaman S, Adams B, Hassan A. A qualitative study on performance bugs. 2012 9th IEEE Working Conference on Mining Software Repositories (MSR); 2012. p. 199–208. doi: 10.1109/MSR.2012.6224281.

[30] Goseva-Popstojanova K, Tyo J. Identification of security related bug reports via text mining using supervised and unsupervised classification. In 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE; 2018. p. 344–55. doi: 10.1109/QRS.2018.00047.
[31] Zheng W, Xun Y, Wu X, Deng Z, Chen X, Sui Y. A comparative study of class rebalancing methods for security bug report classification. IEEE Trans Reliab. 2021;70(4):1658–70. doi: 10.1109/TR.2021.3118026.
[32] Pal S. An integration of clustering and classification technique in software error detection. Afr J Comput ICT. 2015;8(2):9–16.
[33] Odera D, Odiaga G. A comparative analysis of recurrent neural network and support vector machine for binary classification of spam short message service. World J Adv Eng Technol Sci. 2023;9(1):127–52. doi: 10.30574/wjaets.2023.9.1.0142.
[34] Aljedaani W, Mkaouer MW, Ludi S, Ouni A, Jenhani I. On the identification of accessibility bug reports in open source systems. In Proceedings of the 19th International Web for all Conference; 2022 Apr 25. p. 1–11. doi: 10.1145/3493612.3520471.
[35] Zhou C, Li B, Sun X, Guo H. Recognizing software bug-specific named entity in software bug repository. In Proceedings of the 26th Conference on Program Comprehension; 2018 May 28. p. 108–19. doi: 10.1145/3196321.3196335.
[36] Cavnar WB, Trenkle JM. N-gram-based text categorization. In Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. 161175, 1994 Apr 11. p. 14.
[37] Sureka A, Jalote P. Detecting duplicate bug report using character n-gram-based features. In 2010 Asia Pacific Software Engineering Conference. IEEE; 2010. p. 366–74. doi: 10.1109/APSEC.2010.49.