Automated Software Engineering (2024) 31:35
https://doi.org/10.1007/s10515-024-00432-1

Bug reports priority classification models. Replication study

Andreea Galbin-Nasui[1] · Andreea Vescan[1]

Received: 6 May 2023 / Accepted: 13 March 2024 / Published online: 10 April 2024

Abstract
Bug tracking systems receive a large number of bugs on a daily basis. The process of maintaining the integrity of the software and producing high-quality software is challenging. The bug-sorting process is usually a manual task that can lead to human errors and be time-consuming. The purpose of this research is twofold: first, to conduct a literature review on the bug report priority classification approaches, and second, to replicate existing approaches with various classifiers to extract new insights about the priority classification approaches. We used a Systematic Literature Review methodology to identify the most relevant existing approaches related to the bug report priority classification problem. Furthermore, we conducted a replication study on three classifiers: Naive Bayes (NB), Support Vector Machines (SVM), and Convolutional Neural Network (CNN). Two sets of experiments are performed: first, our own NLTK implementation based on NB and CNN, and second, based on Weka implementation for NB, SVM, and CNN. The dataset used consists of several Eclipse projects and one project related to database systems. The obtained results are better for the bug priority P3 for the CNN classifier, and overall the quality relation between the three classifiers is preserved as in the original studies. The replication study confirmed the findings of the original studies, emphasizing the need to further investigate the relationship between the characteristics of the projects used as training and those used as testing.

Keywords Bug priority prediction · Bug report · CNN · SVM · BN

✉ Andreea Vescan
andreea.vescan@ubbcluj.ro

Andreea Galbin-Nasui
andreeagalbin13@gmail.com

[1] Computer Science Department, Babes-Bolyai University, M. Kogalniceanu 1, 400084 Cluj-Napoca, Cluj, Romania

1 Introduction

As the software engineering process converges to large dimensions and develops hundreds of new features, the product becomes more complex, and regardless of the workload and quality of product creation, it may still have defects or improper implementations that lead to production bugs. Here, software testing processes play an important role in the process of implementing quality products.

A software system may receive a large number of bugs on a daily basis, and to keep track of them, most software development teams adopt a bug tracking system (Bugzilla 2023) (e.g., Bugzilla). When a bug is found, the end user or the developer usually writes a bug report to keep track of the issue.

One of the questions when sorting bug reports is how to establish the order in which to fix them. This is tracked in the field "Priority" of each bug report and can have values from P1 to P5. Unfortunately, although there are some guidelines on how to assign the correct priority of a bug report, the process is manual and depends on the expertise of the bug report author.

This approach uses a method to combine text processing with a CNN-based approach to assign the correct priority label to each bug report. The hypothesis is that each description of the bug report has an important impact on the priority classification.

The purpose of the current paper is two-fold: to provide a literature review on the bug report priority classification approaches, and to conduct a replication study using various classifiers for the prioritization of bug reports problem based on processed text using natural language processing (NLTK 2023). Moreover, there is a link between the two investigations: the outcome of the SLR is used to select the state-of-the-art study and then, in the second stage, to conduct a replication study to confirm the results of the original study and to expand the knowledge about the replicated method.

The contributions of this paper are the following.

- A systematic review of the existing approaches for automating the prediction of priority of bug reports.
- Combining and selecting text processing using the NLTK and various classifiers to assign the correct priority label to a bug.
- Apply the classifiers to a dataset using cross-project validation.

The results of our investigation regarding the SLR revealed that there are a vast number of approaches for the automatic bug report priority assignment using machine learning algorithms like CNN, NB, SVM, and others, with various incorporated features, even an "emotion" perspective. Referring to the results of the experiments performed, our findings confirmed the previous results, both CNN and NB obtaining good results.

The paper is organized as follows: In Sect. 2, the background concepts related to bug reporting and text processing are explained. Section 3 summarizes a review of the systematic literature on related work, also discussing original studies to replicate and the replication methodology in software engineering. In Sect. 5, we elaborate on the research design, detailing the experimental design and objects, along with used metics. The experiments and their results are described in Sect. 6. We discuss the threads to validity in Sect. 7 and conclude the results in Sect. 8.

2 Background on concepts

This section outlines several concepts with which we operate in the paper sections, concepts such as error or bug, bug report, priority of a bug report, and Convolutional Neural Networks.

2.1 The notion of error or bug

A bug is an error or malfunction of a program that produces unwanted or incorrect results. This prevents the application from working as it should (Myers 2005). These are often discovered after a product is launched or during public testing. When this happens, users need to find a way to avoid using the faulty code or get a patch from the software developers, which is related to fixing the found bug.

2.2 Bug report

The activity of a software tester is not simple; it must pay attention to several aspects: “designing the testing process (preparing test cases and a test plan), testing the application itself, and reporting any bugs” (Myers 2005).

The bug reporting process is complex, so a bug tracking system is the best tool in this regard. There are five steps in tracking bugs:

1. *Bug detection*: The test team detects bugs. They can also be detected and reported by end users.
2. *Bug reporting*: The tester identifies the existing bug and reports it using a bug tracking system.
3. *Bug fixes*: The developers are trying to fix some bug fixes.
4. *Retest*: Software is repeatedly tested to make sure that a bug does not exist.
5. *Data capture*: all bug data are recorded in the bug report to avoid the same occurrence in the future (Patton 2000).

There are certain rules when writing a bug report. Most of them refer to the way we formulate the reported problem, but also to how attentive we are to the small details related to the product. The most important of these are the following.

- assign a unique number to each defect, which will later help us identify it. If we use an automatic bug reporting system, it will automatically generate.
- Clear, chronological, and concise description of the reproduction steps.
- Be as specific as possible in the description.

- It is important to ensure that all the details about the failure are correct (Patton 2000).

It is of particular importance that a "bug report" be written, so that it has a high probability of being resolved as efficiently as possible. These fields include:

1. Title—It is advisable to be concise and possibly to mention the component in which we discovered the bug.
2. Severity/priority—These refer to how severely the defect found affects the proper functioning of the program, and the more severe the failure, the faster the bug needs to be fixed.
3. The description is an overview of the bug and how or when it happened. This part should include more details than the title and be quite explicit.
4. Work environment—Applications may behave completely differently depending on their environment. This part should include all the details about the environment configuration and the configuration on which the application is running.
5. Reproduction steps—This should include the minimum steps required to reproduce the bug. Ideally, the steps should be short and simple and can be followed by anyone. The goal is to allow the developer to reproduce the bug so that they can better see where the bug originated.
6. The result obtained.
7. The desired result (Myers 2005).

2.3 Bug priority

The priority of the bug is key to deciding which bug should be resolved first (Patton 2000). When testers submit a bug report, they can select the priority level based on their background knowledge. A P1 priority usually means that this bug is a blocker and prevents the software from working properly. Meanwhile, priority P5 represents a minor issue; these types of issue can remain unresolved for months. In Bugzilla (2023), the meaning of priority is the importance of the bug, decided by the developers.

Priority is defined as the order in which a defect should be fixed. The higher the priority, the sooner the defect should be resolved. This label determined the order in which the developer should resolve a defect. There is often a confusion between Priority and Severity, the difference between them being that Severity is the degree of impact that a defect has on the operation of the product. Based on its label from 1 to 5, we can prioritize the bug reports into five types: critical, major, moderate, minor, and cosmetic.

Intuitively, a critical priority defect affects a large number of system features, a high priority defect affects a smaller subset of functionality, a medium priority defect affects an individual functionality, and a low priority defect is considered minor irritating (Patton 2000), must constantly monitor the status of bugs, to ensure that it agrees with any changes made to it and to provide additional test data (Myers 2005).

With the process mentioned above, the bugs go through 3 important states, “open state, resolved state, and closed state” (Patton 2000). Fig. 1 contains the states “open”, “fixed” and “closed”. The bug-sorting process begins with the user or tester noticing a discrepancy between how the product should work and how it actually works and the allocation of the bug to the programmer so that it can be fixed. Here comes the accuracy with which the bug was reported and the importance of establishing the most relevant value of the fields in the report.

Figure 1 represents the stages that a bug goes through. Each step is described in the following.

Each element in Fig. 1 plays an important role in the bug-sorting process.

- *New*: When the tester finds a bug or defect in the testing phase, it is reported to the development team through the bug handling tools. Therefore, the tester assigns the initial bug status as “NEW”.
- *Assigned*: Once the tester has reported a new defect, the technique leads to its validation and assignment to a specific programmer to work on it. Then the bug status is marked “ASSIGNED”. This step consists of designating a specific individual or developer to take the responsibility for addressing and resolving a reported software issue or defect. Once a bug is assigned, the designated person is tasked with investigating, fixing, and testing the identified problem to ensure its proper resolution within the software system. This assignment process helps streamline bug resolution by assigning responsibilities and facilitating a structured approach to debugging and development tasks.
- *Open*: At this stage, once the tester assigns the bug to the developer, it investigates and fixes it. The bug or fault status is displayed as “OPEN”. The repair of bugs begins at this stage.

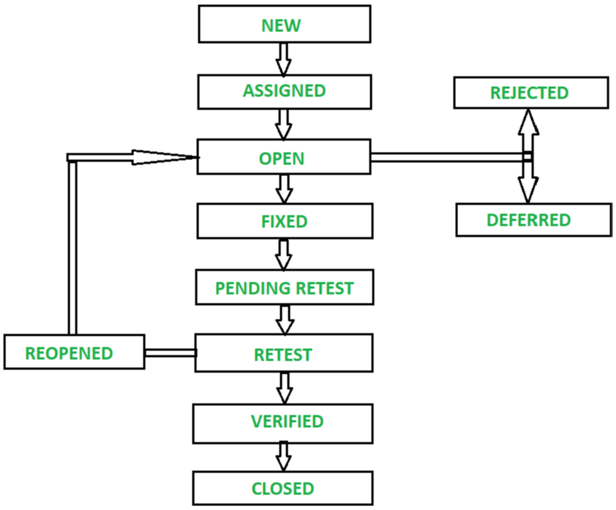
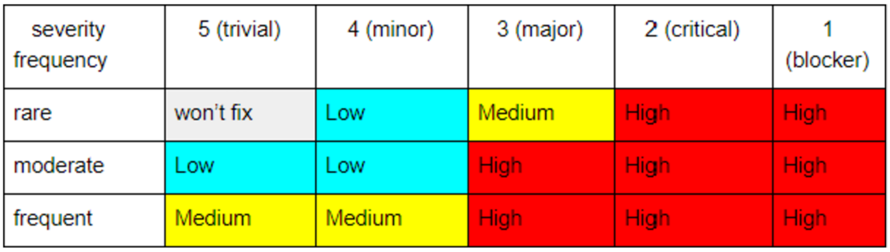


Fig. 1 Development stages of a bug (Patton 2000)

- *Resolved*: When the programmer takes the necessary actions regarding the changes to fix the defect, it changes the status of a bug as "FIXED".
- *Verified*: The developer fixes the bug and then the application is tested again. If there is no bug, then change the status to "VERIFIED".
- *Reopened*: The tester will re-examine the application after bug fixes by the developer, while there is an bug (either a new bug or an old bug is not fixed), then it has assigned the status of "REOPENED" means that it will go through the bug life cycle again.
- *Closed*: If there are no problems or issues in the application after testing, then the tester will be assigned as "CLOSED" by the tester (Myers 2005).

In the process of creating a new bug report, the impact of the problem encountered and the concise identification of its severity will be taken into account. The method by which this process is carried out is to assign a "bug report" priority. This priority indicates the importance of the bug and some hierarchy to fix it. Figure 2 represents the stages by which a priority level can be marked as "blocker" or "minor" Myers (2005). Each step is described in the following. There is a common list of priority classifications, including five categories.

- *P1*: Represents the highest risk threshold, meaning that the bug is causing system problems and needs to be fixed as soon as we call this priority "critical".
- *P2*: This is a bug that should be fixed before launching the product as soon as critical bugs have been fixed.
- *P3*: This bug should only be fixed after serious bugs have been fixed. Represents a medium priority fence.
- *P4*: This defect can be fixed in the future and does not need immediate attention, and low-severity defects fall into this category. The bugs reported here do not greatly affect the operation of the software.
- *P5*: This priority level indicates a bug of the lowest severity. Here, we can classify certain design, font, or even spelling mistakes (Myers 2005).



severity frequency	5 (trivial)	4 (minor)	3 (major)	2 (critical)	1 (blocker)
rare	won't fix	Low	Medium	High	High
moderate	Low	Low	High	High	High
frequent	Medium	Medium	High	High	High

Fig. 2 The impact of a bug priority on the software (Myers 2005)

3 Systematic literature review method: exploring relevant studies on bug priority

A systematic search and analysis were performed according to the problem studied using a general method of dividing the question into individual parts and compiling a list of alternative synonyms and spellings, as in the Kitchenham and Charters (2007) report.

Systematic literature review process. We performed a Systematic Literature Review (SLR) using Kitchenham's approach (Kitchenham and Charters 2007). In this SLR we analyze using software references and studies conducted in the direction of improving the bug sorting process. We performed and documented this Systematic Literature Review in the thesis of the same name (Galbîn-Năsui 2022).

The stages of the conducted SLR are emphasized in Fig. 3. In the planning phase, the motivation to carry out the investigation is provided along with the research questions, and the rules for searching and selecting the studies are provided. In the next phase, the studies are collected using the rules defined in the previous phase and the data synthesis is performed. In the last phase, the reporting on findings, the answering research questions, and final results discussions are performed.

3.1 SLR planning

3.1.1 Review need identification

The aim of our work is to explore how bug priority is approached in previous studies, and whether the emotion value of the bug report is considered. The objective of

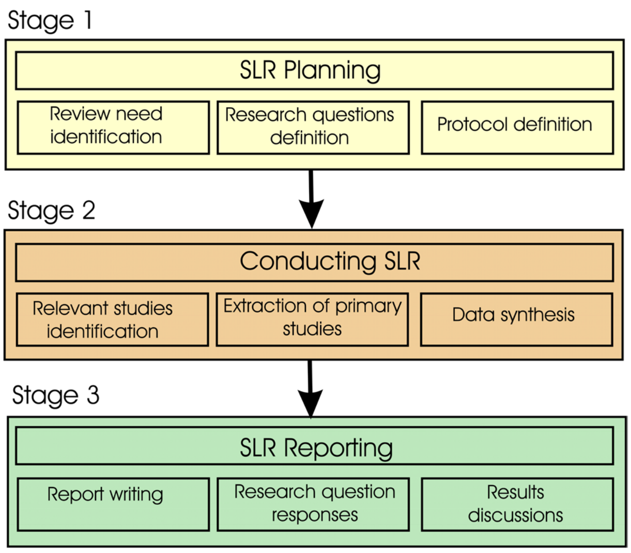

Stage 1
SLR Planning
Review need identification
Research questions definition
Protocol definition
Stage 2
Conducting SLR
Relevant studies identification
Extraction of primary studies
Data synthesis
Stage 3
SLR Reporting
Report writing
Research question responses
Results discussions

Fig. 3 SLR process overview

the SLR is to identify existing research on bug priority classification and to summarize the findings. The motivation that drives us to conduct this SLR is given by the following reasons: (1) to characterize the state-of-the-art to identify and understand the ongoing scientific research on bug reports priority classification, and (2) to position our work in the current research.

3.1.2 Research questions definition

To address the goal of the SLR, the following questions are defined:

1. RQ1. What are the most relevant research papers in this field?
2. RQ2. How should we approach this issue in software engineering and how has it been addressed in the past?
3. RQ3. What are the most important methods studied in this context?

3.1.3 Protocol definition

The steps and rules for conducting the SLR are defined in this section. The steps are: (1) search terms and resources for the search of the primary studies, (2) the inclusion/exclusion criteria for the selected studies, (3) the strategy for data extraction, (4) the data synthesis, and (5) reporting the results of the investigation.

3.2 Conducting SLR

3.2.1 Search and selection process

A systematic search and analysis were performed according to the problem studied using a general method of dividing the question into individual parts and generating a list of alternative synonyms and spellings. These terms could be used by taking into account the titles of the topics used in articles and publications.

Initially, six keywords were defined that were relevant to the problem studied: *bug, report, priority, severity, prediction, testing*.

The criteria for the inclusion of articles are delineated as follows: to be considered, articles must feature identified keywords either in the title, the dedicated keywords section, or in the abstract of the paper. Additionally, the articles must be published in an international scientific database. In essence, these criteria serve as a filtering mechanism to ensure that the selected articles are directly relevant to the defined problem and are accessible through widely recognized scientific databases.

We include articles from the following databases:

- https://ieeexplore.ieee.org/Xplore/home.jsp
- https://dl.acm.org/
- https://www.springer.com/gp

3.2.2 Collecting the studies

Numerous articles retrieved from previously specified databases, using the provided keywords, were selected and saved based on their titles and keywords. Subsequently, from the pool of identified articles, 30 were chosen. Among these, we meticulously selected 10 particularly relevant studies for incorporation into the latest research. The selection process was primarily based on the information presented in the respective abstracts.

3.2.3 Data synthesis

In a Systematic Literature Review (SLR), data synthesis refers to the process of systematically analyzing and combining information from selected studies to draw meaningful conclusions or identify patterns in the literature. The goal of data synthesis is to provide a comprehensive and evidence-based understanding of the research topic under investigation. The narrative synthesis technique is used to explain and interpret the results found in the analysis. The results of this activity are presented in the next section.

3.3 SLR reporting

3.3.1 Report of the systematic literature review

This section contains the analysis and findings of the SLR conducted. The ten articles selected following the first selection procedure have in common the study of the same issue but using different methods of prediction and data processing. Although studies such as Umer et al. (2018), Umer et al. (2020), Yu et al. (2010) proposed implementing a machine learning-oriented model, using the SVM Umer et al. (2018), CNN Umer et al. (2020) and ANN Tian et al. (2013) methods. Some of the authors proposed using probablistic classifiers such as Naive Bayes Alenezi et al. (2013), and the nonparametric supervised learning method Nearest Neighbor Ramay et al. (2019).

In Umer et al. (2018), the authors proposed a method using SVM (Support Vector Machine) in a set of bug reports from the Eclipse project. The use of an SVM has been shown to scale relatively well to large data set sizes, complexity can also be explicitly controlled, and flexible threshold can be applied during selection. This method shows an improvement over P3 (Drone) in the F1 score of up to 6.10%.

The paper (Ramay et al. 2019) investigated the use of a Deep Neural Network to determine the severity of a bug by analyzing its textual description and processing it using NLTK (Natural Language Processing). This study motivates researchers to implement a more efficient automated process for identifying bugs. They also investigated whether a longer text included in a bug report would result in a better prediction.

One of the most relevant studies in this field is Tian et al. (2013), this one uses linear regression and has similar results to those of the latest study on Galbîn-Năsui (2020). In Ramay et al. (2019) there were significant improvements in the results, using "Deep Neural Network-Based Severity Prediction of Bug Reports".

To automatically predict the severity of bug reports, a new approach using Nearest Neighbor Method information, especially the BM25-based document similarity function, has been proposed in Tian et al. (2012). Their approach automatically analyzes past bug reports from Bugzilla for Eclipse, OpenOffice, and Mozilla. Compared to the current state-of-the-art severity prediction study, the proposed approach is significantly improved. Unlike SEVERIS, which has taken a similar approach, this method shows an improvement in F1 for critical, minor, and trivial scores of 41%, 94%, and 178%, although for the major score it shows a higher result. small by 2%.

In the research outline in Umer et al. (2020), the authors suggest using a CNN (convolutional neural network) with the help of NLTK to determine the priority of a bug. Perform engineering-specific bug reporting on bug reports and calculate the emotion value for each of them using specific software. Datasets from the most popular Bugzilla, Jira, and Github software systems have been selected as datasets. For the classifier, the generated vector and the emotion from the report of each bug are used. This approach shows a bug reduction of up to 2.5% compared to Drone Tian et al. (2013).

The study determined by the authors in Alenezi et al. (2013) includes an approach to predict the priority of a reported bug using 3 different machine learning algorithms: Naive Bayes, Decision Trees, and Random Forest. The experiment is conducted using bug reports for the Eclipse and Firefox projects in a number of more than 65,000 reports. As a result of this study, the authors demonstrated that both the Random Forest and Decision Tree outperformed the Naive Bayes. To confirm the results of this study, only reports with the status of: RESOLVED, CLOSED, or VERIFIED were selected.

The authors of the study (Sharma et al. 2015) evaluated the performance of various machine learning techniques to predict the priority of a bug. They use the bug reporting feature to predict the priority of new bugs. They also used a series of Rapid Miner operators to preprocess reports, such as removing stop words (which would be irrelevant to use in analysis) and tokenizing. The evaluation of the model was performed by applying project validation for 76 cases from five Open Office and Eclipse data sets in Bugzilla.

The technique of using an artificial neural network (ANN) was used by Yu et al. (2010) to predict the priorities of the defects. They improved troubleshooting efficiency by proposing a technique that uses neural networks to predict bug priorities and then adopt evolutionary data set training. Experiments were conducted with five different software products of an international healthcare company to demonstrate feasibility and effectiveness. Compared to the Bayes algorithm, the ANN model showed better qualification in terms of recall, accuracy, and the F1 measure. The comparison was made with RIS2 software, and the project included more than 2000 sample bug reports.

The paper (Uddin et al. 2017) presents a theoretical study of bug reports and the motivation for the need to work on prioritizing bugs. Existing work on bug prioritization and some possible issues with working with automatic prediction of bug priority are included here.

All these approaches in software engineering have in common a classification of a software bug based on the textual analysis of the report, on the basis of which certain characteristics will be extracted in order to determine the priority or severity of the respective bug. Many researchers have focused on automating the sorting of bugs using machine learning methods. The main challenge with these traditional supervised machine learning methods is that they require a large amount of tagged data for classifier training.

A final classification of the selected papers based on the used approach, dataset and tools is provided in Table 1.

In what follows, we outline several advantages and disadvantages of the above investigated studies.

The systematic literature review unveils several key advantages regarding bug report priority classification. Firstly, there is a large diversity in the used methodologies, from machine learning models such as SVM, CNN, DNN to the Nearest Neighbor method. Specific studies, such as the one utilizing SVM, showcase improvements in accuracy, demonstrating a 6.10% enhancement in the F1 score compared to alternative methods. The integration of NLP tools like NLTK for efficient textual analysis emerges as a promising trend, particularly evident in a Deep Neural Network approach. Moreover, the practical application of these models to real-world bug reports ensures relevance and diversity in testing contexts. Lastly, comparative studies that investigated strengths and weaknesses of various bug reports priority classifiers contribute to a better understanding of the methods.

Regarding disadvantages, some may be stated. First, a clear separation between severity and priority, more specifically, particular cases where high priority may not necessarily correspond to severe issues. The dependence on labeled data for traditional supervised machine learning methods is another drawback. Additionally, there is a limitation regarding generalization of findings since the model's results may vary based on

Table 1 Related work on bug priority

Approach	Studies	Dataset	Tool
SVM	Umer et al. (2018)	Eclipse project	NLP, TextBlob, NLTK
CNN	Umer et al. (2020)		
ANN	Tian et al. (2013), Yu et al. (2010)		
Naive Bayes, Decision Trees, Random Forest	Alenezi et al. (2013)	Eclipse project	SentiWordNet, NLTK
Deep Neural Network, SVM	Ramay et al. (2019)	Eclipse project	NLTK, LSTM
Nearest Neighbor	Tian et al. (2012)		
Deep Neural Network	Sharma et al. (2015)	Open Office, Eclipse and Mozilla	Senti4SD, NLTK
SVM, NB, KNN	Uddin et al. (2017)	Eclipse projects	NLTK

the dataset characteristics. However, in this aspect, the text length in bug reports may have an impact on the accuracy of the classification.

3.3.2 Answers to the SLR research questions

Following the SLR study, we can now state the answers to the research questions set at the beginning of this article.

RQ1. What are the most relevant research papers in this field?

Comparing the results obtained from DRONE framework and the other relevant studies in the field it may be seen that the Tian et al. (2013) approach outperforms the baselines in terms of average F-measure by a relative improvement of 58.61% which is one of the best results on this study. Another effective approach is the one studied in the article (Umer et al. 2018) where Deep learning techniques are used to increase the efficiency of the algorithm, unlike Drone, which so far uses classical classifiers such as linear regression. Proper selection of features is the basis for the classification task, so there are studies that included other elements of the report in the data set, such as the “emotion” factor Umer et al. (2018), Tian et al. (2012), improvement of the F1 score ranges from 2.04% to 11.59%.

RQ2. How should we approach this issue in software engineering and how has it been addressed in the past?

First, the problem referring to automatically prioritize a bug report is approached in software engineering by defining it as a classification problem: based on the textual analysis of a report with various characteristics, the priority of the respective bug is established. Second, this problem has been addressed by many researchers who focused on automating the assignment of bug priority using machine learning methods Yu et al. (2010), Ramay et al. (2019), Umer et al. (2020). Various features, incorporating even sentiment analysis results, were used in previous approaches with different tools and libraries. The main challenge with these traditional supervised machine learning methods is that they require a large amount of data tagged for classifier training.

RQ3. What are the main methods studied in this context?

The main methods that investigated this bug reports priority classification problem are SVM, ANN, CNN, Naive Bayes, Random Forest, Deep Neural Networks as seen in Table 1, most of them using the NLP technique with similar features. There are other studies that also incorporated emotion-based features (computation of an emotion value for each bug report) to classify bugs Umer et al. (2018), Umer et al. (2020) and also used machine learning techniques to classify a “bug report” depending on the description of the bug and other features such as author, product, and time factors.

4 Current explorations in bug report priority: investigating modern classification models

4.1 The original study

We conducted a replication study of the approaches in Umer et al. (2018) and Umer et al. (2020) regarding bug reports priority classification.

Umer et al. (2018) proposed a SVM-based bug priority classification model. They extracted and stored the Eclipse bug reports (Eclipse-bugs 2023) from Bugzilla (2023) and adopted (NLTK 2023) text preprocessing models to clean up the data. Prioritization of bug reports was carried out by analyzing the summary and calculation of emotion value using (SentiWordNet 2023).

Umer et al. (2020) proposed a CNN-based automatic prioritization of Bug Reports. They also applied NLTK natural language processing methods (NLTK 2023) to preprocess the textual information from the bug reports. From the pre-processed bug reports they performed an emotional analysis and constructed a vector for each bug using the Word2Vec model (Word2Vec 2023).

4.2 Replication in software engineering

Replication is an essential method that attempts to validate the findings of previous experiments and research.

Several aspects that are relevant for a replication study have been outlined by Shepperd et al. (2018). First, the authors must state the original experiment that is being replicated and also include experiments that allow extension of external validity. Another important aspect is that both of the research have to contain common research questions and enclose comparisons between the 2 sets of results. Confirming or, on the contrary, disconfirming the original experiment.

Carver (2010) and Carver et al. (2014), provided four elements to be comprised in a replication report. These are (1) information about the original study has to provide enough context for understanding the replication in question; (2) information about the replication to help readers understand specific important details about replication itself; (3) comparison of replication results with original study results in commonalities and differences in the results obtained, and (4) conclusions across studies to provide readers with important insights that can be drawn from the series of studies that may not be obvious from a single study.

Other dimensions of replication were pronounced by Gómez et al. (2014) expressing: operationalization, population, protocol, and experiments. On top of that, Fagerholm et al. (2019) advises that replication designs should be meticulously documented.

The Operationalization dimension describes the act of translating a construct into its manifestation. the constructs are operationalized into treatments and indicate how similar the replication is to the baseline experiment. The effect constructs are translated into metrics and measurement procedures.

The Population dimension is related to the experimental objects. This implies exploring the boundaries of the properties of the experimental objects for which the results hold.

The Protocol dimension is related to the factors used in the experimental analysis that can differ in each replication. These differences could be experimental design, experimental objects, guides, measuring instruments, and data analysis techniques.

The Experimenters dimension is describing the people involved in the experiment. Gómez et al. (2014) has specified the fact that while doing a replication study, the observed results should be independent of the experimenters, changing the people who performed each role.

The dimensions mentioned will be contextualized in our replication study design. The details are provided in the following sections.

In conclusion, replication in software engineering is one of the key ways to build merit results and to check the consistency of the studies.

4.3 Motivation for the replication

One of the main reasons for conducting a replication study is to confirm the results of the original study but also to expand the knowledge about the replicated method (Dyba et al. 2005).

Our replication further investigates the NB, SVM, and CNN approaches using the Word2Vec model as by the Umer et al. (2020) approach, however, different from the Umer et al. (2018) approach that used the SentiWordNet.

Thus, our replication strategy uses the same methodology for preprocessing the bug report files with specific modifications related to how the feature vector is built. Also, the same source of Eclipse bug reports (Eclipse-bugs 2023), but with some modifications due to the changes in existing bugs and fixes.

5 Research design and analysis

We conducted experiments on CNN based approach classification model and SVM to perform a multiclass prioritization (p1-p5) of bug reports.

In this section, we provide the details of our replication approach, the experimental setup with the used dataset, focusing on employed models, and conducted experiments (for both confirmation and acknowledging the current state of the art on this subject.

5.1 Overview

We replicated the original study from Umer et al. (2018) and Umer et al. (2020) as an operational replication (Juristo and Vegas 2009) where we changed researchers and preserved the replication objects (Gómez et al. 2014). We use the original study protocol, following it as close as possible the initial protocol (based on our understanding of the steps). We tried contacting the authors regarding the scripts, however, we did not succeed. This replication design addresses the internal and external validity threats of the original study and adds new information regarding the use of an automated tool (Frank et al. 2023) to train the model used, but also to implement a new model from scratch using (Python 2023) libraries Galbîn-Năsui (2020). The dimensions stated by (Gómez et al. (2014)) in Sect. 4.2 are next discussed in coordinating our replication design.

Regarding *Operationalization dimension* our experimental replications are similar to the baseline experiment in that they use a similar set of bug reports, and the dataset was extracted from the same repository as this provides both consistency and Methodological Transparency, although the distribution of the bug reports might be slightly different due to the timestamp difference between the extraction of each repository. Details are provided in Sect. 5.3. Using the same dataset allows for a clear exposition of the steps taken to reproduce the study, fostering transparency and facilitating a comprehensive understanding of the research process.

With respect to *Population dimension*, the properties of experimental objects (i.e. bug reports in our context) are considered when designing the experiments. Each experiment considered bug reports extracted from the exact source (Bugzilla 2023), although some of the bugs may be missed from the current repository due to the time difference of the extraction. Some of the bugs were added or removed from the source.

With reference to *Protocol dimension*, some of the elements of the experimental protocol varied in replication. For instance, the study for the JDT dataset using CNN and NB was performed by implementing the code to work on the priority prediction, whereas we have also used some (Python 2023) libraries such as scikitleran. Also, on top of that, we performed a cross-validation study between the four datasets of Eclipse projects (Eclipse-bugs 2023).

With regard to the *Experimenters dimension*, thus referring to the people involved in the experiment, we use different researchers.

In the next sections, we will provide the key elements on top of our experiments and the objects that were used in this approach followed by evaluation metrics that were used to analyze these results.

5.2 Experimental design

We conducted a replication study on the papers (Umer et al. 2018) and (Umer et al. 2020) using two sets of experiments.

We have employed two ways to replicate the approaches by Ramay et al. (2019) and by Umer et al. (2020). We applied the same pre-processing tasks as described below and then we used two different implementations for the used classifiers, i.e., one based on (NLTK (2023)) and one based on WEKA (Frank et al. (2023)). The experiments of the design of the study can be observed in

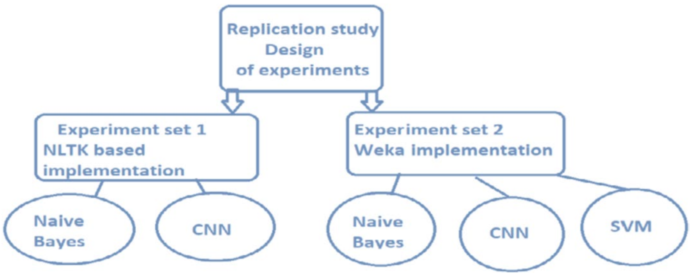

Replication study Design of experiments
Experiment set 1 NLTK based implementation
Experiment set 2 Weka implementation
Naive Bayes
CNN
Naive Bayes
CNN
SVM

Fig. 4 Replication study - design of experiments

Fig. 4. The methods were chosen based on the results in the two original studies, the newly proposed method in each paper with another one for comparison.

5.2.1 Bug reports transformed to vector of words

We apply Natural Language (NLTK 2023) to process the dataset that consists of bug description. Based on the pre-processed bug report, we construct a vector for each bug using the word2vector model. With the predefined word embedding layer from "Google News Word2Vec", we construct our CNN layers. We pass these values to a chosen classifier and evaluate the approach by comparing the evaluation metrics F1, Recall and Precision.

A graphical representation of the approach is shown in Fig. 5. Starting from a simple bug report and taking its description and then using text preprocessing methods so that it is eligible to apply the model, we come to a data set readable for the proposed method. The data set is also divided into training and testing values. The last step is to apply the chosen classifier and get the results to be the priority of the bug.

Text processing. The selected bug reports may contain irrelevant details. In order to get the "clean data" for this study we have applied text processing methods. The aim is to increase the performance of the proposed approach. We include: removal punctuation, stop word removal, lemmatization and tokenization that are described in what follows:

- *Remove punctuation*: Descriptions usually contain punctuation, we remove that as it has no value on the process.
- *Stop word removal*: Bug description may include some unwanted words that have to meaning in the model applyed. We remove such words from the extracted tokens.
- *Lemmatization*: In this process, the endings of the words are removed to return the base word, which is also known as Lemma.
- *Tokenization*: Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string.

To perform the above, we used Python (2023) libraries such as NLTK (2023). After this process, the bug report will be detailed as a vector of words. We apply a *word2vector* model to this grammar, using a predefined embedding layer (Word-2Vec 2023).

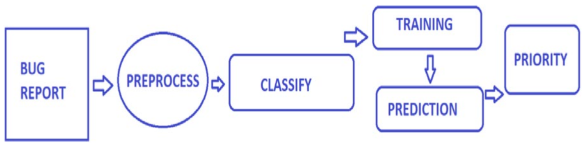


Fig. 5 Overview of bug priority classification process

5.2.2 Employed classification models

As mentioned above, we have employed two ways to conduct the experiments: one using our own implementation using NLTK and the other using the WEKA implementation. We describe in what follows each of them.

Our NLTK - based NB and CNN classification models.

We operated a Naive Bayes classifier approach in order to automatically prioritize bug reports using the report information retrieved from the summary. The factors extracted from the bug reports were further tokenized using Natural Language Processing (NLTK 2023) and served as training and testing data to a Naive Bayes classifier. The first step was extracting the Bugzilla (2023) dataset that includes JDT project bug reports, followed by cleaning this data using NLTK. Furthermore we applied the mentioned model in order to predict the priority class p1-p5.

We also use a CNN classification method in order to predict the priority of each bug report based on its description.. CNN (*Convolutional Neural Network*) uses the vector concatenation method to concatenate incoming inputs into one long input vector (Weber 2020). Consequently, CNN can handle long-term dependencies better than a recurrent neural network.

The first step is to create the embedded layer to which we will apply the other 3 layers using different activation functions. Next, we forward the output of the filter to create a one-dimensional vector.

Regarding the vocabulary formed from this predefined model and the dimensions of the "embedded" matrix, the results can be seen in Fig. 6.

Following this step, the text sequence is transmitted to a CNN and the model is processed (see Listing 1). Finally, we apply a softmax layer that contains the probability distribution of the priority levels.

WEKA - based NB, SVM and CNN classification models.

WEKA Frank et al. (2023) is a collection of machine learning algorithms for data mining tasks. This is an open-source project that provides tools helpful in implementing several Machine Learning algorithms and visualization tools in order to develop machine learning techniques and apply them to real-world data.

Naive Bayes classifiers are a collection of algorithms that implement the probabilistic Bayes Theorem. The main principle while applying NB is that each pair of features is classified independently of the other and contributes to the outcome. The functioning of the NB modeling is determined by formula 1 where A and B are 2 events.

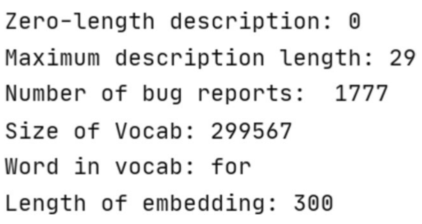


Fig. 6 Embedding model values

```
Zero-length description: 0
Maximum description length: 29
Number of bug reports:  1777
Size of Vocab: 299567
Word in vocab: for
Length of embedding: 300
```

```
embedding_dimension = 180
cnnmodel = Sequential()
cnnmodel.add(layers.Embedding(vocab_size ,
    embedding_dimension ,
    input_length=maxlen))
cnnmodel.add(layers.Conv1D(128, 5,
        activation='relu'))
cnnmodel.add(layers.GlobalMaxPooling1D())
cnnmodel.add(layers.Dense(10, activation='relu'))
cnnmodel.add(layers.Dense(1, activation='sigmoid'))
cnnmodel.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
cnnmodel.summary()
```

Listing 1 CNN Layers

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{1}$$

Support Vector Machine (SVM) is a supervised learning model that uses classification algorithms for the two-group classification problem. It works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable.

CNN Convolutional Neural Network is a artificial neural network. It uses the vector concatenation method to concatenate incoming inputs into one long input vector. Consequently, CNN can handle long-term dependencies better than a recurrent neural network. This is widely used for image/object recognition but has great impact on text classification models.

In the extended study, we applied the SVM, CNN and Naive Bayes classification models from Frank et al. (2023). LibSVM is a wrapper class from Frank et al. (2023) that implements the SVM machine learning model. In order to use Naive Bayes from weka we operate the following class *weka.classifiers.bayes* installed by default on Frank et al. (2023) which uses estimator classes. Furthermore, for CNN, we installed the following WEKA library: *l4jMlpClassifier*. This library allows users to train Neural Networks and conduct experiments using various architectures for the layers.

5.3 Experimental objects

We used two datasets for this implementation, both extracted from the Bugzilla (2023) Eclipse-bugs (2023) project for each study: the dataset for our NLTK-based classification models and the one used in WEKA.

Dataset for our NLTK-based NB and CNN classification models. In this study, we evaluated the performance of the proposed approach on two different datasets, one containing 17700 bug reports of the Bugzilla JDT project (Bugzilla 2023) and one from the same bug tracking system, but containing 1000 bug reports of issues of the database management system reported in Bugzilla. These results were then compared with the approach of using Naive Bayes and Linear Regression from the previous study (Galbîn-Năsui 2020) on the same JDT bug report dataset.

To show the effectiveness of our model, we performed experiments on two data sets; one contains bugs from the JDT project (Eclipse-bugs 2023). The second is part of the same product range (Eclipse-bugs 2023), but explicitly refers to data in the field of databases. Both the reported datasets were downloaded from Bugzilla's repository (Bugzilla 2023). For the dataset related to bug reports from database management field we apply filters by words *database*, *sql* on the same bug tracking system.

Dataset for our WEKA -based NB, SVM and CNN classification models. In this study, we evaluated the performance of the proposed approach on 4 different datasets performing cross-validation between 4 different projects from Eclipse-bugs (2023) available for *JDT, CDT, PDE and Platform.*

We exploit this dataset and reuse it in a cross-validation project using (Frank et al. 2023). The total number of bugs available in the dataset at that time is 28855 bug reports, where we have a number of 66,8% are for the JDT project, 23,2% are for the PDE project, 3,2% on Platform project and 6,8% on the CDT project. The next step was to perform 4 cross-validation experiments where each dataset represented the test data and each of the other 3 datasets was joined as the training data.

5.4 Metrics

Given the bug reports, the performance of the proposed approach is evaluated by calculating the priority-specific precision P, the recall R, and the F1 score F1 (Trevor Hastie and Tibshirani 2016).

Precision (Eq. 2), Recall (Eq. 3), and F-measure (Eq. 4) are some of the most widely used metrics for evaluating classifications. Where for a set of bug reports D, whose real priorities are P_1, P_2..., P_i,.. P_n (n-number of bugs). TP is the number of bugs classified as P_i, FP is the number of bugs that are falsely classified as P_i, and FN is the number of bugs that are not anticipated as P_i, but are actually P_i.

$$P = \frac{TP}{TP + FP} \tag{2}$$

$$R = \frac{TP}{TP + FN} \tag{3}$$

$$F = \frac{2 * P * R}{P + R} \tag{4}$$

6 Replication results

The following section will describe the experimental results and itemize the answers to the research questions raised for this study.

6.1 Our NLTK-based NB and CNN replication experiments results

We first study the accuracy with which each data set is classified, as can be seen in Table 2. Both datasets were used as input data in the CNN algorithm.

We have also investigated the impact of different distributions of training and testing percentages. Table 2 summarizes the results, that is, better results with 25%.

As can be seen, both datasets produced an accuracy of more than 70 %, indicating that most of the priority classes were correctly classified. The high accuracy is due to the frequency of P3 priority bugs, for which the prediction is the most efficient, as can be seen in Tables 3 and 4.

Although the proposed approach has good evaluation metrics for the P3 priority class, as seen in Table 4 for the database dataset and Table 3 for the JDT dataset, the average is less than 50% for the 3 evaluation metrics. We believe that this is due to the dataset we applied, as most of the bug reports had priority P3. This is natural, since usually, for a software product, most of the bugs raised are of medium importance.

Table 2 CNN model. Various testing % distributions

Model used	Accuracy (%)	Test data (%)
JDT dataset	88.1	25
	86.23	15
Database dataset	75.2	25
	72.42	15

Table 3 *JDT* bug reports dataset

Priority	Precision (%)	Recall (%)	F-measure (%)
P1	0.05	0.12	0.13
P2	0.3	0.24	0.28
P3	94.87	93.61	94.99
P4	0.45	0.22	0.41
P5	0.5	0.19	0.2
Average	19.22	19.98	19.02

Table 4 *Management system bug reports* dataset

Priority	Precision	Recall	F-measure
P1	1.25	1.05	0..98
P2	0.85	0.91	0.89
P3	89.97	90.61	90.08
P4	0.13	0.14	0.15
P5	0.1	0.12	0.14
Average	18.92	19.8	19.04

Table 5 Comparison of results for the different models

Model	Accuracy (%)
Naive Bayes + JDT dataset	89.1
CNN + JDT dataset	88.1
CNN + database bug reports	75.2

Table 6 WEKA based evaluation JDT

Model	Precision (%)	Recall (%)	F-measure (%)
NB	19.72	21.58	19.26
SVM	19.04	19.40	19.22
CNN	51.26	21.30	20.32

As seen in Table 3, this approach has shown effectiveness similar to other well-known methods such as Tian et al. (2013), Umer et al. (2018), Yu et al. (2010), Umer et al. (2020), Tian et al. (2012), Sharma et al. (2015).

An important question regarding the model used is the size of the training and testing dataset. For the testing stage, we chose 25% from the bug reports. The results compared are 25% of the total data set size and 15% of the dataset size. For the 15% of the dataset we only managed to get an accuracy score of 72.42% and 83.23%.

We have also compared the CNN model with that implemented in Galbîn-Năsui (2020), Naive Bayes. The results are provided in Table 5 for which we used 1000 bugs from the same JDT Eclipse-bugs (2023) dataset. For this applied method, we have obtained results that are close to the results obtained in Galbîn-Năsui (2022), that aim to predict a bug priority based on description.

As a future direction on this topic, we will employ more bugs to our database dataset and run the experiments on the CNN model and both the Naive Bayes and the Linear Regression model from Galbîn-Năsui (2020). We also consider that in order to improve the results, we could add a few more fields included in the bug report, like: author, product version, and the emotion score of each description of the bug.

6.2 Weka-based NB and CNN replication experiments results

The results obtained usign Weka implementation are provided in the following tables. Table 6 contains the results for the JDT-based testing project for cross-validation, Table 7 for the CDT-based testing project for cross-validation, Table 8 for PDE-based testing project for cross-validation, respectively, Table 9 for the Platform-based testing project for cross-validation. It is important to mention that the values in the specified tables are the average values for all P1 to P5 priorities.

We can notice from the above results that the NB classifier performs best when the project used as testing is Platform (JDT + CDT + PDE as training), the SVM classifiers performs best in the same conditions as NB, while the CNN classifiers perform best when the JDT project is used as testing (CDT+PDE+Platform as

Table 7 WEKA based evaluation CDT

Model	Precision (%)	Recall (%)	F-measure (%)
NB	29.58	22.12	22.42
SVM	17.26	20.00	18.52
CNN	17.26	19.98	18.52

Table 8 WEKA based evaluation PDE

Model	Precision (%)	Recall (%)	F-measure (%)
NB	30.00	21.08	21048
SVM	16.54	20.00	18.10
CNN	19.16	20.76	19.66

Table 9 WEKA based evaluation platform

Model	Precision (%)	Recall (%)	F-measure (%)
NB	24.88	27.86	22.58
SVM	18.92	20.00	19.44
CNN	18.92	19.96	19.42

training). Therefore, more research needs to be done to identify if there is a relation between the characteristics of the projects used for training and the characteristics used for validation. Also, if some classifiers performed better for a specific type of bug reports (thus, a specific fault that is associated with).

Comparing the results considering the bug reports priorities, we can notice that weka obtained better results for the P3 bug priority, as seen in Fig. 7.

We have also investigated with the same classifiers if using as dataset the bug reports for all four projects, thus splitting in 80% training and 20% testing. The results are provided in Table 10. There is a slight improvement for the CNN results compared with all the other cross-validation experiments, the same for SVM, however, not for the NB classifier.

6.3 Results comparison between original and replicated studies

The results of this replication study are lower than the results obtained in the original studies (Umer et al. 2018) and (Umer et al. 2020). In addition, we discuss various perspectives on the results.

The results obtained by Umer et al. (2018) are provided in Table 11 and the results obtained by Umer et al. (2020) are provided in Table 12.

Figure 8 graphically depicts the values for the average F-measures of the four projects (JDT, CDT, PDE, Platform): approach by Umer et al. (2018), approach by Umer et al. (2020), and Weka-based executions.

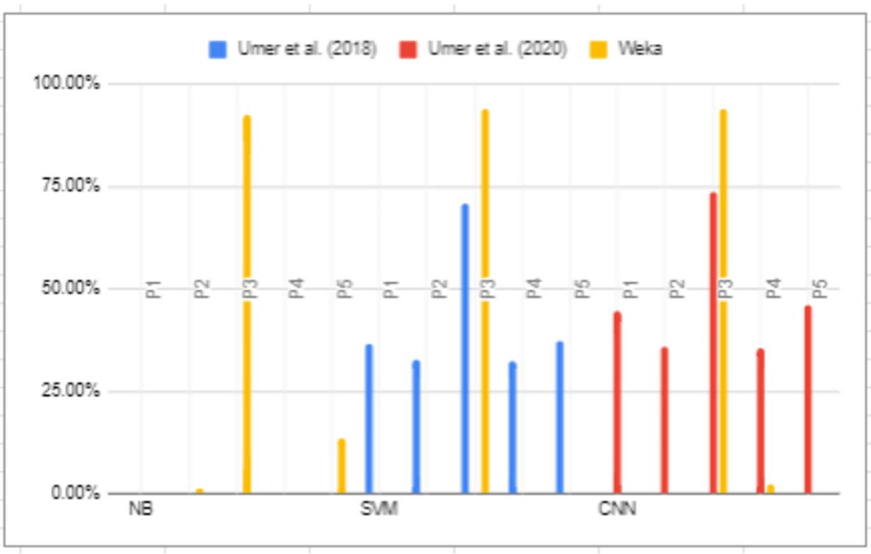


Fig. 7 F-measure results (average from all four projects)

Table 10 WEKA based evaluation All projects

Model	Precision (%)	Recall (%)	F-measure (%)
NB	29.10	19.80	18.24
SVM	31.26	20.86	19.44
CNN	51.26	21.30	20.32

Table 11 Results from Umer et al. (2018) - F-measure

Model	JDT (%)	CDT (%)	PDE (%)	Platform (%)
NB	43.46	43.43	41.47	38.01
SVM	48.84	48.81	45.13	41.98

Table 12 Results from Umer et al. (2020) - F-measure

Model	JDT (%)	CDT (%)	PDE (%)	Platform (%)
SVM (eApp)	48.84	48.81	45.13	41.99
CNN (cPur)	62.07	63.64	73.47	57.67

Our results obtained are different considering each individual project in the cross-validation approach, obtaining lower results; however, when considering the evaluation based on bug priorities, the weka implementation obtained better results for the P3 bug priority case.

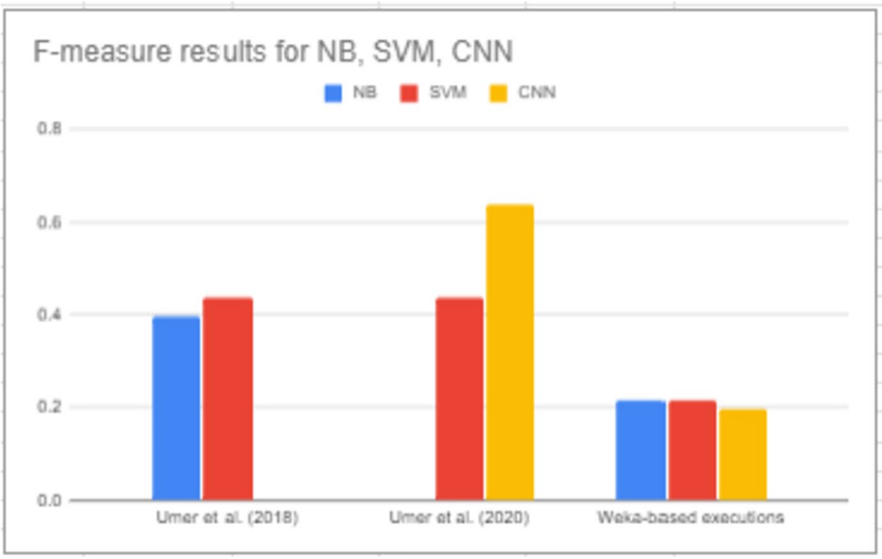


Fig. 8 F-measure results (average from all four projects)

When comparing based on accuracy, the "order" of best classifiers is confirmed as in the previous studies, Umer et al. (2020) to be NB, followed by SVM and CNN. The accuracy for all three classifiers and for each cross-project validation are provided in Table 13.

The differences in the obtained results may be due to how the feature vector was constructed, and also due to the used dataset (which may have different bug reports due to time differences when the data was extracted.

6.4 Discussions

Several contributions have been made in this study. We next summarize them along with possible future improvements from the two perspectives: (1) SLR investigation, and (2) replication study investigation.

Table 13 Accuracy Results for all projects

Model	JDT (%)	CDT (%)	PDE (%)	Platform (%)
NB	90.46	84.32	79.28	90.92
SVM	92.47	86.27	82.68	94.63
CNN	93.61	86.22	80.30	94.41

6.4.1 Discussions on SLR investigation and findings

A notable initial contribution is derived from the *systematic literature review (SLR)* carried out as the basis for this study. Keywords were extracted using our technical expertise, and certain articles that we identified were included based on several criteria in our investigation. The initial database, which consists of 30 articles selected based on an initial assessment of their abstracts, has been reduced to 10 articles that are particularly relevant for this replication. Future research may include other approaches and articles that were published after our SLR.

6.4.2 Discussions on replication study investigation and findings

Furthermore, we *combined text processing tools, incorporating NLTK to streamline the text cleaning process*, especially in the context of an open-source bug database. This integration played a crucial role in boosting the integrity of our research results by eradicating duplicate entries in the articles. In addition to NLTK text preprocessing tools, we employ three different classifiers: SVM, NB, and CNN to determine priority levels. Future investigations may be done considering other classifiers.

An integral aspect of this study is the use of *three different classifiers in the same dataset*. Significantly, these three classifiers are integrated into two different approaches: first, a manual implementation approach involving a script developed by the authors using open-source libraries, and second, the Weka approach.

Following the completion of this study, various areas of potential improvement in future work were identified. A key conclusion drawn is that a cleaner dataset has the potential to enhance the accuracy of the proposed method. The authors intend to expand the dataset in future research efforts and implement a more robust cleanup process to mitigate redundancy. Additionally, addressing the issue of equal distribution of priority classes is crucial. Furthermore, a facet included in future work is the expansion of the classifier base and the incorporation of new features into the input data.

7 Threads to validity

Every experimental analysis may suffer from some threats to validity and biases that can affect the results of the study, and thus our proposed approach in bug reports priority classification is no exception. Several issues which may have influenced the obtained results and their analysis are pointed out in the following.

Threats to internal validity refer to the subjectivity introduced in the selection of the datasets, i.e., the bug reports. To minimize threats to internal validity, we selected two different datasets (JDT and database datasets) with different characteristics (JDT is a plug-in tool support and Eclipse Database tool-related bugs.

Threats to external validity are related to the generalization of the results obtained. Only two datasets were used in our experiments. Another threat to external validity refers to the fact that the datasets used had most of the bug reports with priority P3. We plan to extend the experimental evaluation to other bug reports with all priority levels.

Construct validity refers to checking if the proposed construct is real and if the proposed indicators reflect its target construct. Regarding intentional validity (the constructs adequately represent what we intend to study), we studied the classification of bug report priority using information from bug reports recovered from a bug tracking system (Bugzilla 2023). Regarding the validity of the representation (how well do the constructs or abstractions translate into observable measures), we used bug reports with various priority levels. Thus, the sub-constructs define the construct. Regarding observation validity (how good are the measures themselves), the values in our feature vector come from real bug reports.

Threats to *conclusion validity* are associated with the relationship between treatment and outcome. These threats are mitigated by ensuring that the experimental assumptions are valid. In this investigation, the threat to the conclusion is mitigated by classifying the bug reports using the same categories/priorities that are used in other investigations.

8 Conclusions

A bug is often transmitted with either an incorrect priority level or without defining the priority level. The people who sort the bugs entered in the system go through the reports manually and have a number of responsibilities, among which they assign priority to each bug report. Manually prioritizing bugs requires both expertise and human and time resources. Priority is particularly important in influencing the lifespan of a bug, and, therefore, setting this value properly increases efficiency in product development.

In this paper, we reviewed the literature regarding bug report priority classification approaches and then conducted a replication study on three classifiers, that is, Naive Bayes, Support Vector Machines and Convolutional Neural Networks, using the Eclipse dataset with four projects. Cross-project validation was performed and the results confirmed the initial studies regarding the three classifiers, i.e., obtaining the best results for the SVM and CNN classifiers. Ultimately, we conducted a comparison of the three different classifiers using both the implemented methods and Weka. The results analysis indicates that the Naive Bayes Classifier outperforms CNN on the JDT dataset when utilizing the implemented classifier. However, when employing Weka, CNN yields superior results.

Further investigation is needed to establish the relationship between the characteristics of the projects being used for training (and therefore also the type of bug report). Exploring various combinations of project type (thus bug reports) for the cross-validation process may bring new insights on the best classifier in a specific context.

Acknowledgements This work was funded by the Ministry of Research, Innovation, and Digitization, CNCS/CCCDI - UEFISCDI, Project number PN-III-P1-1.1-TE2021-0892 within PNCDI III. We also would like to thank professor Alexander Serebrenik, our research collaborator, from the Eindhoven University of Technology for providing us with improvement suggestions for the study and useful insights on how to improve the paper.

Data availability and materials Data will be made available on request.

Declarations

Conflict of interest Author Andreea Vescan and Andreea Nasui-Galbin declares that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Alenezi, M., Banitaan, S.: Bug reports prioritization Which features and classifier to use? 2013 12th International conference on machine learning and applications **2**, 112–116 (2013)

Bugzilla. Bugzilla: Bug tracking systems. https://www.bugzilla.org/. (Accessed March 2023)

Carver, J.C.: Towards reporting guidelines for experimental replications: a proposal. The international workshop on replication in empirical software engineering (pp. 2–5) (2010)

Carver, J.C., Juristo, N., Baldassarre, M.T., Vegas, S.: Replications of software engineering experiments. Emp. Softw. Eng. **19**(2), 267–276 (2014). https://doi.org/10.1007/s10664-013-9290-8

Dyba, T., Kitchenham, B.A., Jorgensen, M.: Evidence-based software engineering for practitioners. IEEE Softw. **22**(1), 58–65 (2005)

Eclipse-bugs, E.: Eclipse. https://bugs.eclipse.org/bugs/. (Accessed March 2023)

Fagerholm, F., Becker, C., Chatzigeorgiou, A., Betz, S., Duboc, L., Penzenstadler, B.,...Venters, C.C.: Temporal discounting in software engineering: A replication study. 13th acm/ieee international symposium on empirical software engineering and measurement (pp. 1-12). IEEE.(2019, 10 17)

Frank, E., Hall, M.A., Witten, I.H.: Weka - waikato environment for knowledge analysis. (Accessed March 2023). https://www.cs.waikato.ac.nz/ml/weka/

Galbîn-Năsui, A.: Predictia automata a prioritatii unui bug (Unpublished master's thesis). Bachelor's thesis. Babes-Bolyai University Cluj-Napoca Faculty of Mathematics and Computer Science.(2020)

Galbîn-Năsui, A.: Bug reports priority classification model (Unpublished master's thesis). Babes-Bolyai University Cluj-Napoca Faculty of Mathematics and Computer Science. (2022)

Gómez, O.S., Juristo, N., Vegas, S.: Understanding replication of experiments in software engineering: a classification. Inf. Softw. Technol. **56**(8), 1033–1048 (2014). https://doi.org/10.1016/j.infsof.2014.04.004

Juristo, N., Vegas, S.: Using differences among replications of software engineering experiments to gain knowledge. In: 2009 3rd international symposium on empirical software engineering and measurement (356–366) (2009)

Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. 2 (2007)

Myers, G.: The Art of Software Testing. Wiley, Hoboken (2005)

NLTK. Natural language toolkit. https://www.NLTK.org/. (Accessed March 2023)

Patton, R.: Software testing. SAMS. (2000)

Python.: Python: Python-programming language. https://www.python.org/ (Accessed March 2023).

Ramay, W.Y., Umer, Q., Yin, X.C., Zhu, C., Illahi, I.: Deep neural network based severity prediction of bug reports. IEEE Access **7**, 46846–46857 (2019). https://doi.org/10.1109/ACCESS.2019.2909746

SentiWordNet.: Sentiwordnet-lexical resource for opinion mining. https://github.com/aesuli/SentiWordNet. (Accessed March 2023)

Sharma, G., Sharma, S., Gujral, S.: A novel way of assessing software bug severity using dictionary of critical terms. Procedia Comput. Sci. **70**, 632–639 (2015). https://doi.org/10.1016/j.procs.2015.10.059

Shepperd, M., Ajienka, N., Counsell, S.: The role and value of replication in empirical software engineering results. Inf. Softw. Technol. **99**, 120–132 (2018). https://doi.org/10.1016/j.infsof.2018.01.006

Tian, Y., Lo, D., Sun, C.: Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: 2012 19th Working conference on reverse engineering (215–224). (2012)

Tian, Y., Lo, D., Sun, C.: Drone: predicting priority of reported bugs by multifactor analysis. In: 2013 IEEE international conference on software maintenance (200–209). (2013)

Trevor Hastie, J.F., Tibshirani, R.: The Elements of Statistical Learning. Springer, Cham (2016)

Uddin, J., Ghazali, R., Deris, M.M., Naseem, R., Shah, H.: A survey on bug prioritization. Artif. Intell. Rev. **47**, 145–180 (2017). https://doi.org/10.1007/s10462-016-9478-6

Umer, Q., Liu, H., Illahi, I.: CNN-based automatic prioritization of bug reports. IEEE Trans. Reliab. **69**(4), 1341–1354 (2020). https://doi.org/10.1109/TR.2019.2959624

Umer, Q., Liu, H., Sultan, Y.: Emotion based automated priority prediction for bug reports. IEEE Access **6**, 35743–35752 (2018). https://doi.org/10.1109/ACCESS.2018.2850910

Weber, B.G.: Data science in production: Building scalable model pipelines with python. (2020). Independently published (January 1, 2020)

Word2Vec . Word2vec- used to learn word embeddings. https://www.tensorflow.org/tutorials/text/word2vec/. (Accessed March 2023)

Yu, L., Tsai, W.-T., Zhao, W., Wu, F.: Predicting defect priority based on neural networks. In: Cao, L., Zhong, J., Feng, Y. (eds.) Advanced Data Mining and Applications, pp. 356–367. Springer, Berlin Heidelberg (2010)