

HOSTED BY



Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

ProRE: An ACO- based programmer recommendation model to precisely manage software bugs

Ashima Kukkar^a, Umesh Kumar Lilhore^b, Jaroslav Frnda^{c,d}, Jasminder Kaur Sandhu^e, Rashmi Prava Das^{f,g}, Nitin Goyal^h, Arun Kumarⁱ, Kamalakanta Muduli^{j,*}, Filip Rezac^d

^aChitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India

^bDepartment of Computer Science and Engineering, Chandigarh University, Gharuan, Mohali, 140413, India

^cDepartment of Quantitative Methods and Economic Informatics, Faculty of Operation and Economics of Transport and Communications, University of Zilina, 01026 Zilina, Slovakia

^dDepartment of Telecommunications, Faculty of Electrical Engineering and Computer Science, VSB Technical University of Ostrava, 70800 Ostrava, Czech Republic

^eDepartment of Computer Science and Engineering, Chandigarh University, Mohali 140413, India

^fDepartment of Computer Science and Engineering, CV Raman Global University, Bhubaneswar, India

^gDepartment of Computer Science and Engineering, Bhubaneswar Engineering College, Odisha, India

^hDepartment of Computer Science and Engineering, Central University of Haryana, Mahendergarh, Haryana 123031, India

ⁱDepartment of Computer Science and Engineering, Galgotias College of Engineering and Technology, Greater Noida, (U.P.) 201306, India

^jPapua New Guinea University of Technology, Lae, Morobe, PMB 411, Papua New Guinea

ARTICLE INFO

Article history:

Received 20 October 2022

Revised 21 December 2022

Accepted 23 December 2022

Available online 29 December 2022

Keywords:

Software Engineering

Bug Assignment

Programmer Recommender System

Ant Colony Optimization

Feature Weighting

Triager

Machine Learning Methods

ABSTRACT

The process of assigning bugs to particular programmers is called bug assignment in software engineering. The programmer can fix the bugs by applying their knowledge. This research article presents an Ant colony optimization-based programmer recommendation model (ProRE) to manage software bugs precisely. The proposed ProRE model performs four operations: data pre-processing, i.e., data Pre-processing, extraction, feature selection, and programmer recommendation process. The feature selection stage utilized the Ant colony optimization (ACO) method to determine the appropriate subsets of features from all features. In the programmer recommendation stages, three programmer metrics, i.e., functionality ranking, bug occurrence, and mean Bug fixing time, are utilized for the recommendation assignment. The effectiveness of the proposed programmer recommendation system is assessed using datasets from Mozilla, Eclipse, Firefox, JBoss, and OpenFOAM. It is noted that the proposed model offers a better recommendation strategy over the other available systems. The simulation findings of the proposed ProRE model are also analyzed with well-known available ML methods, i.e., SVM, NB, and C4.5. It is observed that the recommendation results have improved by an average of 4%, 10%, and 12% compared to SVM, C4.5, and NB-based models. Programmer recommendation software is implemented for allocating the bugs to accurate programmers. It has been found that the proposed ProRE model generates more optimistic outcomes than existing ones.

© 2022 The Author(s). Published by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

The software propagates bugs during the SDLC's evolution, testing, and maintenance stages (Wu et al., 2022). Software bugs' primary reasons are misinterpreting requirements, implementation and design strategies, error handling mechanisms, incorporation of a new provision in code, etc., (Jorayeva et al., 2022). Whenever a software project is implemented, a centralized database is maintained to store bugs reported during the execution of software (Batoool and Khan, 2022). Bug directories and surveillance systems play a vital role in the SDLC process. Many bugs are deposited in bug directories (Zaidi et al., 2022).

* Corresponding author.

E-mail addresses: ashima@chitkara.edu.in (A. Kukkar), umeshlilhore@gmail.com (U. Kumar Lilhore), jaroslav.frnda@uniza.sk (J. Frnda), jasminder2011@gmail.com (J. Kaur Sandhu), rashmidas3@gmail.com (R. Prava Das), dr.nitingoyal30@gmail.com (N. Goyal), ranarun1.ece@piet.co.in (A. Kumar), kamalakanta.muduli@pnuot.ac.pg (K. Muduli), filip.rezac@vsb.cz (F. Re).

Peer review under responsibility of King Saud University.



Production and hosting by Elsevier

Further, it is observed that bug fixing is an expensive, standard, and time-consuming task (Mo et al., 2022). So, in modern practice, a programmer set can be considered to fix the bugs rather than individual ones. A bug report contains some basic information regarding the failure, such as content, status, textual description, etc., and much information is missing (Zheng et al., 2022; Guo et al., 2022). If a programmer knows Bug, then bugs can be efficiently fixed. Hence, identifying the appropriate programmer to fix the bugs is challenging (Cynthia et al., 2022).

For example, in Eclipse open-source project, approximately 30 new bug reports were reported daily to BTS in 2007 (Lilhore et al., 2022). Between October 2001 and December 2010; 333,371 bug reports were reported to the Eclipse bug tracker. By 34,917 users and programmers, these issues have received more than fifteen Lakhs comments. Besides, all such bugs are allocated to different programmers for debugging and fixing. If the programmer handed the Bug cannot solve it, it is assigned to some other programmers. The process of reassigning bugs to another programmer is known as bug tossing. This process remains continued until Bug could not be solved.

Fig. 1 shows the bug-tossing process. The bug tossing length is described in terms of the number of people (Yang et al., 2022). Let programmer A is allocated the job of fixing a bug. The programmer, however, is unable to resolve the Bug. The identical Bug was then given to programmer B, who could not fix it. Then the system will assign the Bug to programmer C.

The process will repeat until the Bug is not fixed (Jahanshahi et al., 2022). The length of bug tossing can be given as AaBbC. It must be defined as the total number of employees allotted to rectify the Bug. The B bug chucking can also enhance the bug-checking time (Liu et al., 2022). For example, the Eclipse open-source implementation needs 40 to 50 days to complete the tossing procedure before the appropriate programmer receives the bugs (Malhotra et al., 2021).

It is due to the manual inspection of bug reports through an expert triager. The manual Bug triaging process is costly and time-consuming due to the lack of information regarding bugs (Zaidi and Lee, 2021, January.). For example, in the open-source Eclipse project, approximately 44 percent of bugs are allocated accidentally to different programmers (Oliveira et al., 2021). The time cost of opening bugs and their first assignment is 19.3 days (Almhana and Kessentini, 2021). So, the programmer recommendation system (DRS) aims to minimize the bug tossing length (BTL).

Further, it is noticed that some bugs require immediate attention from the programmer. The triager makes a manual effort to identify the severity and priority of bugs. This manual effort is

tedious and time-consuming and can be reduced through automated bug resolution. So, one of the solutions for the problem mentioned above is to design a programmer recommendation system that can identify the appropriate programmer for bug fixing and measure the severity of bugs (Garg et al., 2021, August.). In literature, it is noticed that various researchers have mined the software repositories, i.e., source code repositories, bug repositories, and e-mail archives, to find unseen information from data. It is also observed that several recommendation systems have been developed to help the triagers, such as priority recommendation of bugs (Nath et al., 2021), predicting components of bug reports, identification of relevant programmers (Panda and Azar, 2021), bug resolve time prediction (ABsRo et al., 2021), and bugs severity prediction (Khleel and Nehéz, 2021). Machine learning and mining techniques are also applied for severity prediction and recommendation. These techniques aim to predict the label of newly submitted bug reports. Table 1 shows the two bug reports of the eclipse, and Fig. 2 illustrates the communication among programmers.

1.1. Motivation

Consider two fixed bug reports of the Eclipse project with bug IDs 127,495 and 403,841 (Sun et al., 2021; Ferenc et al., 2020). Table 1 contains the details of the programmers. A single programmer provides the Bug information through bug reports; while several comments on bug reports. It is observed that both bug reports consist of two standard programmers, i.e., csmclaren and tjwatson. The commenting process of different commenters is illustrated using Fig. 1. Edges represent the connection between various commenters and are labeled with numeric values. The numeric value denotes the number of times a programmer commented on the same bug report. It is seen that two commenters provide comments on both bug reports twice. At the same time, the rest are commenting a single time.

Further, researchers are active programmers in research (Trivedi et al., 2020; Esteves et al., 2020; Prabha and Shivakumar, 2020, June.) because they comment more than once to solve bug issues. So, it is the task of the triager to identify the relevant programmer (Gupta et al., 2020, June.). The triager provides sufficient details concerning the bugs and programmers' competence. But, this is a significant challenge for which the programmer is nominated to fix the Bug. The triager has two options for csmclaren and tjwatson (who have remarked on both bug statistics) to work on the Bug's resolution28. Additionally, it is noted that there is no evidence regarding the severity of the Bug, which necessitates immediate attention.

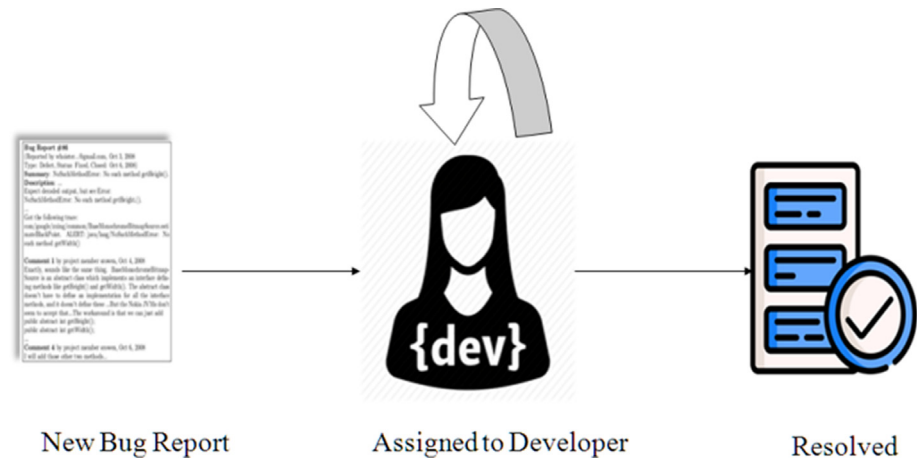


Fig. 1. Bug Tossing Procedure.

Table 1
Two Bug Reports of Eclipse.

Commenter Order	Programmer	
	Bug ID 127,495	Bug ID 403,841
Reporter	Chris Bailey	csmclaren
Commenter 1	csmclaren	csmclaren
Commenter 2	john_arthorne	kim.moir
Commenter 3	ChrisBailey	csmclaren
Commenter 4	john_arthorne	kim. moir
Commenter 5	ChrisBailey	tjwatson
Commenter 6	john_arthorne	kim.moir
Commenter 7	tjwatson	csmclaren

1.2. Contribution of work

This work aims to propose a programmer recommendation system for bug assignments. The proposed programmer recommendation system addresses the tossing length and identification of appropriate programmer issues of the automated bug resolution process. The contribution of work is highlighted as follows:

- The main focus of this research is to develop an efficient bug-fixing system that can correctly identify the desired programmer for a particular Bug.
- This research developed an ACO-based feature selection system, which helps to select the relevant set of data features from the available bug dataset, which helps to improve the bug-fixing process.
- Develop three metrics to rank the programmers (ProRE), which helps programmers with bug assignments.
- The proposed scheme minimizes the tossing length and determines the correct bug-fix programmer.
- The proposed model improves the bug-allocation accuracy and reduces the allocation of time and effort.

In this work, the ProRE model is designed to help the triagers for assigning the bugs. The proposed technique aims to shorten the tossing period and pinpoint the ideal programmer for problem fixes. The proposed system's evaluation is conducted through well-bug datasets. Its results are compared with many existing ml algorithms suggested by previous research. The simulation

results revealed that the proposed ProRE model works more efficiently than existing systems and predicts more accurate programmers for assigning bugs.

The complete article is organized as follows: [Section 2](#) covers the related work; section 3 covers the materials and methods; section 4 covers the procedure of recommendation and results in discussion and [Section 5](#) covers the conclusion and future work.

2. Related work

Software engineering is an active study area for the software development process. It helps in the identification and triage of software bugs based on importance. Bug triage aims to allocate a suitable programmer to solve the bugs. This section gives the most recent research on finding the correct programmers to repair bugs and bug-fixing life processes. Murphy addressed the automatic bug triage problem ([Alazzam et al., 2020](#)). The goal of this study is to lower the amount of conventional physical assignments. Text classification algorithms are adopted in this work to predict the programmers for bug fixing. Research ([Pandey et al., 2020](#)) implemented a semi-automatic bug triage model using ML and data processing strategies. The proposed approach considered a database of appropriate programmers based on skill set to solve the bugs.

Additionally, it can be observed that the proposed approach immediately allocates the new bug tracker to the programmer for bug repair and reduce the time and effort. Research ([Pombo and Teixeira, 2020](#)) developed a tossing graph-based recommendation system. The proposed recommendation model integrates the Markov-based approach to predict the set of appropriate programmers. It is also observed that the proposed model reduces the bug tossing length to 72 % compared to a machine learning-based recommendation system.

The bug triage process's programmer assignment and assignee task problem are discussed in ([Subbiah et al., 2019](#)). The researchers took three difficulties into account using a social media network-driven technique: I) Programmer rating, II) the Evolution process, and III) Tolerance. Throughout this approach, social network assessment aspires to prioritize the coders who actively engage in the posting comment procedures of Bug solving activities. Additionally, NB and SVM classifiers are used to choose the

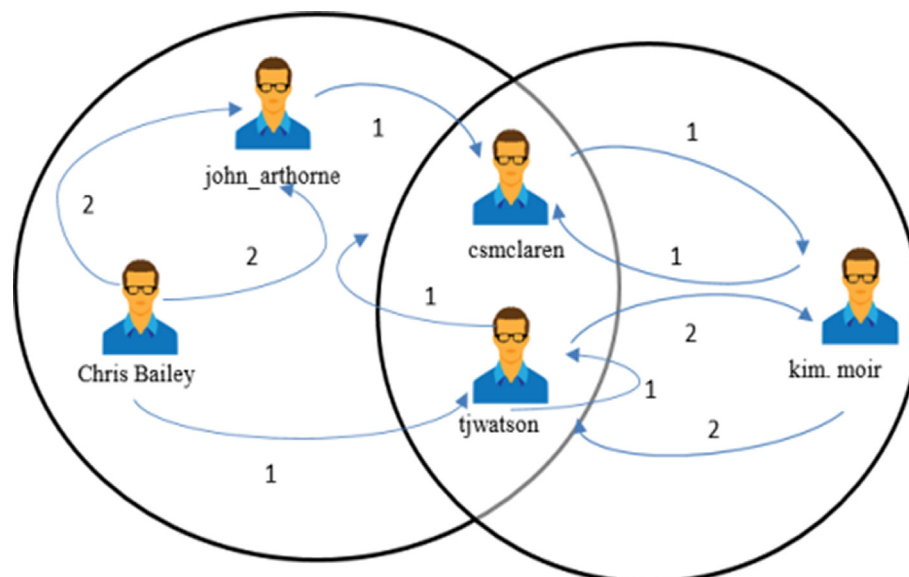


Fig. 2. An illustration of the communication among programmers.

correct coder. According to the findings, the social network analysis strategy significantly enhances NB & SVM's recognition rate by 2 % and 10 %. A novel technique was suggested by research (Yadav et al., 2019) to improve the outcomes of bug assessments. The proposed model depends on the text resemblance of bug reports and the tradition of programmers passing around bugs. The researchers also found duplicate issues for all of this work's bug allocation methods. The suggested method cut the bug tossing time by approximately 80 to 90 percent. The bug triage challenge was described as an optimization process in research (Mani et al., 2019). In this research, the researchers create an organizational filtering-based premium triage method to reduce the time required for issue fixes.

A programmer recommendation engine incorporating five ML techniques is discussed in (Lee and Seo, 2019). These strategies are Rules, DT, NB, C4.5, and SVM. The proposed model operates in two stages. An adequate group of bug reports is gathered from bug directories throughout the first stage. In the second step, ml algorithms rated the software programmers. The proposed model obtains a 75 % prediction performance for five projects, i.e., Myly, GCC, Bugzie, Eclipse, and Firefox projects. It is shown that the suggested system works with current programmers who can fix related defects for a particular project. A social network-based rating strategy for selecting the coder for Bug solving was formed by research (Hammouri et al., 2018). The suggested approach integrates competence rating with the KNN model. The research findings demonstrated that the social media platform's out-degree and basic frequency measurements outperform existing methods. Further, the proposed model achieves more than a 60 % recall rate for Mozilla and Firefox datasets.

A bug tossing graph (BTG) and a machine learning-based approach is discussed in (Aljedaani and Javed, 2018). The authors consider bugs' products and components in this work to determine the dataset's relevant features. These features train the various ML-based classifiers like NB, SVM, C4.5, BN, and bug tossing results. With shorter tossing paths for Mozilla & Eclipse, the NB classification performed better than the BN classifier; these shorter tossing paths are approximately 80 to 90 percent and 87 percent faster. To allocate the applicable programmer for bug correction, Research (Angel et al., 2018) created a tossing graph-based platform termed "programmer recommendation based on topic models" (PREAM). Using the recall factor, the simulation outcomes of the DRETOM system are contrasted with that of the Bugzie (Sahu et al., 2018) and DREX (Jaiswal et al., 2018) methods. According to research observations, DRETOM fares better than the Bugzie & DREX frameworks.

Therefore for the Eclipse JDT sample and the Mozilla sample, the proposed DRETOM obtains recall rates of more than 80 % in both. Research (Kumari et al., 2018) presented a bug triage approach to finding the appropriate bug fixer or resolver. The proposed bug triage approach is based on cache modeling and a fuzzy set called Bugzie. In this approach, the score of each programmer is calculated using the previous Bug fixing records. The simulation results stated that the proposed Bugzie approach obtains a prediction rate between 75 % and 83 % for bug datasets.

A programmer ranking model based on concept location (CL) approaches to help the bug triager (Uddin et al., 2017). The proposed model uses latent semantic indexing to extract comments and identify source code and corpus indexation. To increase the precision of bug allocations, Research (Kumari et al., 2018) established a TG-based technique taking programmers' problem models and relationships into consideration. It is noted that the experimental initiative showed a better F-measure proportion over DRETOM. Another technique for semi-automatic bug triage was created by investigation (Xuan et al., 2017). This method analyzed the influence of its multi-features selection in bug monitoring and

evaluation. The suggested methodology combines several techniques, including Kullback-Leiber, which involves a detailed and basic NLP. The simulation results reveal that, across all databases, the suggested system outperforms In, AP-based product recommendation, and DRETOM in terms of excellent performance.

A data reduction problem related to bug triage is reported in (Shrivastava et al., 2017). In this work, the authors reduced the dimensions of bug report data using feature selection approaches. A predictive design is produced employing the characteristics of earlier bug reports. The accuracy of the model is evaluated over Eclipse & Mozilla feature sets. The experimental results indicate that only the dimension reduction procedure increases the precision of bug triage.

A diverse approach, KSAP, with the K-nearest-neighbor search technique to enhance the precision of bug attribution, is discussed in (Chandra et al., 2017). It can be demonstrated that KSAP offers better outcomes than the currently used models. Research (Xuan et al., 2017) presented a multi-score programmer recommendation system based on modern portfolio theory for bug triage to consider time cost and recommendation accuracy. The efficiency of the proposed model is measured using the Bugzie project. The proposed ProRE model provides more effective recommendation results for the bug triage process. Research (Kaur et al., 2017) developed a semi-supervised classifier to address the deficiency associated with labeled Bug. The proposed approach adopted Naive Bayes (NB) classifier and the expectation-maximization (EM) algorithm to predict the relevant programmer. It is noted that the precision rate of the suggested semi-supervised prediction model is significantly greater than NB and NBEM methods.

A novel programmer suggestion technique to decrease the human labor required to identify the comments (Sun et al., 2017). The research findings demonstrated that over the top 10 rankings, the proposed ranking method generates a more excellent recognition rate of 81 percent. Another work to reduce the length of bug tossing is presented in (Sama et al., 2016). In this work, the authors considered the concept of programmer expertise score for reducing bug tossing length. The suggested techniques mainly utilize three Stage s. The first stage corresponds to the programmer's scores. The programmer score is computed using Jaccard, cosine similarity, and feature-based similarity measures. In the second stage, programmers are ranked based on programmer scores. The performance evaluation is done through reassignment accuracy and Hit-ratio.

The proposed algorithm's effectiveness is validated in the third stage with ML-based techniques such as NB, SVM, and C4.5. It is noted that the proposed ProRE model achieves twenty percent improved results than ML-based approaches. Research 51 developed a bug triage model called "Deep bidirectional recurrent neural network with attention" (DBSRNN-A). The performance of DBSRNN-A is evaluated using Mozilla Firefox, Google Chrome, and Mozilla datasets. It is revealed that DBSRNN achieves better results as compared to existing models.

2.1. Research gaps

The comprehensive literature review reveals that most investigators evaluate the similarity among relevant topics and remarks based on historical and new logs. Programmers are selected depending on participation to solve the nearly identical bug report. Furthermore, a few works also developed metrics to build the programmer performance assessment profiles (RPAP) and recommend the programmer. It is also seen that a lack of work is presented on feature selection techniques in bug severity prediction and programmer recommendation. In contrast, these techniques provide better results in diverse fields. To determine the research gap in existing studies, three parameters/attributes are considered. These

parameters are the availability of feature selection technique, no. of metrics, and programmer ranking.

Further, the number of citations is also considered a parameter that signifies the importance of existing work. Hence, it is concluded that a lack of work is reported on feature selection techniques for programmer recommendation, especially the severity level of bugs. Some results also considered metrics for recommending the programmer's name. Furthermore, it is also seen that programmer ranking can be done either based on bug report comments or suggested programmer names in the comment section.

Hence, it is stated that the current work did not consider all parameters mentioned above for the programmer's recommendation to fix the bugs. This work considers all three parameters, i.e., feature selection method, metrics, & bug report remark, to advise the programmer to take corrective action. The feature selection method focuses on determining a relevant set of features for programmer recommendation, which can be accomplished through the ACO-based feature weighting technique.

The three metrics are also designed for programmer ranking. This research utilizes three key components a) programmer capability ranking, b) level of bug severity, and c) Mean Bug fixing time. The proposed recommendation is also considered the various components of bug reports in the final recommendation of the programmer name. It is also said that researchers are not focused on reducing BTL through the combination of severity classification and appropriate programmer prediction to fix a bug. This work also explores the applicability of bug severity levels to determine the appropriate programmer for fixing and reducing the BTL. Table 2 represents the studies of a programmer recommendation system for software bugs.

3. Materials and methods

This section covers the proposed model working. The complete steps are as follows:

3.1. Proposed ProRE model

This section presents the programmer recommendation system for bug assignment. Before predicting the appropriate pro-

grammer for fixing the bugs, an ACO-based feature weighting method is incorporated into the proposed recommendation system. Fig. 3 demonstrates the ProRE model's intended schematic operation.

The feature weighting method aims to identify the optimum features from the feature space. Further, three metrics are developed to rank each programmer. These measurements include programmer ability ratings, bug intensity levels, and mean Bug resolving times. NB and SVM-based ML methods estimate the severity of bugs on an online dataset (Onan et al., 2016; Dorigo and Blum, 2005). Additionally, it is presumable that a directory of programmers with a track record of effectively and speedily resolving defects is also present according to the seriousness of the Bug. The main steps of the proposed ProRE model are described below:

- Data Pre-processing,
- Extraction of features,
- Feature weighting by using the ACO method, and
- Assignment of bugs (to programmer).

a) Data Pre-processing Stage

Data Pre-processing Stage is the initial stage of the planned ProRE model. The contents of bug reports are processed by using different data-preprocessing techniques. In practice, bug reports are composed of paragraphs written in any linguistic form, preferably English. The different phases in this stage are used to examine the bug reports. These actions are provided as follows:

- The tokenization process** involves breaking the text input into individual words, numerals, and whitespace. The tokenization procedure aims to identify the unique tokens for each bug document. Additionally, it is mentioned that whitespace might be used in place of punctuation. Additionally, the non-printable escape character is eliminated, and capitalization is changed to lowercase for phrases. Stop-word elimination process: The stop words in the logs are identified in this phase. Stop words are words that include content that is unrelated to a particular context. The stop words can be any verb, noun, article, pronoun, adverb, prepositions, etc. Few examples of stop words such as

Table 2
Studies of Programmer Recommendation systems.

Reference	Key Method Used	Feature Selection	No of Metrics	Programmer Ranking	Hybrid Model used
(Subbiah et al., 2019)	Machine learning as a service (MLaaS)	✗	0	✓	✗
(Yadav et al., 2019)	Expertise score method	✗	1	✗	✗
(Mani et al., 2019)	Deep learning	✗	2	✗	✗
(Lee and Seo, 2019)	Bug Report Processing Techniques	✗	0	✓	✗
(Hammouri et al., 2018)	Machine learning approach	✗	0	✓	✗
(Aljedaani and Javed, 2018)	Data Mining techniques	✗	0	✓	✗
(Angel et al., 2018)	Effective Bug Processing and Tracking System	✗	0	✓	✗
(Sahu et al., 2018)	KNN & NB with hybrid selection method	✗	0	✓	✗
(Jaiswal et al., 2018)	Genetic Approach and sequencing the instance and features	✗	1	✗	✗
(Kumari et al., 2018)	Machine Learning Methods	✗	1	✗	✗
(Uddin et al., 2017)	Artificial Intelligence	✗	1	✗	✗
(Kumari et al., 2018)	Machine Learning Methods	✗	2	✗	✗
(Xuan et al., 2017)	Semi-supervised methods	✗	2	✗	✗
(Shrivastava et al., 2017)	CCM and textual features	✓	0	✓	✗
(Chandra et al., 2017)	Modified Random forest	✗	0	✓	✗
(Xuan et al., 2017)	Bug commenting method	✗	0	✓	✗
(Kaur et al., 2017)	software entropy with Machine Learning	✗	2	✗	✗
(Sun et al., 2017)	ant colony optimization	✗	0	✓	✗
(Sama et al., 2016)	Ant colony optimization	✗	3	✗	✗
(Zhang et al., 2016)	ML based bug resolution	✗	0	✓	✗
Proposed Model	ProRE Model	✓	✓	✓	✓

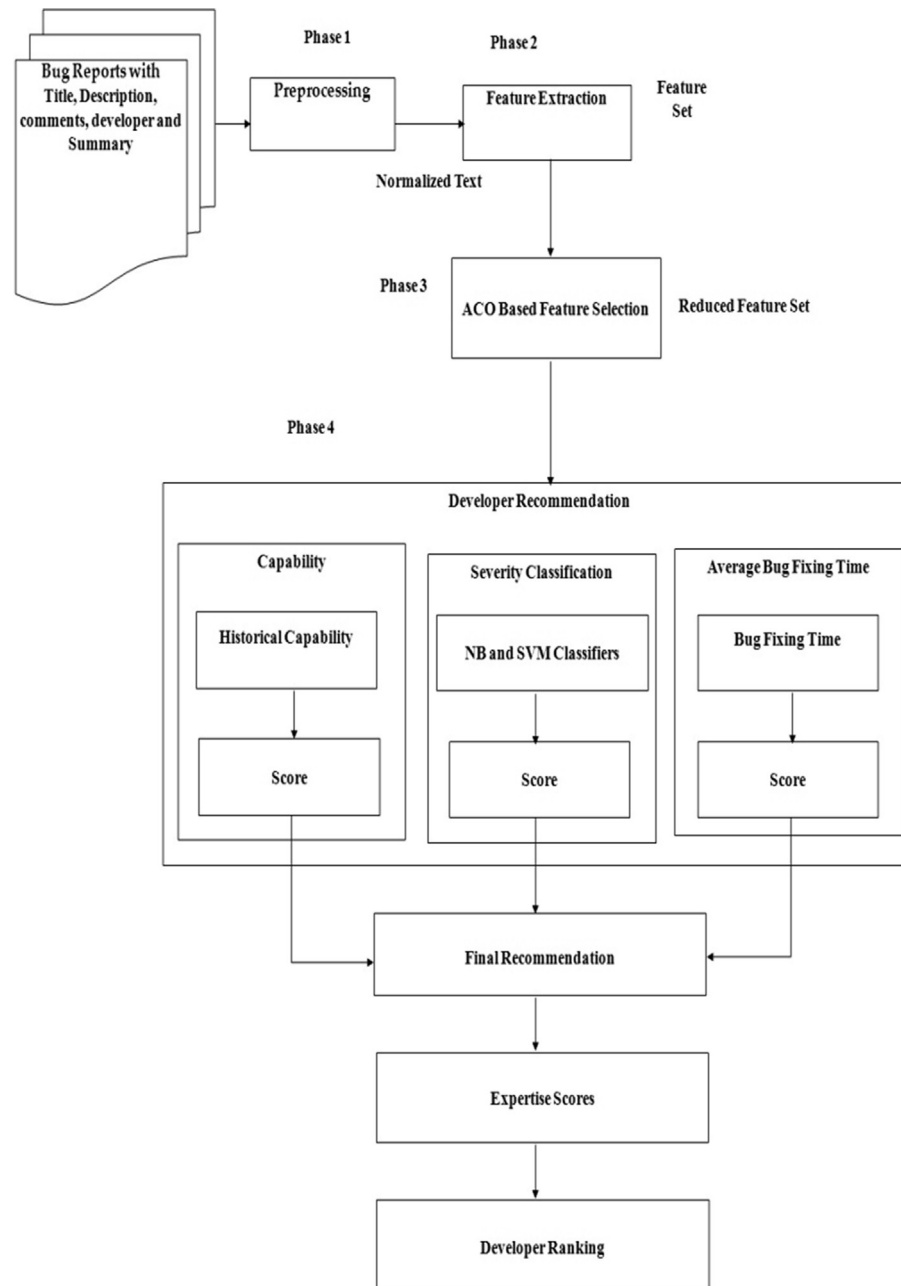


Fig. 3. Proposed Programmer Recommendation System for Bug Assignment.

Table 3

Illustrates a few of the stop words identified in bug reports.

a	about	across	again	against	they	this
almost	alone	long	already	also	them	below
although	always	am	am	another	all	from
among	anyhow	anyone	an	at	themselves	using
any	are	around	anything	the	else	that
anywhere	not	being	as	at least	for	because

'an', 'the,' at,' 'and,' 'although,' 'anywhere,' 'as,' 'a', etc., are removed from tokens 53, 54. In this study, <https://pypi.python.org/pypi/stop-words> API is used to identify the stop words, and a total of 328 words are identified as stop words, and a few of these are listed in Table 3.

• **Stemming:** This is a method of determining the typical phrase stem. "Stemming is a natural language processing method (NLP) that reduces phrase phrasing to its root aspects, helping to prepare text, phrases, and records for signifiers". The whole widespread word stem can be kept in several features as

attributes. Let's say that a word appears several times in a bug report, with variations including "relocate," "relocates," "relocated," and "relocating." The word "relocate," which is referred to as an attribute, can be used in place of each incidence. Equations (1) and (2) can be used to simply mathematical express it. Where BSR represents the set of determining stemming, BS represents the set, and fs is the feature set.

$$BSR = \{BS_1, BS_2 \dots \dots B.S_j\} \quad (1)$$

$$B.S_j = \{fs_1, fs_2 \dots fs_k\} \quad (2)$$

b) Extraction of features: Inside this stage, text-based features can be extracted and described as a vector model using TF-IDF and N -gram techniques 55–56. The weighted characteristics of the proposed ProRE framework are used to explain bug reports. Eq. (3) shows a representation of this procedure.

$$BSR_x = (wb_1fs_1, wb_2fs_2, \dots, wb_nfs_n) \quad (3)$$

Where $wb(i)$ represents the weight of an i th unique feature and n represents the maximum quantity of features. TF-IDF is described as an additional statistical technique for retrieving significant attributes. This approach takes word recurrence into account when identifying attributes in bug reports. The frequency of features determines their importance. The significance and weight of phrases (features) repeatedly occurring in event logs are greater than those of other phrases or characteristics, as described in Eq. (4).

$$TF - IDF(x, y) = tf(x, y) \times \log \left(\frac{N}{df(y)} \right) \quad (4)$$

Algorithm 1 (Pre-processing and Feature weighting).

Input: $BSR_h = (BSR_1, BSR_2, \dots, BSR_h)$ a textual bug report from the bug report dataset.
 Output: $BSR_h = (BSR_1', BSR_2', \dots, BSR_h')$ a set of features for bug report dataset.
 Step 1: First, peruse each bug document from the bug report sample data.
 Step 2: The text has been segmented, and stop terms were eliminated.
 Step 3: Use the TF-IDF formula to measure the frequency of each unigram characteristic.
 Step 4: Determine the average severity of every unigram feature using the TF-IDF equation.
 So every feature can be represented by TF-IDF in vector space, and formula 5 describes the correlating bug report vector.

$$V_{BSR_x} = (tf(BSR_x, y_1), tf(BSR_x, y_2) \dots tf(BSR_x, y_x)) \quad (5)$$

A unique vocabulary feature expresses the size of the bug report vector. Consider the term frequency with feature y_1 in bug document (BSR) 2 to be $tf(BSR_2, y_1)$. Let's say the bug document does not mention a query characteristic. Within the scenario, the correlating feature inside the matrix is given a zero. Numerous different features' fundamental values will also be set to zero, where they exceed this valuation. Data sparsity is the inability to extract the correlating bug report. The N -gram technique addresses the problem of sparse data (Jiang et al., 2016).

In the N -gram method, the smoothing function handles the sparsity issue. The smoothing function can be defined as add-1 smoothing, add- k , stupid back off, and Kneser-Ney. In this work, we adopt the add-1 smoothing function of the N -gram model, also known as Laplace smoothing. This function adds "1" to all counts of

the bigram method. Now, the "0" count can be interpreted as "1," "1" by "2," "3" by "4," and so on. Further, the normal bigram probabilities are computed through normalized row counts of the unigram method, which is given in Eq. (6).

$$P_{Lap}(f_n/f_{n-1}) = \frac{C(f_n f_{n-1}) + 1}{C(f_{n-1}) + V} \quad (6)$$

$P_{Lap}(f_n/f_{n-1})$ Denotes the Laplace smoothing function of n th feature w.r.t. $(n-1)$ th feature, $C(f_n f_{n-1})$ denotes the count of the word using the bigram method, $C(f_{n-1})$ represents the count of words using unigram method. V is the total vocabulary words of the given bug report. Further, the adjusted count is computed for smooth bigram probabilities using Eq. (7).

$$C^*(f_n f_{n-1}) = \frac{[C(f_n f_{n-1}) + 1] \times C(f_{n-1})}{C(f_{n-1}) + V} \quad (7)$$

$C^*(f_n f_{n-1})$ denotes the adjusted value of $C(f_n f_{n-1})$. Finally, a data matrix is constructed that contains the probability of words using the bigram method and removes the maximum of "0" entries before applying the N -gram method.

c) Feature Selection Stage

The whole stage's primary objective is to choose the pertinent characteristics for appraising the intensity of bugs. Fig. 4 depicts the ACO-based feature ranking process in action. An example is a bug report with n features, m data instances, and k clusters. The BD1, BD2, BD3, ..., B.D. N codes are used to characterize the characteristics of a bug document.

Numerous characteristics are identified in the feature extraction stage utilizing statistical and semantic data from logs. It has been noted that none of the features carry the same weight. The presence of many features can occasionally cause non-linearity issues in the data as well as a reduced prediction percentage. Effectiveness can sometimes consequently experience severe. Hence, selecting essential features is a crucial trick that can enhance the performance of a classifier by selecting the pertinent characteristics through the feature set. The best features for bug occurrence are chosen in this research using an ACO-based feature selection method. To resolve constrained but also unconstrained combinatorial optimization complications, Dorigo et al. created this method (Mohammad et al., 2016). This method has been used to remedy issues like feature optimization, cluster analysis, internet of things, image analysis, sending data, etc., and it produces the best outcomes for these issues.

An ACO-based feature ranking method is used in the feature screening stage to find the salient characteristics for bug severity forecasting. This method seeks to determine the relative importance of each characteristic in the extracted features. When predicting the intensity of bugs, characteristics with higher weights are chosen because they have a high impact. Following is a description of how the ACO-based feature ranking method operates. The complete dataset is separated into the k subgroups to evaluate the essential features. Random selection is employed to determine the k number of clustered centers. An ACO-based clustering is also used to improve the cluster centers. The following step involves using the range feature to group the various data occurrences into the k clusters. To organize data into appropriate clusters, the Euclidean distance method is typically utilized (Vijayarani et al., 2015; Wang et al., 2015). Now, the weight of every attribute in the feature set is determined using a weight feature function. The features are chosen for intensity forecasting according to the weight feature.

Fig. 4 shows the ACO-based feature weighting and selection process.

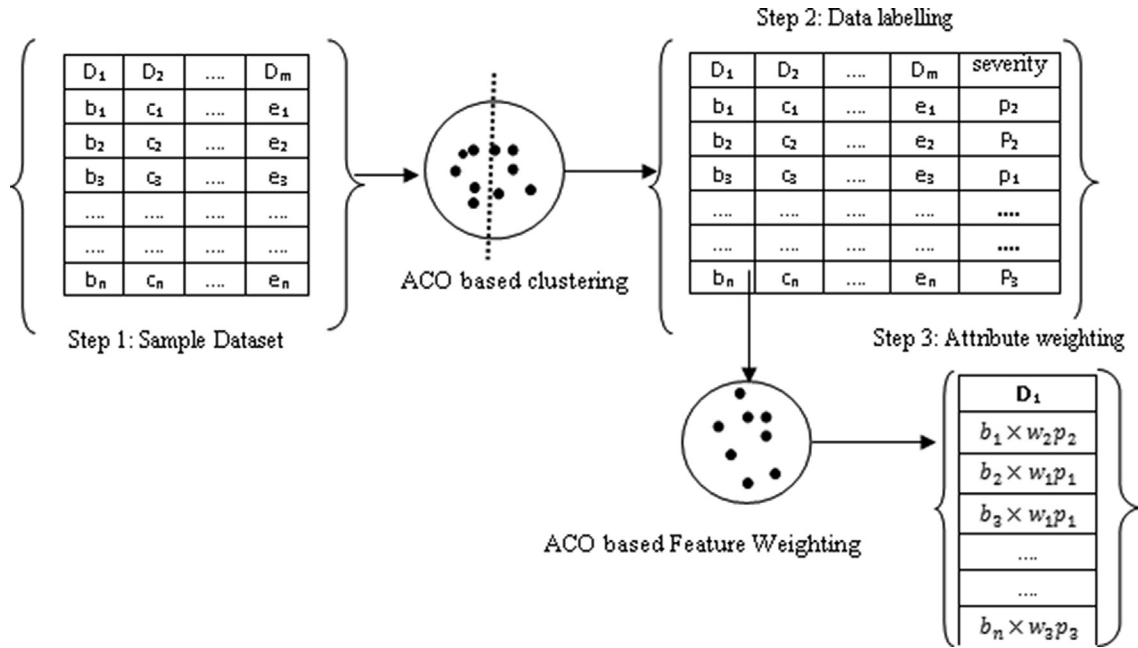


Fig. 4. ACO-based feature weighting and selection process.

3.1.1. The steps of ACO based feature weighting technique are given as follows:

- Step 1: shows the bug dataset's characteristics and the values obtained.
- Step 2: Correlates directly to the level of the bug severity explored by the ACO algorithm, including BSLEV_1, BSLEV_2, BSLEV_3, ..., and BSLEV_n, which can be connected to each bug document. For every level of severity, a weight feature is also calculated. To determine the final weight of attributes, the above weight feature is multiplied first by the quality of the individual features.
- In step 3: An algorithm called "weight function determination" is used to determine the significance of each feature in the bug data source. Eq. (8) determines each element's weight function (w_f) for the test dataset. Where W_i shows the value of a weight parameter for an i th parameter, F_i represents the average weight for an i th parameter, C_i represents a center value of the i th cluster, as described in Eq. (8).

$$W_i = \frac{\sum_{j=1}^D F_{ij}}{\sum_{j=1}^D C_i}, \text{ i.e., } F_i = \frac{\sum_{k=1}^{BSL} \sum_{i=1}^n x_{i,k}}{i} \quad (8)$$

Variable i shows the count of reports within the characteristics with similar severities, and $x_{i,k}$ signifies the count of files with k^{th} stages. The BSL indicates bug report severity. Algorithm 2 outlines the phases of the proposed ACO-based method for selecting features. A smaller set of features is the result of the feature selection stage. Additionally, these features forecast the Bug's severity and start training the classification algorithm.

d) Programmer Recommendation Stage

The principle of a programmer recommendation framework is presented in this subsection. A ranking of the appropriate programmers to recognize bugs and issues is provided in this stage. Through the use of metrics, correlation, and recommender system, tools, the right programmers are located.

3.2. Proposed programmer metrics

The principle of a programmer recommendation framework is presented in this subsection. A ranking of the appropriate programmers to recognize bugs and issues is provided in this stage. Through the use of metrics, correlation, and recommender system, tools, the right programmers are located. Following is a description of all these operations.

- **Capability Score (C_score):** The unique ability of the programmer to resolve bug data is the basic foundation for all of this ranking. It has been observed that a particular bug report involves a unique set of skills to be resolved. A programmer is said to be capable of dealing with bug reports effectively if a bug of a similar nature has recently been fixed. Thus, the skills of programmers can be investigated to address bug solutions. Furthermore, it has been found that programmers' skills can be evaluated by using algorithms for machine learning about historical records. A new bug is confirmed in the database, and its severity has been considered first. To solve it, algorithms for machine learning identify an application programmer. A Capability Rating is determined by the capacity of programmers to solve bugs. These can be calculated by employing programmers' expertise to rectify the bugs. As a result, the Capability Score is used in this research rather than machine learning classification models to grade the programmers.

Notations: η_A shows the prior understanding of features to enhance the outcomes, $\tau_A(T)$ captures the quantity of pheromone for the A th characteristic at period T , α , and β show the command variables that supply the pheromone and context, and $p_A(T)$ shows the transition probability value, ρ shows the extraction efficiency of the pheromone route and lies among intervals 0 and 1, $\Delta\tau_A(T)$ indicates the number of pheromones incorporated into an A th characteristic over time Δ , and ΔT , d shows the count of features, h shows the count of data instances, k shows the number of clusters, X_{ih} is the i th feature value of h^{th} data instance, C_{ik} shows the i th, feature of a k th cluster set.

- **Bug Severity Level:** Bug severity is a prospective measurement for overall ranking programmers throughout this research. It is a known simple truth that not each Bug requires attention immediately or is crucially significant. A few bugs are rated more severe than others and need a programmer's urgent attention. If not, strange events will also be observed. The impact of event logs, divided into various severity levels, must be evaluated. Learning techniques enable quick completion of the task. The literature subsequently contains datasets for bug reports. Classifiers are trained on any of these datasets to anticipate risk levels. Consider that the database is updated with the latest bugs. The risk of all these bugs can be anticipated using classifiers such as NB and SVM.

Additionally, the perfect programmer is assigned to resolve the bugs. It should be noted that a ranking of programmers is created here based on bug severities. Then a programmer is capable of fixing bugs with multiple severities. To sustain the order of severities managed by each programmer, a weighted severity final list is described. It can be described as the proportion between the total counts of bugs fixed and the various severities of bugs managed by a programmer. Eq. (12) shows the weighted severity calculation function for each programmer.

$$SL_{list} = \sum_{i=1}^{SL} \beta \times \frac{Bug_i}{Total\ Bug} \quad (12)$$

In Eq. (12), Bug_i denotes an *i*th level of bug severity, SL denotes the number of severity levels, β represents the criticality of bugs to be resolved and SL_{list} denotes the severity list associated with programmers.

- **Average Bug Fixing Time (BFT_{avg}):** A average amount of time it takes for a programmer to resolve bugs is used to explain this standard measure. Finding and fixing involves many steps, including formulation of the problem by software programmers, System Development, and source code examination and evaluation. Depending on the severity of the bugs, it is also critical to determine the appropriate programmer. The proportion of the overall duration a programmer spends fixing bugs towards the total count of bugs fixed can be used to figure out the average Bug solving time. When bug reports are analyzed, it is discovered that they contain a variety of fields, including element, component, time and date, severity level, competence, etc. Eq. (13) calculates the programmers' overall rankings after multiplying the earlier performance.

$$D_{rank} = \gamma_1 C_{dev} + \gamma_2 SL_{list} + \gamma_3 T_{avg} \quad (13)$$

In Eq. (13), D_{rank} denote the final rank of programmers, C_{dev} describes the capability learning of programmers, SL_{list} represents the severity level of bugs and T_{avg} represents the average bug-fixing time. γ_1 , γ_2 and γ_3 are three control parameters such that $\gamma_3 < \gamma_1 < \gamma_2$ moreover, these parameter values are in the range of 0 and 1.

3.2.1. Procedure of similarity

A feature is drawn directly from bug reports only when a new bug is noted within the bug archive. Additionally, the similarity procedure is used to assess the severity of occurrences and choose the right programmer. The whole research incorporates similarity measures to evaluate the similarities between events of bugs and suitable programmers. A group of potential programmers to resolve similar bug types is created thru the similarity procedure.

Algorithm 2 (Proposed “ACO-Based Feature Weighting” Algorithm).

Input: Bug logs dataset	
Output: Feature set with no redundant value	
Step 1:	Set the initial cluster centers' numbers after loading the dataset.
Step 2:	Specify the highest iteration & pheromone trial severity after initializing the ant population.
Step 3:	Each ant's initial cluster centers have been randomly chosen.
Step 4:	Allocate the tokens to the clusters with the minor optimization criterion after evaluating the objective value weighting scheme using the squared sum of the distances.
Step 5:	While (condition true).
Step 6:	Apply Eq. (9) to determine the transfer probability.
	$p_A(T) = \frac{(\tau_A(T))^{\alpha} \cdot \eta_A^{\beta}}{\sum_A (\tau_A(T))^{\alpha} \cdot \eta_A^{\beta}} \quad (9)$
Step 7:	Eq. (10) should be used to upgrade the pheromones.
	$\tau_A(T+1) = \rho \tau_A(T) + \Delta \tau_A(T) \quad (10)$
Step 8:	Explore the ants' new locations.
Step 9:	An objective function's significance must be recalculated.
Step 10:	End while
Step 11:	Generated the best possible cluster centers.
Step 12:	Using Eq. (11), calculate the end weights for each feature vale fv _i .
	$fv_i = \left(\sum_{i=1}^d \sum_{j=1}^h \sum_{k=1}^k \frac{x_{ik}}{c_{ik}} \right) \times \frac{1}{d} \quad (11)$
Step 13:	Choose the functionalities with the highest weight.

3.3. Procedure of recommendation

Selecting the programmers' rank represents the final process of the programmer recommendation phase. Programmers are rated based on their aptitude for handling and resolving bugs. Each programmer is ranked using data and information related to their capabilities, bug intensity, and mean bug fixing period. Eq. (13) shows how to calculate the programmer's overall rankings. The similarity measure is used to identify a competent programmer based on the end ranking. It retrieved the right programmer from the framework to resolve the bugs.

Further, it is said that the top-ranked programmer receives responsibility for fixing bugs. The next programmer on the ranking will acquire it in case the top-ranked programmer is unavailable. Bugs are still not fixed till this process has been finished.

4. Results and discussion

This section outlines the simulation outcomes of the proposed programmer recommendation framework. Java coding is used to develop and implement the proposed methodology. The efficiency of the proposed framework is evaluated over four bug report datasets, i.e., JBoss, Firefox, Eclipse, Mozilla, and OpenFOAM (Wang et al., 2015; Yang et al., 2015). Table 4 shows the description of all these datasets. The precision, recall, accuracy, and F-measure parameters are also considered when determining the severity of bugs (Jadhav et al., 2015).

Table 4
Specific details of logs (Bug reports) taken into account for the experiment.

Dataset	No. of Bugs	BTS	Severity Level
Eclipse	693	Bugzie,	5
JBoss	573	Redhat Bugzie,	5
OpenFOAM	795	Manits	8
Mozilla	539	Bugzie,	7
Firefox	620	Bugzie,	7

The Recall@5 and the Recall@10 Parameters (Alazzam et al., 2020) [63] are utilized as a comparison to assess how well the proposed programmer recommendation system continues to perform. In addition, ten crossfold strategies, as opposed to the training and testing set techniques, are used in this investigation. The original dataset is split into ten equal portions for the ten crossfold methods; nine of these parts serve as the training phase, and the remaining portions serve as the testing set. The correct procedure is iterated up to ten times, so the testing set differs from the last. Two experiments evaluate how efficiently the proposed developed recommendation algorithm performs. These tests are described as follows:

- **Experiment 1:** Forecasting of Bug Severity Rating
- **Experiment 2:** Selecting the Accurate Programmers to Fix Bugs

4.1. Experiment 1: Results and discussion

The simulation outcomes of the predicted bug severities are shown in this subparagraph. ACO-NB and ACO-SVM strategies are used to anticipate the different bug severity. The weight of each feature in a specified dataset is first calculated using an ACO-based feature weighting method. A weight function is used to choose the best attributes.

To compute the weight of features, we can implement the proposed feature selection technique in an unsupervised manner. Further, the proposed approach is executed thirty times independently with different seed points (different data points selected from the given dataset) such that a fair computation can be performed. The average weight of these thirty runs is assigned to each feature of the dataset. The parameter settings of ACO based feature weighting technique is mentioned in Table 5. An ACO method execution depends on the iteration count; in the experiment, we use max iterations value of 100. Besides this, SVM and NB classifiers anticipate the bug severity. Table 4 shows all the user-defined specifications of the SVM method. The precision, recall, f-measure, and accuracy parameters are used to assess the simulation outcomes of these methods.

The following key parameters are used to measure the performance of the proposed model:

The necessary notations used in the equation are TP: True Positive, TN: True Negative, FN: False Negative, and FP: False Positive.

Table 5
User-defined parameters of ACO-based feature weighting technique and SVM classifier.

ACO-based Feature Weighting Technique		SVM	
ρ	0.4	c	1.0
α	0.7–1	Kernel function	RBF
β	0.2–0.6	degree	3
τ_0	0.1	Width(p)	0.2–0.5
η_0	ranging between 0 and 1	–	–

- **Accuracy (Ac):** Accuracy is one measure for measuring the classifier model. It measures how many forecasts of a method are correctly predicted. Eq. (14) shows the formula for accuracy.

$$Ac = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (14)$$

- **Precision (Pc):** The model's capacity to make a correct prediction. Precision is calculated by dividing the cumulative percentage of relevant optimistic forecasts by the number of correct. Eq. (15) shows the formula for precision.

$$Pc = \frac{TP}{(TP + FP)} \quad (15)$$

- **Recall (Rc):** The recall is determined by dividing the total quantity of Positive tests by the number of Positive tests correctly identified as Positive. The recall predicts how well the model can identify positive cases. Eq. (16) shows the formula for precision.

$$Recall = \frac{TP}{(TP + FN)} \quad (16)$$

- **F-measure (Fm):** A harmonic average of precision and recall is used to determine the F-measure, providing almost the same strength. Eq. (17) shows the formula for precision.

$$Fm = \frac{2TP}{(2TP + FP + FN)} \quad (17)$$

Table 6 shows bug severity quality forecasting simulation outcomes utilizing five well-known bug datasets, i.e., OpenFOAM, Mozilla, Eclipse, Firefox, and JBoss. Using all available datasetRs, it is noticed that the ACO-NB method outperforms the ACO-SVM, SVM, and NB approaches in terms of performance. This is also noted that the SVM algorithm generates the worst outcomes for most data sets. In contrast, it is also claimed that ACO-SVM offers more precise bug severity than SVM and NB strategies. Thus, it can be said that the ACO-based method for selecting features significantly enhances the results of SVM and NB computation. Table 6 shows the Simulation results of proposed bug assignment models, i.e., NB, ACO-NB, SVM, ACO-SVM, & SVM approaches using five datasets.

Fig. 5 displays the simulation outcomes for the Eclipse bug dataset for NB, ACO-NB, SVM, ACO-SVM, and SVM. In this experiment, the performance parameters calculate each method's accuracy, precision, F-measure, and recall. According to the resulting proof, the ACO-NB procedure achieves more remarkable outcomes than other strategies. Additionally, it has been noted that SVM performs the lowest of all methods.

The simulation results of bug severity level prediction using the Firefox dataset are presented in Fig. 6. The ACO-NB approach obtains better results predicting severity levels than the ACO-SVM, NB, and SVM techniques. It is also revealed that the NB approach gives more accurate results than the ACO-SVM and SVM approaches due to the linearity of data. It is noticed that if data is linearly separable, then the performance of the SVM classifier is not up to mark.

Fig. 7 presents the simulation results of ACO-NB, NB, ACO-SVM, and SVM approaches using the OpenFOAM dataset. It is noticed that ACO-NB outperforms other methods being compared. It is also observed that the SVM approach achieves better recall and precision rates than the NB approach. At the same time, the NB approach provides better results using F-measure and accuracy parameters for OpenFOAM datasets.

Table 6
Simulation results of proposed bug assignment models.

Approaches	Bug Datasets	Performance Measuring Parameters Results in %			
		Precision	Recall	F-measure	Accuracy
ACO-NB	Mozilla	78.75	77.51	78.61	80
	Eclipse	81.05	77.43	79.69	79.6
	JBoss	80.85	79.23	80.52	79.6
	OpenFOAM	77.59	76.85	77.71	78.25
	Firefox	77.04	77.22	77.62	79
Avg. Results of ACO-NB		79.06	77.65	78.83	79.29
ACO-SVM	Mozilla	75.71	77.09	76	76.21
	Eclipse	73.33	72.08	72.53	74.72
	JBoss	73.65	72.56	72.62	75.24
	OpenFOAM	76.46	73.33	74.52	74.52
	Firefox	71.78	68.01	68.78	71.78
Avg. Results of ACO-SVM		74.19	72.61	72.89	74.49
NB	Mozilla	74.32	71.79	73.52	75.85
	Eclipse	72.85	71.22	72.52	73.4
	JBoss	74.05	70.42	72.69	74.8
	OpenFOAM	73.71	72.24	73.45	74.5
	Firefox	73.03	72.08	73.05	75.57
Avg. Results of NB		73.59	71.55	73.05	74.82
SVM	Mozilla	72.62	75.51	73.57	73.67
	Eclipse	73.56	68.73	70.89	71.67
	JBoss	73.95	69.12	70.91	71.67
	OpenFOAM	74.46	72.65	73.14	71.87
	Firefox	69.22	67.22	67.38	65.34
Avg. Results of SVM		72.76	70.65	71.18	70.84

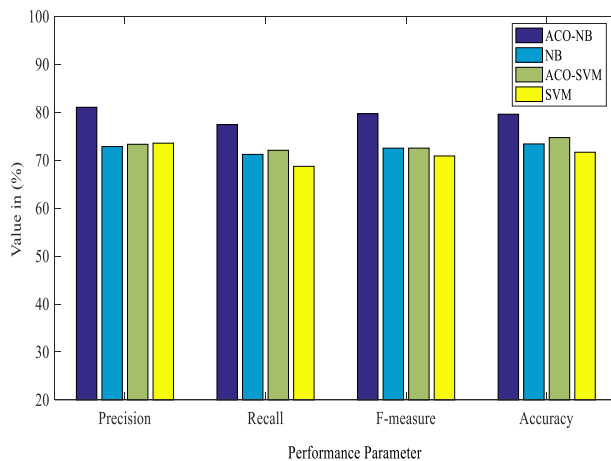


Fig. 5. Comparison graph for Proposed ML methods (NB, ACO-NB, SVM, ACO-SVM, SVM) on Eclipse bug dataset.

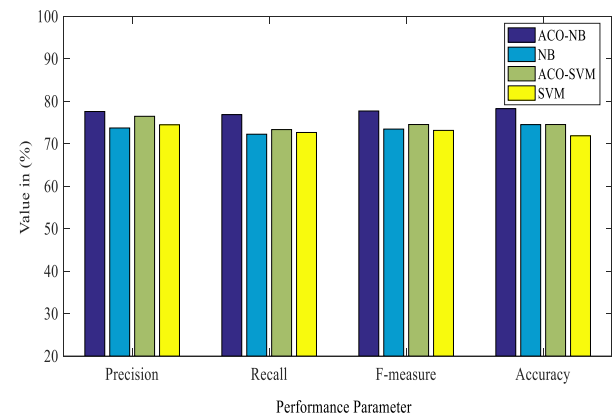


Fig. 7. Bug severity level prediction using ACO-NB, NB, ACO-SVM, and SVM approaches using the OpenFOAM dataset.

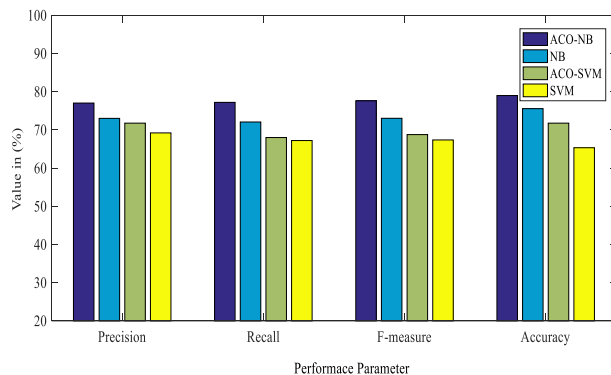


Fig. 6. Bug severity level prediction using ACO-NB, NB, ACO-SVM, and SVM approaches using the Firefox dataset.

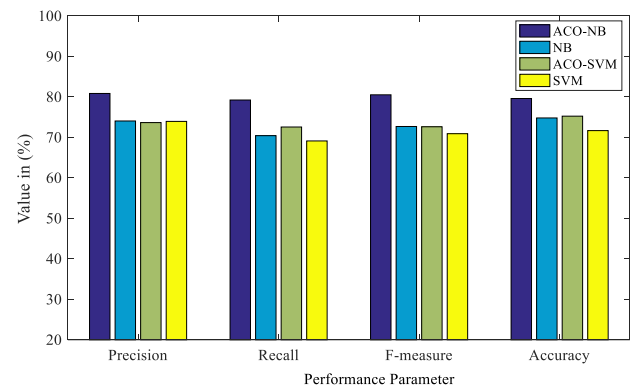


Fig. 8. Bug severity level prediction using ACO-NB, NB, ACO-SVM, and SVM approaches using the Jboss dataset.

The simulation results of all approaches for severity level prediction using the JBoss dataset are reported in Fig. 8. It is stated that ACO-NB provides better outcomes for severity level prediction as compared to ACO-SVM, NB, and SVM approaches. Further, the SVM approach exhibits worse results for the JBoss dataset using recall, F-measure, and accuracy parameters. It is also observed that the precision rate of ACO-SVM, NB, and SVM approaches is almost similar.

Fig. 9 illustrates the results of the bug severity level of ACO-NB, ACO-SVM, NB, and SVM approaches using the Mozilla dataset. It is noted that the ACO-NB approach predicts the bug severity more accurately compared to other approaches. It is observed that SVM exhibits poor performance for the Mozilla dataset in terms of precision and accuracy parameters. In contrast, the recall rate of the NB approach is worse among all approaches. It is also observed that the accuracy rate of ACO-SVM and NB approaches is almost similar. Hence, it can be revealed that the ACO-NB approach provides more promising results for bug severity level prediction.

4.1.1. Statistical test

This subsection presents the statistical outcomes of the Friedman test (F-test) to determine the significant disparity between the proposed and other methods' effectiveness. In this study, two approaches are developed based on ACO based feature weighting technique. These approaches are ACO-NB and ACO-SVM. Two hypotheses are developed before applying the Friedman test, i.e., H0 and H1. Hypothesis (H0) corresponds to no significant difference between the proposed approaches and other techniques, whereas hypothesis (H1) corresponds to the considerable variation between proposed approaches and other techniques. The statistical results of the Friedman test are presented in tables 7–8.

Table 7 shows the overall rating of NB, ACO-NB, SVM, ACO-SVM, & SVM strategies. The results show that the ACO-NB method ranks first among other strategies. Additionally, it has been found that the SVM algorithm produces more unsatisfactory outcomes than other methods.

Table 8 includes details on the Friedman test statistical data. The F test yielded an analytical value of 14.04. The F test (0.01, 4) has critical and p-values of 11.34489 & 0.002851, respectively. The freedom level degree is three. As a result, it is declared that at a significance value of 0.01, hypothesis (H0) is rejected. The results of NB, ACO-NB, SVM, ACO-SVM, & SVM strategies undertake completely different. This is observed that the ACO-NB strategy is considerably different from other strategies.

A post hoc assessment is also run to verify the effectiveness of the ACO-NB strategy. Table 9 displays the outcomes of the Post-

Table 7

Average ranking of NB, ACO-NB, SVM, ACO-SVM, & SVM approaches using accuracy parameter for all datasets.

SVM	NB	ACO-SVM	ACO-NB
4	2.8	2.2	1

Table 8

Statistics of Friedman Test using accuracy parameter of all approaches used in experiment 1.

Statistical Value	Critical Value	p-value	Hypothesis
14.04	11.34489	0.002851	Rejected

hoc assessment. The “S” denotes the significant difference between the performances of different approaches. It is observed that the ACO-NB approach is significantly different from ACO-SVM, NB, and SVM. Hence, it can be stated that the Post-hoc test also validates the performance of the ACO-NB approach.

The Post-hoc test groups the approaches into three clusters. Cluster 1 consists of the ACO-NB technique; Cluster 2 consists of the ACO-NB and SVM approach; Cluster 3 contains the SVM approach. Hence, there is no significant difference between the performance of ACO-SVM, and NB approaches as per Post-hoc test results. In contrast, these techniques obtain significantly different simulation results. It is revealed that ACO-NB (ACO-SVM and NB) and SVM approaches exhibit pretty different performances. Hence, it has been proved that the ACO-NB approach significantly differs from other techniques.

4.2. Experiment Two: Results and discussion

This experiment corresponds to the selection of the best programmers for bug fixes. To identify the best programmer, each programmer is ranked using three measures. Based on the report, the severity level measurement is said to carry a relatively high weight compared to the other two measures. Five bug datasets are used to evaluate the performance of the proposed ProRE method. When assessing the performance of the ProRE recommendation framework, recall & precision factors are chosen to take into account. Here is a description of these factors.

Recall@k: It can be described as the percentage of bugs recommended to the relevant programmers in a given list size, i.e., k, concerning the total number of programmers. This metric is calculated using Eq. (18).

$$\text{RecallRec} = \frac{\text{Bugs Recommended to programmer @ k}}{\text{Total number of programmer}} \quad (18)$$

Precision@k: It can be described as the percentage of bugs recommended to the relevant programmer in a given list size, i.e., k, concerning @k relevant programmer. This metric is calculated using Eq. (19).

$$\text{Precision} = \frac{\text{Bugs Recommended to programmer @ k}}{\text{Number of programmers @ k}} \quad (19)$$

The effectiveness of the proposed methodology is evaluated throughout this research utilizing recall@5, precision@5, and recall@10, precision@10 metrics. The simulation outcomes of the proposed programmer recommendation system are shown in Table 10. It has been noticed that the proposed recommendation system performs better with recall@10. Additionally, the recall@10 percentage for the proposed system, which is 68.13 for the Eclipse bug dataset, is significantly more significant. However, for the Firefox bug dataset, a poor recall@10 percentage of 52.17 is achieved. Based on the assessment of recall@5, the proposed ProRE model

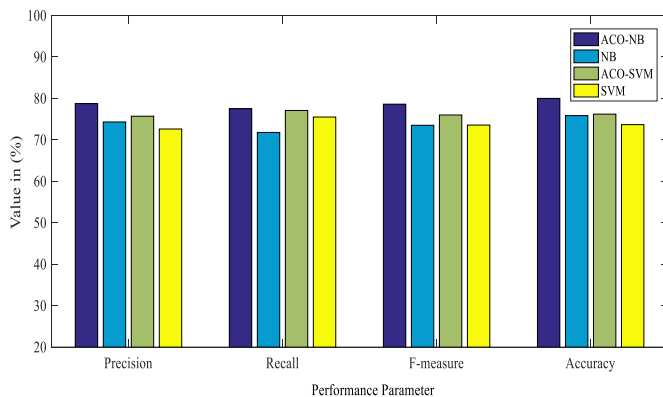


Fig. 9. Bug severity level prediction using ACO-NB, NB, ACO-SVM, and SVM approaches using the Mozilla dataset.

Table 9

Results of Post-hoc test after performing the Friedman test.

Techniques	SVM	NB	ACO-SVM	ACO-NB
SVM	–	S	S	S
NB	S	–	–	S
ACO-SVM	S	–	–	S
ACO-NB	S	S	S	–

Table 10

Simulation results of proposed ProRE model using different bug datasets.

Bug Dataset	Performance Measuring Parameters			
	Recall@5	Recall@10	Precision@5	Precision@10
Eclipse Dataset	48.98	68.73	18.94	15.59
Firefox Dataset	38.66	52.17	25.73	19.55
JBoss Dataset	42.14	55.87	19.71	15.53
OpenFOAM Dataset	48.86	59.18	22.88	18.09
Mozilla Dataset	54.17	62.04	18.76	15.93

has a high recall@5 percentage with the Mozilla bug dataset, 54.17, and a lower recall@5 percentage with the Firefox bug dataset, 38.66. Further, it is noticed that the proposed recommendation system gives better precision results using precision@10 rather than precision@5. It is observed that optimum precision@5 results are obtained for the Firefox dataset.

In contrast, Mozilla datasets exhibit the worst precision results among all datasets. It is also seen that the proposed ProRE model provides a similar precision@10 rate, i.e., 15.59 and 15.93 for Eclipse and Mozilla datasets. The higher recall@10 rate obtains for the Firefox dataset, i.e., 19.55 among all datasets. The proposed programmer identification system results are compared with other existing recommendation systems using Eclipse and Firefox datasets. These systems are ReComm, Drex-Frequency, Drex-Outdegree, Bugzie, and Dretom. Drex recommendation system is based on k-nearest-neighbor search and expertise ranking. Bugzie recommender system is based on fuzzy set theory and cache method. It is developed for bug triage. The Dretom recommendation system is based on topic models and designed for bug resolutions. The results are compared using recall and improvement parameters (Table 10).

The proposed recommendation system and other current systems are shown in Table 11 by using the recall@5 and perhaps improvement (imp) variables. The proposed recommendation system outperforms other current systems in terms of performance. Eclipse and Firefox bug datasets have recall@5 rates of 49.68 and 36.76, respectively. Additionally, it is shown that the Dretom system is functioning poorly compared to all other recommender systems. It has also been observed that the ProRE recommendation

Table 11

Experimental outcomes on Eclipse and Firefox bug datasets (BD).

Recommendation system	Parameter	Eclipse BD.	Firefox BD.
Proposed ProRE	recall@5	49.68	36.76
	recall@5	42.85	27.19
ReComm Method	improvement(%)	19.33	38.26
	recall@5	38.96	23.75
Drex-Frequency Method	improvement(%)	33.98	66.16
	recall@5	35.04	23.23
Drex-Outdegree Method	improvement (%)	38.19	71.29
	recall@5	18.98	15.17
Bugzie Method	improvement(%)	81.87	62.69
	recall@5	14.7	14.89
Dretom	improvement(%)	75.06	72.05

Table 12

Experimental outcomes on Eclipse and Firefox bug datasets.

Recommendation system	Parameter	Eclipse BD.	Firefox BD.
Proposed ProRE method	recall@10	67.83	51.07
	recall@10	56.41	40.63
ReComm method	Improvement (%)	20.24	25.7
	recall@10	47.98	35.56
Drex-Frequency method	improvement(%)	41.37	43.62
	recall@10	47.75	35.95
Drex-Outdegree method	improvement(%)	42.05	42.06
	recall@10	30.39	25.02
Bugzie method	improvement(%)	123.2	104.12
	recall@10	24.45	24.99
Dretom	improvement(%)	77.42	74.36

framework under consideration can dramatically improve computational results.

Table 12 indicates the computational results for the proposed ProRE model and other existing systems utilizing recall@10 & improvement (imp). Compared with other recommender systems, the proposed ProRE model accomplishes a higher recall@10 percentage, 67.83 and 51.07 for the Eclipse & Firefox bug datasets, respectively. Additionally, it is claimed that the Dretom framework yields unsatisfactory outcomes compared to other recommendations. Additionally, it was discovered that the suggested system considerably raises the rate of bug assignment recommendations.

The outcomes of simulations using precision@5 & improvement (imp) criteria for the proposed ProRE framework and other recommendation systems are shown in Table 13. It has been discovered that the proposed system, which generates the use of the two datasets, offers more significant precision than other existing systems. Additionally, the proposed ProRE model produces improved precision@5 outcomes for the Firefox bug dataset (24.73) compared to the Eclipse bug dataset (19.34). The Dretom & Bugzie systems, with precision@5 outcomes of 14.18 and 14.44, respectively, indicate the worst outcomes. The precision@5 percentage using the suggested system is confirmed to have improved significantly.

Table 14 summarizes the simulation results for the proposed model and other recommender systems. The effectiveness of the proposed ProRE model and other recommendation systems utilizing precision@5 & precision@10 criteria differs significantly. Compared to other existing systems, the proposed ProRE model offers better outcomes. Comparatively, the Dretom system performs worse across all datasets. A few machine learning-based recommenders are also evaluated by comparing the effectiveness of the proposed ProRE recommendation system. The accuracy parameter is also considered when assessing these systems' effectiveness. On datasets from Mozilla, OpenFOAM, Eclipse, and Firefox, these recommendation schemes' effectiveness is measured.

The results of the proposed ProRE framework, SVM, NB, and C4.5 methods are shown in Table 15. Compared to other recom-

Table 13

Experimental outcomes on Eclipse and Firefox BD.

Recommendation system	Parameter	Eclipse BD.	Firefox BD.
Proposed ProRE method	precision@5	19.34	24.73
	precision@5	16.91	21.17
ReComm method	improvement(%)	14.37	16.82
	precision@5	16.79	20.87
Drex-Frequency method	improvement(%)	15.19	18.5
	precision@5	15.27	20.21
Drex-Outdegree method	improvement(%)	26.65	22.37
	precision@5	9.8	14.44
Bugzie method	improvement (%)	97.35	71.26
	precision@5	7.97	14.18
Dretom	improvement(%)	71.66	76.4

Table 14

Experimental outcomes on Eclipse and Firefox bug dataset BD.

Recommendation system	Parameter	Eclipse	Firefox	
Proposed ProRE method	precision@10	13.69	18.45	
	ReComm method	precision@10	11.82	16.34
	improvement(%)	15.82	12.91	
Drex-Frequency method	precision@10	10.23	15.16	
	improvement(%)	33.82	21.7	
	precision@10	10.17	15	
Drex-Outdegree method	improvement(%)	34.61	23	
	precision@10	8.69	12.24	
	improvement(%)	57.54	50.74	
Bugzie method	precision@10	6.68	11.97	
	improvement(%)	104.94	54.14	

Table 15

Experimental outcomes using accuracy parameter for proposed ProRE, and other existing systems.

Bug Dataset	NB	SVM	C4.5	Proposed ProRE
Eclipse BD	66.32	78.31	74.65	81.26
Firefox BD	71.03	77.14	66.83	79.24
OpenFOAM BD	68.99	78.54	67.82	80.04
Mozilla BD	62.51	73.29	68.51	77.93

recommendations, the proposed ProRE model produces outcomes of superior quality. For the Eclipse Bug dataset, the proposed ProRE model achieves a greater average accuracy of 81.26. In contrast, the Mozilla Bug dataset outputs a lower accuracy rate of 77.93. The outcomes for the Eclipse & Mozilla bug datasets are less favorable for the NB method. In contrast, the outcomes for the Firefox & OPEN Foam bug datasets are worse for C4.5. Therefore, it can be said that the proposed bug assignment system produces more cost-effective and efficient outcomes than the algorithms.

5. Conclusion

This research provides a framework for bug assessment exercises for programmers. The proposed ProRE model aims to identify the right programmers who can assign and fix the bugs. The proposed ProRE model consists of four stages that make up how it operates. The proposed ProRE model incorporates a feature weighting method focused on ACO. Finding the pertinent features for severity forecasting is the goal of the feature weighting method. Consequently, three performance indicators, i.e., capability ranking, bug severities, and mean Bug fixing time probably, are built to find the suitable programmer for Bug solving. All these performance measures aim to rank the programmers, so each programmer's mean rank has been determined. The average Bug fixing measurement allocates a low rating, whereas severity level measurements allocate greater importance than others. The proposed ProRE model also utilizes NB and SVM classification algorithms to predict the various levels of bug intensity. To assess the effectiveness of the proposed ProRE framework, two experiments have been taken into consideration. The initial experiment predicted bug intensity levels using ACO-NB & ACO-SVM methods.

In contrast to ACO-SVM, SVM, & NB methods, ACO-NB methods demonstrate superior outcomes for predicting the severity of bugs. In the second part of the experiment, the effectiveness of the proposed ProRE model is compared to several other current

recommendations. The proposed ProRE model yields more precise results than other recommendation systems.

The proposed model iteratively refines the pre-trained framework on latest data and several sections of the historic information. It does, however, have a catastrophic memory issue and needs to be improved since the pre-trained framework usually loses a few experiences and understanding. We will not feel that the current method is ideal for dealing with new programmers; however it is a feasible approach.

In future work Considerable investigation is needed to find an improved way to incorporate new programmers into the existing system. We will compared the proposed model with the various machine learning and deep learning based hybrid models in a real time environment.

Declarations

Consent to participate (include appropriate consent statements): All authors provide consent for publication.

Funding

This research was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the grant SP2023/007 conducted by VSB - Technical University of Ostrava, Czechia, and partially supported by Institutional research of the Faculty of Operation and Economics of Transport and Communications – University of Zilina, no. 2/KKMH/2022.

Availability of data and material (data transparency):

The experimental data and results are available with the corresponding author and will be shared based on personal request.

Author contributions

All the authors contributed equally to this research.

CRediT authorship contribution statement

Fatichah Ashima Kukkar: Writing – original draft. **Umesh Kumar Lilhore:** Writing – original, Conceptualization, Supervision. **Jaroslav Frnda:** Formal Analysis. **Jasminder Kaur Sandhu:** Visualization. **Rashmi Prava Das:** Project Administration and Formal Analysis. **Nitin Goyal:** Writing – review & editing. **Arun Kumar:** Data Curation and Validation. **Kamalakanta Muduli:** Investigation. **Filip Rezac:** Formal Analysis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Researcher Yu Zhou from the College of Computer Science, Nanjing University of Aeronautics and Astronautics, for providing the dataset.

Appendix

List of Abbreviations

Acronyms	Details
ACO	Ant Colony Optimization
ACO-NB	Ant Colony Optimization-Naïve Bayes
ACO-SVM	Ant Colony Optimization-Support Vector Machine
BN	Bayesian Network
BTG	Bug Tossing Graph
BTL	Bug Tossing Length
CF	Collaborative Filtering
DBSRNN-A	Deep Bidirectional Recurrent Neural Network with Attention
RPAP	Programmer Performance Assessment Profiles
DRS	Programmer Recommendation System
EM	Expectation maximization
FN	False Negative
FP	False Positive
Imp	Improvement
KNN	K-Nearest Neighbour
KSAP	K-Nearest-Neighbor Search Algorithm and Heterogeneous Proximity
ML	Machine Learning
NB	Naive Bayes
NLP	Natural Language Processing
SN	Social Network
SDLC	Software Development Life Cycle
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
TG	Tossing Graph
TN	True Negative
TP	True Positive

References

- ABSRo, Z.F., REHMAN, S.U., Khushal, D.A.S. and Goswami, A., 2021. An Analysis of Artificial Neural Network for Recommending Programmers to Fix Reported Bugs. *Avrupa Bilim ve Teknoloji Dergisi*, (24), pp.375-379.
- Alazzam, I., Aleroud, A., Al Latifah, Z., Karabatis, G., 2020. Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. *IEEE Trans. Comput. Social Syst.* 7 (5), 1288–1303.
- Aljedaani, W., &Javed, Y. 2018. Bug Reports Evolution in Open Source Systems. In 5th International Symposium on Data Mining Applications (pp. 63-73). Springer, Cham.
- Almhana, R., Kessentini, M., 2021. Considering dependencies between bug reports to improve bugs triage. *Autom. Softw. Eng.* 28 (1), 1–26.
- Angel, T.S., Kumar, G.S., Sehgal, V.M., Nayak, G., 2018. Effective Bug Processing and Tracking System. *J. Comput. Theor. Nanosci.* 15 (8), 2604–2606.
- Batool, I., Khan, T.A., 2022. Software fault prediction using data mining, machine learning, and deep learning techniques: A systematic literature review. *Comput. Electr. Eng.* 100, 107886.
- Chandra, P., Lilhore, U.K., Agrawal, N., 2017. Network intrusion detection system based on modified Random forest classifiers for kdd cup-99 and nsl-kdd dataset. *International Research Journal of Engineering and Technology (IRJET)* 4 (08), 786–791.
- Cynthia, S.T., Roy, B. and Mondal, D., 2022. FeBsRuary. Feature Transformation for Improved Software Bug Detection Models. In: 15th Innovations in Software Engineering Conference (pp. 1-10).
- Dorigo, M., Blum, C., 2005. Ant colony optimization theory: A survey. *Theor. Comput. Sci.* 344 (2–3), 243–278.
- Esteves, G., Figueiredo, E., Veloso, A., Viggiano, M., Ziviani, N., 2020. Understanding machine learning software defect predictions. *Autom. Softw. Eng.* 27 (3), 369–392.
- Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., Gyimóthy, T., 2020. An automatically created novel bug dataset and its validation in bug prediction. *J. Syst. Softw.* 169, 110691.
- Garg, A., Lilhore, U.K., Ghosh, P., Prasad, D., Simaiya, S., August. Machine Learning-based Model for Prediction of Student's Performance in Higher Education. In: In 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN). IEEE, pp. 162–168.
- Guo, Z., Sun, Y., Fu, B., Cheng, X., 2022. FeBsRuary. Research on software defect prediction method based on machine learning oriented to large-scale complex information system. In 4th International Conference on Informatics Engineering & Information Science (ICIEIS2021), Vol. 12161. SPIE, pp. 9–15.
- Gupta, A., Sharma, S., Goyal, S., Rashid, M., June. Novel xgboost tuned machine learning model for software bug prediction. In: In 2020 International Conference on Intelligent Engineering and Management (ICIEM). IEEE, pp. 376–380.
- Hammouri, A., Hammad, M., Alnabhan, M., Alsarayrah, F., 2018. Software bug prediction using machine learning approach. *Int. J. Adv. Comput. Sci. Appl.* 9 (2), 78–83.
- Jadhav, A., Jadhav, K., Bhalerao, A., Kharade, A., 2015. A Survey on Software Data Reduction Techniques for Effective Bug Triage. *IJCSIT on Computer Science and Information Technologies* 6 (5), 4611–4612.
- Jahanshahi, H., Cevik, M., Navas-Sú, J., Başar, A., González-Torres, A., 2022. Wayback Machine: A tool to capture the evolutionary behavior of the bug reports and their triage process in open-source software systems. *J. Syst. Softw.* 189, 111308.
- Jaiswal, R., Sahare, M., Lilhore, U., 2018. In: January. Genetic approach based bug triage for sequencing the instance and features. IEEE, pp. 1–7.
- Jiang, L., Li, C., Wang, S., Zhang, L., 2016. Deep feature weighting for naive Bayes and its application to text classification. *Eng. Appl. Artif. Intel.* 52, 26–39.
- Jorayeva, M., Akbulut, A., Catal, C., Mishra, A., 2022. Machine learning-based software defect prediction for mobile applications: A systematic literature review. *Sensors* 22 (7), 2551.
- Kaur, A., Kaur, K., Chopra, D., 2017. An empirical study of software entropy based bug prediction using machine learning. *International Journal of System Assurance Engineering and Management* 8 (2), 599–616.
- Khleel, N.A.A., Nehéz, K., 2021. Comprehensive Study on Machine Learning Techniques for Software Bug Prediction. *Int. J. Adv. Comput. Sci. Appl.* 12 (8).
- Kumari, A., Agrawal, N. and Lilhore, U., 2018, January. Clustering malicious spam inemail systems using mass mailing. In 2018 2nd International Conference on Inventive Systems and Control (ICISC) (pp. 870-875). IEEE.
- Kumari, M., Sharma, M., Singh, V.B., 2018. Severity assessment of a reported bug by considering its uncertainty and irregular state. *International Journal of Open Source Software and Processes (IJOSSP)* 9 (4), 20–46.
- Lee, D.G., Seo, Y.S., 2019. Systematic Review of Bug Report Processing Techniques to Improve Software Management Performance. *J. Inf. Process. Syst.* 15 (4).
- Lilhore, U.K., Simaiya, S., Chourasia, L., Trivedi, N.K., and Anand, A., 2022. Role of Virtual Reality and Multimedia Computing in Industrial Automation System. *Multimedia Computing Systems and Virtual Reality*, p.231.
- Liu, Y., Qi, X., Zhang, J., Li, H., Ge, X., Ai, J., 2022. Automatic Bug Triaging via Deep Reinforcement Learning. *Appl. Sci.* 12 (7), 3565.
- Malhotra, R., Dabas, A., Hariharasudhan, A.S. and Pant, M., 2021, January. A study on machine learning applied to software bug priority prediction. In 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence) IEEE. (pp. 965-970).
- Mani, S., Sankaran, A., & Aralikatte, R. 2019, January. Deeptrriage: Exploring the effectiveness of deep learning for Bug triaging. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (pp. 171-179). ACM.
- Mo, R., Wei, S., Feng, Q., Li, Z., 2022. An exploratory study of bug prediction at the method level. *Inf. Softw. Technol.* 144, 106794.
- Mohammad, A. H., Alwada'n, T., & Al-Momani, O. 2016. Arabic text categorization using support vector machine, Naïve Bayes and neural network. *GSTF Journal on Computing (JoC)*, 5(1), 108.
- Nath, V., Sheldon, D. and Alphonso-Gibbs, J., 2021, December. Principal Component Analysis and Entropy-based Selection for the Improvement of Bug Triage. In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 541-546).
- Oliveira, P., Andrade, R.M., Barreto, I., Nogueira, T.P. and Bueno, L.M., 2021, June. Issue Auto-Assignment in Software Projects with Machine Learning Techniques. In 2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP) IEEE. (pp. 65-72).
- Onan, A., Korukoğlu, S., Bulut, H., 2016. Ensemble of keyword extraction methods and classifiers in text classification. *Expert Syst. Appl.* 57, 232–247.
- Panda, M., Azar, A.T., 2021. HybridMulti-Objective Grey Wolf Search Optimizer and Machine Learning Approach for Software Bug Prediction. In: Handbook of Research on Modeling, Analysis, and Control of Complex Systems. IGI Global, pp. 314–337.
- Pandey, S.K., Mishra, R.B., Tripathi, A.K., 2020. BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Syst. Appl.* 144, 113085.
- Pombo, N. and Teixeira, R., 2020, October. Contribution of temporal sequence activities to predict bug fixing time. In 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT) (pp. 1-6). IEEE.
- Prabha, C.L., Shivakumar, N., June. Software defect prediction using machine learning techniques. In: In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184). IEEE, pp. 728–733.

- Sahu, K., Lilhore, U.K., Agarwal, N., 2018. An improved data reduction technique based on KNN & NB with hybrid selection method for effective software bugs triage. *Eng Inf Technol* 3 (5), 1835146.
- Sama, M., Pellegrini, P., D'Ariano, A., Rodriguez, J., Pacciarelli, D., 2016. Ant colony optimization for the real-time train routing selection problem. *Transp. Res. B Methodol.* 85, 89–108.
- Shrivastava, P., Lilhore, U. and Agrawal, N., 2017, September. Genetic approach based image retrieval by using CCM and textual features. In 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 364–369). IEEE.
- Subbiah, U., Ramachandran, M., Mahmood, Z., 2019, January. Software engineering approach to bug prediction models using machine learning as a service (MLaaS). In: In ICSE 2018-Proceedings of the 13th International Conference on Software Technologies, pp. 879–887.
- Sun, Y., Dong, W., Chen, Y., 2017. An improved routing algorithm based on ant colony optimization in wireless sensor networks. *IEEE Commun. Lett.* 21 (6), 1317–1320.
- Sun, X., Zhou, T., Wang, R., Duan, Y., Bo, L., Chang, J., 2021. Experience report: investigating bug fixes in machine learning frameworks/libraries. *Front. Comp. Sci.* 15 (6), 1–16.
- Trivedi, N.K., Simaiya, S., Lilhore, U.K., Sharma, S.K., 2020. An efficient credit card fraud detection model based on machine learning methods. *International Journal of Advanced Science and Technology* 29 (5), 3414–3424.
- Uddin, J., Ghazali, R., Deris, M.M., Naseem, R., Shah, H., 2017. A survey on bug prioritization. *Artif. Intell. Rev.* 47 (2), 145–180.
- Vijayarani, S., Ilamathi, M.J., Nithya, M., 2015. Pre-processing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks* 5 (1), 7–16.
- Wang, R., Zhao, H., Lu, B.L., Utiyama, M., Sumita, E., 2015. Bilingual continuous-space language model growing for statistical machine translation. *IEEE/ACM Trans. Audio Speech Lang. Process.* 23 (7), 1209–1220.
- Wu, H., Ma, Y., Xiang, Z., Yang, C., He, K., 2022. A spatial-temporal graph neural network framework for automated software bug triaging. *Knowl.-Based Syst.* 241, 108308.
- Xuan, J., Jiang, H., Ren, Z., Yan, J., & Luo, Z., 2017. Automatic bug triage using semi-supervised text classification. *arXiv preprint arXiv:1704.04769*.
- Xuan, J., Jiang, H., Zhang, H., Ren, Z., 2017. Programmer recommendation on Bug commenting: a ranking approach for the programmer crowd. *SCIENCE CHINA Inf. Sci.* 60, (7) 072105.
- Yadav, A., Singh, S.K., Suri, J.S., 2019. Ranking of software programmers based on expertise score for Bug triaging. *Inf. Softw. Technol.* 112, 1–17.
- Yang, X., Lo, D., Xia, X., Zhang, Y. and Sun, J., 2015, August. Deep learning for just-in-time defect prediction. In 2015 IEEE International Conference on Software Quality, Reliability and Security (pp. 17–26). IEEE.
- Yang, S., Gou, X., Yang, M., Shao, Q., Bian, C., Jiang, M., Qiao, Y., 2022. Software Bug Number Prediction Based on Complex Network Theory and Panel Data Model. *IEEE Trans. Reliab.* 71 (1), 162–177.
- Zaidi, S.F.A., Lee, C.G., January. One-class classification-based bug triage system to assign a newly added programmer. In: In 2021 International Conference on Information Networking (ICOIN). IEEE, pp. 738–741.
- Zaidi, S.F.A., Woo, H., Lee, C.G., 2022. A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports. *IEEE Access* 10, 20677–20689.
- Zhang, T., Jiang, H., Luo, X., Chan, A.T., 2016. A literature review of research in bug resolution: Tasks, challenges, and future directions. *Comput. J.* 59 (5), 741–773.
- Zheng, W., Shen, T., Chen, X., Deng, P., 2022. Interpretability application of the Just-in-Time software defect prediction model. *J. Syst. Softw.* 188, 111245.