BSDRM: A Machine Learning Based Bug Triaging Model To Recommend Developer Team

K. M. Aslam Uddin[1][0000−0001−6918−2648], Md Kowsher[2][0000−0001−5469−3437], and Kazi Sakib[1]

[1] Institute of Information Technology, University of Dhaka, Bangladesh
mph003@iit.du.ac.bd, sakib@iit.du.ac.bd
[2] Department of Computer Science, Stevens Institute of Technology, Hoboken, New Jersey, United States
ga.kowsher@gmail.com

Abstract. Bug triage assigns bugs to developers by mapping properties with past experiences. Without this process, many experienced developers may overwhelm with more assignments, whereas new developers are underwhelmed. Therefore, we propose a machine learning bug triaging approach called Bug Solving Developer Recommendation Model (BSDRM) to solve this issue by assigning developers. We first gather several datasets to combine and split them into train and test sets. Then, we create a sentence-embedded model from training set and generate a bag of developer's words. Instead, test set is transformed into a vocabulary list using an embedded model. BSDRM recommends some eligible developers by matching the developer's bag of words and vocabulary list of bug reports using K-Nearest Neighbour (KNN). A developer classification model categorizes these developers into experienced, newly experienced, and fresh graduate developers. It consists of Decision Tree (DT), Extra Tree (ET), AdaBoost (AdC), Bagging Classifier (BC), Gradient Boosting (GB), KNN, Nearest Centroid (NC), Bernoulli Naïve Bayes (BNB), Multinomial Naïve Bayes (MNB), Complement Naïve Bayes (CoNB), Gaussian Naïve Bayes (GNB), Logistic Regression (LR), Perceptron (Pr), and Multi-Layer Perceptron (MLP) produces individual outcomes of these classifiers. In this case, BC shows 96.59% accuracy to categorize developers with different experience levels. Based on these results, a developer team is recommended to solve testing bugs.

Keywords: Bug Triage · Machine Learning · BSDRM · Bug Report · Developer.

1 Introduction

Bug triage is a process where each bug is prioritized based on its severity, frequency, risk etc. This procedure is helpful to justify different severities of bugs and assign appropriate resources for fixing them. It reduces time to enhance the quality of software. There are many software companies that continuously

generate a massive amount of bug reports. When a new bug report is found, several skilled developers are worked on this bug to fix it. In this case, experienced developers are overwhelmed by the assignment of numerous bugs. Instead, new and mid-skilled developers do not get proper opportunities to fix sufficient amounts of bugs. Sometimes, many experienced developers come from other fields to perform this job. Therefore, it is required to assign bugs to different kinds of developers where they get knowledge and experience in bug solving.

We explore available information of bug reports such as severity, priority, source code, and commit logs to find appropriate developers for analyzing their issues. Recent commits are helpful to get the current activities of developers. Instead, previous records are used to define a developer's proficiency for a particular type of bug. Nonetheless, many fresh graduates (i.e., who have no experience in fixing any bugs) join in the company. In most cases, they are not used to doing such tasks before. On the other hand, some developers gather certain experiences to solve various types of bugs. Besides, developers who switch from another development project are assigned to fix received bugs. Both of these groups are treated as new experienced developers. However, they do not get more bugs to enhance their qualities. In these circumstances, experienced developers become overloaded and unable to fix numerous bugs in the expected time frame. Thus, freshers and new experienced developers are required to engage in the bug fixing tasks. If we build a developer team with experienced, new professionals, fresh graduates, all types of developers are got chance to solve recently arrived bugs. In addition, this team is more useful to solve newly arrived bugs in collaborative ways. As a result, skilled developers are overlooked more bugs and newly joined developers can learn how to solve them.

Numerous methods have been proposed in previous works to understand the impact of bug assignments. Baysal et al. [4] provided a bug triage method that combined different developers' proficiency. They identified expert developers to analyze historical records and suggested required distribution. However, this method is not considered for new fixers. Hu et al. [9] proposed a method called BugFixer, which built a Developer Component Bug (DCB) network using historical bug reports. To prioritize high relevance between the chronological list and new bugs, they recommended a developer list. However, their model does not provide any association of new developers. Zhang et al. [35] represented a team assignment approach called K-nearest-neighbor Search And heterogeneous Proximity (KSAP). This model creates an assorted network from previous bug reports and assigns a team based on their association. The critical drawback of this work is to over-prioritize individual developers' antique actions and does not recommend about new developers assign bugs. Khatun et al. [11] anticipated a Team Allocation technique to ensure bug assignment for both existing and new developers (TAEN). They generated a heterogeneous network from previous bug reports. It suggests a team of N developers based on TAEN score and current workloads. Nevertheless, they do not specify newly experienced developers in their work.

Machine learning is a study of algorithms that automatically improves to take a particular decision by investigating previous records. In this work, we propose a machine learning-based model called Bug Solving Developer Recommendation Model (BSDRM), which recommends a developer team combining experts, medium fixers or skilled developers of other domains, and fresh graduates to solve newly arrival bugs. In this model, all datasets are merged and created into a developer matrix. We divide this matrix into training and testing sets. Also, the training set is split into two parts. Then, we fit one part of this training set with a sentence embedding model that generates a bag of developers' words. Instead, a developer classifier is fitted with another training set that contains various classifiers like DT, ET, AdC, BC, GB, KNN, NC, BNB, MNB, CoNB, GNB, LR, Pr, and MLP. When we evaluate several new bug reports/testing instances, this embedded model generates a vocabulary list from them. Then, eligible developers are extracted by matching the bag of developers' words and testing the vocabulary list using an unsupervised KNN finder. The developer classifier is used to categorize these developers into Experienced Developer (ED), New Experienced Developer (NED), and Fresh Graduate (FG). Then, we combine these developer groups and make a bug-fixing team.

Eclipse, Mozilla, and Netbeans are used in a case study of open source projects to assess BSDRM. The data matrix is manually weighed and labeled with ED, NED, and FG. When a group of developers is identified, individual classifiers are employed to classify them as ED, NED, and FG. Thus, BC shows the highest accuracy (96.59%), and F-measure (96.59%) in this work.

2 Literature Review

Due to the significance of various bug assignment procedures, existing techniques are categorized into text exploration, tossing graph, cost-aware, activity, source, fuzzy, time, relevant search, and novice developer recommendation model. These models are briefly described as follows.

Text exploration approaches are investigated previous records and predicted the rank of developers [1, 3, 4, 9, 15, 18]. Zhang et al. [34] suggested a BUg triage by topic modeling and heTERogeneous network analysis approach (BUTTER), which automatically assigns bugs to the developers. There are included both textual and structural data in bug reports. It employs Latent Dirichlet Allocation (LDA) to categorize texts into various topics. Also, a heterogeneous network is used to collect structural data for developers. Then, the RankClass model is trained based on the topic model and structural data. When a new bug report comes, LDA and RankClass are used to classify the final topics of the bug. This distribution provides a developer list to be assigned for a certain bug. However, this approach does not consider new developers. Hu et al. [9] proposed a bug triaging method called BugFixer that builds based on the past bug-fixing information. It creates a Developer-Component-Bug (DCB) network to connect developers, components, features, and related bugs. BugFixer computes the sim-

ilarity between the new and existing bugs through the DCB network. Then, it depicts the ranked list of developers. Again, it does not consider new developers.

Tossy graph/length-based techniques recommend by developers for fixing bugs [2, 5, 10, 27]. Jeong et al. [10] proposed a graph model which captures the tossing probabilities among developers from tossing history. It produces a low-qualified developer list and does not mention new developers. Yadav et al. [29] improved an algorithm for decreasing bug tossing length based on the Developer Skills Record (DSR). They created a two-step model where DSR is made for each developer from any five datasets. In the second step, they listed all competent developers who provide accurate predictions and effectively reduce bug tossing length. However, this model does not ponder new developers.

Park et al. [19] developed a cost-aware developer ranking algorithm named CosTriage. The authors noticed a problem related to Content-Based Recommendation (CBR), which offers only experienced developers and solved associated bugs. For this reason, they are overloaded, and the bug-solving process becomes slower. In CosTriage, it optimizes performance (i.e., accuracy and cost) and combines a Collaborative Filtering Recommender (CFR) with CBR. It adopts content boosted collaborative filtering (CBCF) for ranking developers. The authors only considered previous bug history; thus, CosTriage fails to assign newly-appointed developers. Naguib et al. [17] proposed an activity-based bug triaging strategy which classifies existing bug reports into topics using LDA and creates an activity profile for each developer. When a new bug report has arrived, this strategy ranks the developer list based on the relevance between individual developer profiles and the topics of this bug report. Moreover, it does not consider new developers.

Source-based bug assignment techniques were proposed [6,12,15] to fix a new bug. Matter et al. [15] suggested a source-based expertize model called DEVELECT (i.e., a portmanteau word of developer and dialect) that parses source code and version history for recommending developers. It extracts a bag of words from each source code contributor. This model is trained with current vocabularies and created by a team-author matrix. It checks lexical similarities and developer keywords in a new report. However, this technique is hugely dependent on experienced developers. It also increases bug re-assignments, and delays and lessens recommendation accuracy. Mani et al. [14] used a Deep Bi-directional Recurrent Neural Network with Attention (DBRNN-A) which learns semantic and synthetic features from a long word sequence. Further, different bug reports are analyzed where DBRNN-A provides a higher rank-10 average accuracy using MNB, Cosine Distance, SVM [13], and softmax classifier. It preserves a word's order and semantic relationship in a more extended context.

Some researchers provided social network-based developer recommendations to accelerate this work [8,33]. Zhang et al. [32] proposed an integrated bug triage algorithm to improve the social network developer recommendation process. At first, a probability model analyzes its social network and determines candidate developers. Later, the experience model investigates a number of fixed bugs

and solves each developer's cost as the estimated factor. It uses the smoothed Unigram Model (UM) and recommends appropriate developers for fixing a new bug. It suggests experienced developers but does not estimate new developers.

Xie et al. [28] proposed a topic-based approach called Developer REcommendation based on TOpic Models (DRETOM), which recommends developers to utilize their bug fixing history. The authors used the Topic Modeling Toolbox (TMT) to create a topic from existing bug reports. A probabilistic model calculates a developer's problem-solving probability based on interest and expertise. It determines the relationship between developers and bug reports. When a new bug report appears, different topics are determined by previous topic models. Then, DRETOM calculates the probability of each developer and ranks the developers in descending order. Finally, top K-developers are recommended to fix new bug reports. Again, this method does not consider new developers. Tamrawi et al. [25] proposed Bugzie which is fuzzy set-based modeling of developers for bug-fixing skills. The fuzzy set determines relevant developers who are suited to solve bugs. The membership function of a developer belongs to a fuzzy set. Also, it is computed with several terms derived from solved bug reports. When a new report is resolved, this function is updated. In this situation, these terms are extracted and recommended by a developer-ranked list based on their membership scores. This method urges developers to use past fixing history. However, it does not indicate newly appointed developers.

Time-based assignment techniques are proposed in the following [16, 23]. Shokripour et al. [23] formulated a time-metadata method where all source code entities are parsed and connected with contributors to construct a corpus. When a new bug is found, several keywords are searched and provided weighted-based frequency with time metadata. Therefore, the high-scored developers are used at the top list. Khatun et al. [12] suggested a strategy called Bug fixing and Source commit activity-based Bug Assignment (BSBA). It contains three modules: source activity collection module, fixing history collection module, and developer suggestion module. The source activity collection module has constructed an index of the developer's source code activities. Besides, the fixing history collection module is built another index for connecting bug keywords with fixing time. Then, the developer suggestion module is employed to make queries into two indexes with new bug keywords. The top-scored developers are suggested as suitable fixers based on the query results.

Various types of techniques were used to suggest developers for solving different bugs. Most of them were trained by previous bug fixing histories. Besides, they extracted relevant topics/keywords by matching developer profiles and newly arrival bug reports. Then, they identified top experts for fixing bugs. In this case, many models are utilized a few information sources that provide shallow performance to identify suitable developers. When many newly generated bug reports arrive, experienced developers are overloaded with them. However, these models are not incorporated more instances of mid-level specialists,

other domain experts, and newly graduated developers. As a result, this model represents less opportunity for them to learn about bug fixing.

3 Materials and Methods

This section explains how BSDRM recommends a qualified development team to fix bugs. Thus, the description of working bug reports and overall team suggestion procedure is given as follows.

3.1 Data Description

Eclipse, Mozilla, and Netbeans are widely used as large-scale open source projects in various bug triage studies. These datasets consist of source code, commit logs and bug reports of these projects from Bugzilla. They contain different types of variables namely severity, priority, fix status, platform or hardware, assignee, single developer, first assignee, developer comments, summary (title), and description. BSDRM uses this dataset to investigate and recommend developers for fixing bugs. We collect 13,7,147 bug reports between 2001 to 2020, 13,2,261 bug reports between 1999 to 2020, and 44,149 bug reports between 2001 and 2017 from Eclipse, Mozilla, and NetBeans, respectively.

3.2 Proposed Bug Solving Developer Recommendation Model (BSDRM)

In this section, we propose a machine learning-based bug triage approach called BSDRM which recommends a developer team with ED, NED and FG developers. Fig. 1 represents the working steps of this model which is briefly described as follows.

- **Generating Developer Matrix:** We combine Eclipse, Mozilla, and NetBeans datasets where 56621 instances are gathered. This dataset is called the developer matrix. Each row represents each developer instance. However, they are not categorized/labeled according to their expertise. Thus, we consider three categories such as ED, NED, and FG to assess their skills. The brief description of these categories are defined as follows.
  1. **ED profile:** Developers who fix a large number of bugs as single developers, comment on various issues, assign as first developers into many bugs, and solve more prioritized and critical bugs are considered as ED. This work finds ED who fixes at least 200 bugs, comments on 1000 or more issues and assigns around 500 bugs as first developers.
  2. **NED profile:** If a fixer solves a moderate amount of bugs, comments on some issues, assigns as the first developer in some bugs, they are called NED. Again, some developers are experts in another software domain. If they are now working on bug fixing tasks, these developers are also

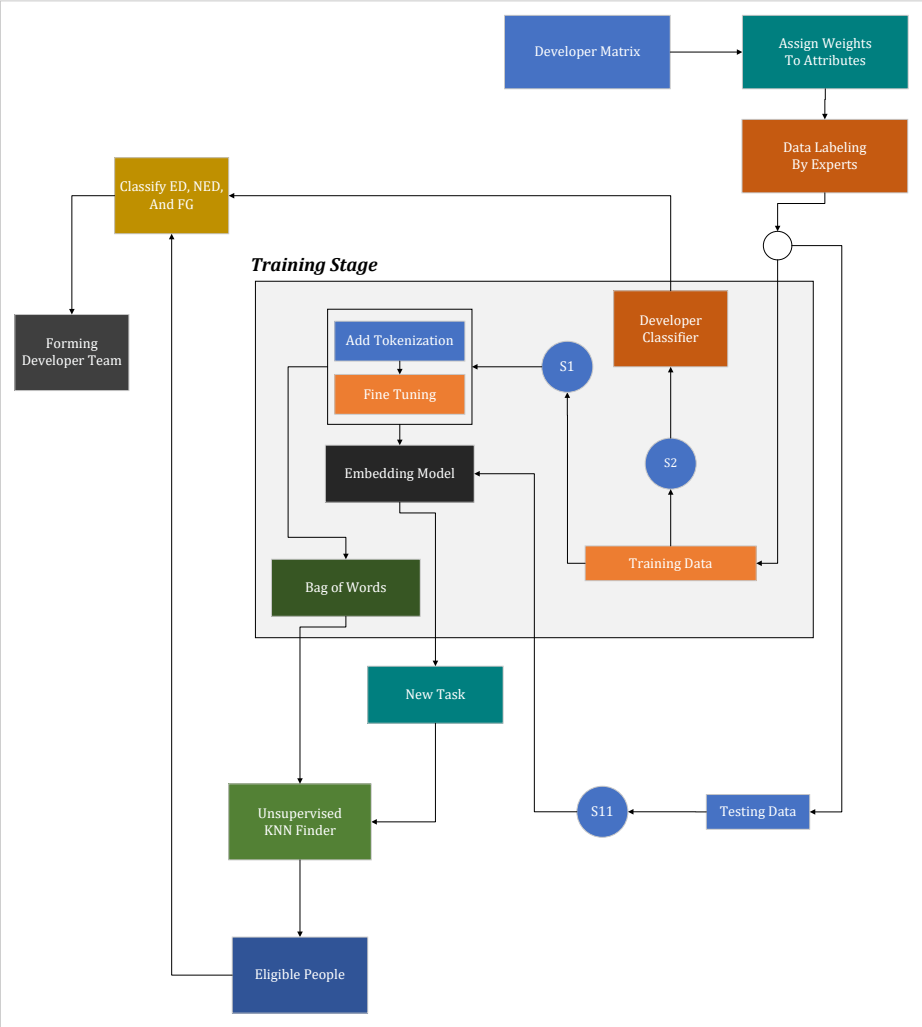

Fig. 1: The Workflow Diagram of BSDRM, Machine Learning based Developer Recommendation Model for Bug Triage

considered as NED. This work defines NED which solves almost 100-200 bugs, comments on less than 1000 bug reports, and is assigned less than 500 as first developers.

3. **FG profile:** Developers who have just passed from academia and started their career in bug fixing are taken as FG. In these circumstances, they do not have enough working experience.So, their histories are not available in the existing bug repository. The authority can give a predefined form

to assess FGs skills such as academic knowledge, technical expertise, interests, developed projects, etc.

According to their importance, we manually assign weights to individual features and determined every developer's expert level [7, 24, 30]. We also emphasize more on human inspection rather than quantitative approaches. This labeled dataset is reviewed by some specialists to assess this manipulation. Then, it is split into 80% training and 20% testing sets for further analysis. In the training set, we divide it into two subsets, where one of them contains a summary and description of this dataset called Subset-1 (S1). Another subset includes the rest of the attributes- like severity, priority, fix status, platform or hardware, assignee, single developer, first assignee, commenter, and developers-called Subset-2 (S2). Besides, we extract another Subset-11 (S11) from the test set that contains a summary and description of this dataset.

– **Training Stage** In this section, we gather S1 and S2 to train sentence-embedded models of different classifiers. This stage is basically conducted different training processes that help to create a suitable developer team. Several steps of this session are briefly described as follows.

  • **Sentence Embedding:** At first, a fine-tuning process is employed on S1 by pretrained Bidirectional Encoder Representations from Transformers (BERT) and created a sentence embedding model. In this case, this model generates a bag of the developer's words by preserving word contexts of S1. If any unknown words are found on tokenizers, add them to the BERT tokenizer.

  • **Data Balancing:** S2 represents an imbalance condition where various types of developers are highly varied. Due to the overfitting issue, different machine learning models are not shown better outcomes. Therefore, we resample and balance this subset where each developer type contains 652 samples.

  • **Employing Different Classifiers:** Then, the following classifiers such as DT, ET, AdC, BC, GB, RF, KNN, NC, BNB, MNB, CoNB, GNB, SVM, SGD, LR, Pr, and MLP are trained with S2. These classifiers configure a developer classifier that categorizes new records based on the experience level.

– **New Task:** When a new bug report is received, it requires assigning several developers and solving this bug. In this section, we utilize S11 which are generated from 20% of testing reports. Then, we apply the pre-trained sentence embedding model into S11 and extract a vocabulary list for developers.

– **Exploring Eligible Developer** An unsupervised KNN finder is employed to match the vocabulary list of testing reports with the existing bag of developer's words. It identifies K nearest developers who are eligible to resolve this particular bug.

- **Classifying Developers:** In this step, we predict the experience level of K-relevant developers employing the developer classifier. The performance of this model is manipulated by the outcomes of its individual classifiers. These results are generated based on Accuracy (Accu), Precision (Prec), Recall (Rec), F1-Score (FS1), Hamming Loss (HL), Jaccard Score (JS), Matthews Correlation Coefficient (MCC), Area Under the Curve (AUC), Balanced Accuracy (BA), and Cohen's Kappa Score (CKS).
- **Forming Developer Team:** Each type of developer are gathered based on the outcomes of the best classifier. Thus, we create a developer team with eligible persons for solving different bugs.

3.3 Evaluation Matrix

We engender a confusion matrix to measure derived results and specify the number of accurate and inappropriate samples, respectively. However, this matrix is formulated with true positive (TP), false positive (FP), false negative (FN), and true negative (TN) values. Different evaluation metrics such as Accuracy, Precision, Recall, F1-Score, Hamming Loss, Jaccard Score, Matthews Correlation Coefficient, Area Under the Curve, Balanced Accuracy and Cohen's Kappa Score, are used to justify the classification results of individual classifiers. A brief description of these metrics is given as follows.

- **Accuracy (Accu)** measures the efficiency of classifiers using this following equation:

$$\text{Accu} = \frac{TP + TN}{(TP + TN + FP + FN)} \tag{1}$$

- **Precision (Prec)** represents the ratio of TPs and all positive predicted values.

$$\text{Prec} = \frac{TP}{(TP + FP)} \tag{2}$$

- **Recall (Rec)** provides the ratio of TP and all actual positive values.

$$\text{Rec} = \frac{TP}{(TP + FN)} \tag{3}$$

- **F1-Score (FS1)** manipulates the harmonic means of precision and recall which is defined as follows:

$$\text{FS1} = \frac{2TP}{2TP + FP + FN} \tag{4}$$

- **Hamming Loss (HL)** is the fraction of wrong labels and the total number of labels.

$$\text{HL} = \frac{\sum_{j=1}^{|L|} xor(I^j - p^j)}{|L|} \tag{5}$$

– **Jaccard Score (JS)** defines the intersection of the predicted and the true labels divided by the union of the predicted labels and the true labels.

$$\text{JS} = \frac{A \cap B}{A \cup B} \tag{6}$$

– **Matthews Correlation Coefficient (MCC)** is a reliable statistical rate that provides a high score and predicts good results in all confusion matrix categories.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{7}$$

– **Area Under the Curve (AUC)** characterizes how well positive classes are isolated from negative classes. It can be represented with TP rate (TPR) and TN rate (TNR) by the following equations:

$$\text{AUC} = \frac{TPR + TNR}{2} \tag{8}$$

– **Balanced Accuracy (BA)** is evaluated for how good a binary classifier is. In addition, when the classes are imbalanced, it is beneficial.

$$\text{BA} = \frac{1}{2} \times \frac{TP}{(TP + FN)} + \frac{1}{2} \times \frac{TN}{(TN + FP)} \tag{9}$$

– **Cohen's Kappa Score (CKS)** is used to manipulate the inter-rater reliability of qualitative items.

$$\text{CKS} = \frac{N \sum_{i=1}^{m} CM_{ii} - \sum_{i=1}^{m} Ci_{Corr} Ci_{Pre}}{N^2 - \sum_{i=1}^{m} Ci_{Corr} Ci_{Pre}} \tag{10}$$

4 Experiment Result and Discussion

In this work, we implement pretrained Bidirectional Encoder Representations from Transformers (BERT) of Google employing for sentence embedding process [21, 26]. Then, resample method is used to balance instances of S2 using the sci-kit learn package [20, 22]. Numerous classifiers such as DT, ET, AdC, BC, GBC, RF, KNN, NC, BNB, MNB, CoNB, GNB, SVM, SGD, LR, Pr, and MLP are employed to identify different types of developers. Again, we utilize the sci-kit learn library to get these classifiers. All experiments are held in Google Colaboratory using Python. Table 1 shows the experiment results of individual classifiers. Here, we compare the performance of the different classifiers which are evaluated based on Accu, Prec, Rec, FS1, HL, JS, MC, AUC, BAC, and CKS.

After observing the results in Table 1, we observe that BC shows the best outcomes (i.e., except AUC) for individual classifiers. Besides, ET provides the best AUC among all classifiers. Moreover, RF and GBC obtain better accuracy

Table 1: Developer Classification Result

Classifier	Accu	Prec	Rec	FS1	HL	JS	MC	AUC	BAC	CKS
DT	95.74	95.75	95.77	95.75	4.26	91.90	93.61	98.85	95.77	93.61
ET	94.89	94.95	94.96	94.95	5.11	90.49	92.34	**99.23**	94.96	92.33
AdC	68.82	74.41	69.80	67.17	31.18	53.59	56.81	95.35	69.80	53.52
BC	**96.59**	**96.62**	**96.56**	**96.59**	**3.41**	**93.42**	**94.89**	98.73	**96.56**	**94.88**
GB	96.42	96.45	96.41	96.43	3.58	93.14	94.63	98.85	96.41	94.63
RF	96.42	96.48	96.40	96.43	3.58	93.15	94.63	98.99	96.40	94.63
KNN	88.93	88.92	88.90	88.82	11.07	80.12	83.47	94.24	88.90	83.39
NC	35.95	51.62	36.52	34.03	64.05	20.74	5.62	48.80	36.52	4.98
BNB	47.02	55.79	47.89	42.45	52.98	27.60	27.20	66.25	47.89	21.40
MNB	46.51	50.72	47.28	39.80	53.49	26.31	26.12	68.34	47.28	20.54
CoNB	44.97	31.75	45.65	34.58	55.03	23.56	24.31	67.50	45.65	18.16
GNB	79.39	83.12	80.13	79.06	20.61	67.78	71.55	99.00	80.13	69.19
SVM	69.85	74.62	69.86	70.52	30.15	54.76	56.35	83.60	69.86	54.84
SGD	83.65	85.64	83.53	82.27	16.35	71.02	77.18	95.71	83.53	75.41
LR	54.86	60.64	54.77	55.95	45.14	40.26	33.53	66.70	54.77	32.62
Pr	78.02	79.82	77.51	77.44	21.98	64.74	68.01	86.77	77.51	66.96
MLP	87.56	88.08	87.46	87.09	12.44	77.86	81.97	92.71	87.46	81.35

(96.42%) to classify developers. DT, ET, and AdC represent good outcomes (i.e., almost all of them are shown greater than 90% results). Also, KNN, SGB, and MLP show their all outcomes within 80%. GNB and Pr give their outcomes around 70%. Instead, the rest of the classifiers are not shown good outcomes in identifying developers. Finally, we determine BC as the best classifier to detect developer experience levels concerning all evaluation matrices.

However, several bug assignment works happened where they determined developer skills based on the previously solved bug reports [10, 12, 31]. A few studies were considered NED (i.e., mid-level, an expert in another domain) and FG for bug-solving tasks [11]. In this work, these types of developers are included along with ED and created a team to fix newly arrival bugs. NED and FG developers who engage in the bug triaging process fix and learn more about it. BSDRM automatically assigns bugs to the appropriate developers. So, it has the lower possibility to reassign the same bugs to similar developers. To compare this model with existing works, we check three criteria that distinguishes how BSDRM provides better outcomes than previous works. These criteria are given as follows:

- C1-Does BSDRM estimate the bug fixing history of a developer while assigning a new task?
- C2-Does BSDRM consider the interest of a developer in the task allocation scheme?
- C3-Does the assigned team consists of developers from all experience level?

Table 2: Comparison of BSDRM with Traditional Models

	Core Concept	C1	C2	C3
BugFixer [9]	Text exploration	Yes	No	No
Jeong et al. [10]	Tossing graph	Yes	No	No
Yadav et al. [29]	Tossing length	Yes	Yes	No
CosTriage [19]	Cost aware ranking	Yes	No	No
DEVELECT [15]	Source code based approach	Yes	No	No
Zhang et al. [34]	Social network based	Yes	Yes	No
DRETOM [28]	Topic model based approach	Yes	Yes	No
Shokripour et al. [23]	Time based approach	Yes	No	No
Khatun et al. [12]	Time based approach	Yes	No	No
TEAN [11]	LDA	Yes	Yes	No
BSDRM	**Proposed Model**	Yes	Yes	Yes

Table 2 shows the comparison of BSDRM with existing models. In this situation, we observe that BSDRM identifies experts based on their bug fixing histories (i.e., resolved C1) to assign a new task like previous models. Besides, many models [11, 28, 29, 34] were considered developers interested to solve new bugs. Again, BSDRM identifies this matter and fixes C2. BSDRM and TEAN ensure a heterogeneous team formation with different types of developers [11]. However, TEAN does not esteem mid or other field experts of the different organizations (i.e., not properly fixed C4). In BSDRM, we consider three types of developer (i.e., ED, NED and FG) whose build a more robust and stable team to solve bugs based on experience level. Thus, this model also resolved C4. According to Table 2, BSDRM responses all of these criterias to fix a bug by creating a suitable developer team. Nevertheless, it is not always properly estimated how many ED, NED, and FG developers are classified by BSDRM.

5 Conclusion

In this work, we implement BSDRM to divide the developer matrix into training and test sets. The training set is split into S1 and S2. We generate a bag of words from S1 using the sentence embedding process. Various classifiers are used to build a developer classifier; then it is fitted with S2. Also, we extract a vocabulary list from S11 using the trained embedded model. Further, eligible developers are determined by matching bag of words of S1 and the vocabulary list of S11. Using a developer classifier, we categorize this group into ED, NED, and FGs which creates a suitable bug-fixing team. Therefore, BSDRM recommends this team with mixed categorized fixers to solve newly arrival bugs in collaborative ways. After experimental analysis, BC is found as the most stable classifier to categorize different developers according to their experience label. It shows the highest 96.59% accuracy, 96.62% prec, 96.56% rec, 96.59 % FS1, 3.41 % HL, 93.42% JS, 94.89% MCC, 96.56% BA, and 94.88% CFS, respectively. Besides, GB and RF obtain very close outcomes to BC. Different criteria related

to experience are investigated to assess the functionality of BSDRM. Thus, it reduces the abundant workload of ED where NED and FG are got the opportunity to learn deeply about bug solving.

In the future, we will balance task assignments of individual developers in the recommended developer team. Also, more bug repositories (i.e., large-scale software, open-source, and commercial projects) will be considered to evaluate this task and make this process more dynamic.

References

1. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: Proceedings of the 28th international conference on Software engineering. pp. 361–370 (2006)
2. Baloch, M.Z., Hussain, S., Afzal, H., Mufti, M.R., Ahmad, B.: Software developer recommendation in terms of reducing bug tossing length. In: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. pp. 396–407. Springer (2020)
3. Banitaan, S., Alenezi, M.: Tram: An approach for assigning bug reports using their metadata. In: 2013 Third International Conference on Communications and Information Technology (ICCIT). pp. 215–219. IEEE (2013)
4. Baysal, O., Godfrey, M.W., Cohen, R.: A bug you like: A framework for automated assignment of bugs. In: 2009 IEEE 17th International Conference on Program Comprehension. pp. 297–298. IEEE (2009)
5. Bhattacharya, P., Neamtiu, I.: Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: 2010 IEEE International Conference on Software Maintenance. pp. 1–10. IEEE (2010)
6. Goyal, A., Sardana, N.: Feature ranking and aggregation for bug triaging in open-source issue tracking systems. In: 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence). pp. 871–876. IEEE (2021)
7. Grana, C., Bolelli, F., Baraldi, L., Vezzani, R.: Yacclab-yet another connected components labeling benchmark. In: 2016 23rd International Conference on Pattern Recognition (ICPR). pp. 3109–3114. IEEE (2016)
8. Hossain, M.B., Arefin, M.S., Sarker, I.H., Kowsher, M., Dhar, P.K., Koshiba, T.: Caran: A context-aware recency-based attention network for point-of-interest recommendation. IEEE Access **10**, 36299–36310 (2022)
9. Hu, H., Zhang, H., Xuan, J., Sun, W.: Effective bug triage based on historical bug-fix information. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering. pp. 122–132. IEEE (2014)
10. Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 111–120 (2009)
11. Khatun, A.: A team allocation technique ensuring bug assignment to existing and new developers using their recency and expertise (2017)
12. Khatun, A., Sakib, K.: A bug assignment approach combining expertise and recency of both bug fixing and source commits. In: ENASE. pp. 351–358 (2018)
13. Kowsher, M., Hossen, I., Tahabilder, A., Prottasha, N.J., Habib, K., Azmi, Z.R.M.: Support directional shifting vector: A direction based machine learning classifier. Emerging Science Journal **5**(5), 700–713 (2021)

14. Mani, S., Sankaran, A., Aralikatte, R.: Deeptriage: 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 (2019)
15. Matter, D., Kuhn, A., Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers. In: 2009 6th IEEE international working conference on mining software repositories. pp. 131–140. IEEE (2009)
16. Mohan, D., Sardana, N., et al.: Visheshagya: Time based expertise model for bug report assignment. In: 2016 Ninth International Conference on Contemporary Computing (IC3). pp. 1–6. IEEE (2016)
17. Naguib, H., Narayan, N., Brügge, B., Helal, D.: Bug report assignee recommendation using activity profiles. In: 2013 10th Working Conference on Mining Software Repositories (MSR). pp. 22–30. IEEE (2013)
18. Nagwani, N.K., Verma, S.: Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. In: 2011 Ninth International Conference on ICT and Knowledge Engineering. pp. 113–117. IEEE (2012)
19. Park, J.w., Lee, M.W., Kim, J., Hwang, S.w., Kim, S.: Costriage: A cost-aware triage algorithm for bug reporting systems. Proceedings of the AAAI Conference on Artificial Intelligence **25**(1), 139–144 (Aug 2011), https://ojs.aaai.org/index.php/AAAI/article/view/7839
20. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
21. Prottasha, N.J., Sami, A.A., Kowsher, M., Murad, S.A., Bairagi, A.K., Masud, M., Baz, M.: Transfer learning for sentiment analysis using bert based supervised fine-tuning. Sensors **22**(11), 4157 (2022)
22. Satu, M.S., Khan, M.I., Mahmud, M., Uddin, S., Summers, M.A., Quinn, J.M., Moni, M.A.: Tclustvid: a novel machine learning classification model to investigate topics and sentiment in covid-19 tweets. Knowledge-Based Systems p. 107126 (2021)
23. Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.: A time-based approach to automatic bug report assignment. Journal of Systems and Software **102**, 109–122 (2015)
24. Signal, B., Gloss, B.S., Dinger, M.E., Mercer, T.R.: Machine learning annotation of human branchpoints. Bioinformatics **34**(6), 920–927 (2018)
25. Tamrawi, A., Nguyen, T.T., Al-Kofahi, J., Nguyen, T.N.: Fuzzy set-based automatic bug triaging (nier track). In: Proceedings of the 33rd International Conference on Software Engineering. pp. 884–887 (2011)
26. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771 (2019)
27. Wu, H., Liu, H., Ma, Y.: Empirical study on developer factors affecting tossing path length of bug reports. IET Software **12**(3), 258–270 (2018)
28. Xie, X., Zhang, W., Yang, Y., Wang, Q.: Dretom: Developer recommendation based on topic models for bug resolution. In: Proceedings of the 8th international conference on predictive models in software engineering. pp. 19–28 (2012)
29. Yadav, A., Singh, S.K., Suri, J.S.: Ranking of software developers based on expertise score for bug triaging. Information and Software Technology **112**, 1–17 (2019)
30. Ye, Z., Xu, Y., Huang, R., Tong, X., Li, X., Liu, X., Luan, K., Hoegner, L., Stilla, U.: Lasdu: A large-scale aerial lidar dataset for semantic labeling in dense urban areas. ISPRS International Journal of Geo-Information **9**(7), 450 (2020)

31. Zhang, T., Lee, B.: An automated bug triage approach: A concept profile and social network based developer recommendation. In: International Conference on Intelligent Computing. pp. 505–512. Springer (2012)
32. Zhang, T., Lee, B.: A hybrid bug triage algorithm for developer recommendation. In: Proceedings of the 28th annual ACM symposium on applied computing. pp. 1088–1094 (2013)
33. Zhang, T., Yang, G., Lee, B., Lua, E.K.: A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In: 2014 21st Asia-Pacific Software Engineering Conference. vol. 1, pp. 223–230. IEEE (2014)
34. Zhang, W., Han, G., Wang, Q.: Butter: An approach to bug triage with topic modeling and heterogeneous network analysis. In: 2014 International Conference on Cloud Computing and Big Data. pp. 62–69. IEEE (2014)
35. Zhang, W., Wang, S., Wang, Q.: Ksap: An approach to bug report assignment using knn search and heterogeneous proximity. Information and Software Technology **70**, 68–84 (2016)