IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 27, 2018, accepted June 22, 2018, date of publication July 2, 2018, date of current version July 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2850910

Emotion Based Automated Priority Prediction for Bug Reports

QASIM UMER, HUI LIU, AND YASIR SULTAN
School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
Corresponding author: Hui Liu (liuhui08@bit.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1000801 and in part by the National Natural Science Foundation of China under Grant 61472034, Grant 61690205, and Grant 61772071.

ABSTRACT Issue tracking systems allow users to report bugs. Bug reports often contain product name, product component, description, and severity. Based on such information, triagers often manually prioritize the bug reports for investigation. However, manual prioritization is time consuming and cumbersome. DRONE is an automated state-of-the-art approach that recommends the priority level information of the bug reports. However, its performance for all levels of priorities is not uniform and may be improved. To this end, in this paper, we propose an emotion-based automatic approach to predict the priority for a report. First, we exploit natural language processing techniques to preprocess the bug report. Second, we identify the emotion-words that are involved in the description of the bug report and assign it an emotion value. Third, we create a feature vector for the bug report and predict its priority with a machine learning classifier that is trained with history data collected from the Internet. We evaluate the proposed approach on Eclipse open-source projects and the results of the cross-project evaluation suggest that the proposed approach outperforms the state-of-the-art. On average, it improves the F1 score by more than 6%.

INDEX TERMS Bug reports, classification, machine learning, priority prediction, software maintenance.

I. INTRODUCTION

Enterprise and business softwares are often released with bugs due to inadequate testing [1]. In order to improve the next version of the software and to meet the new business requirements, developers allow the users to report bugs using issue tracking systems e.g., Bugzilla [2], Mantis [3], Google Code Issue Tracker [4], GitHub Issue Tracker [5], and JIRA [6]. These bug-tracking systems often rely on unstructured natural language bug descriptions [7]. Users help developers through reporting bugs which is a standard practice in software maintenance process. For instance, resolution of bug reports is one of the most important task [8] in software maintenance process. Resolving bug reports becomes more and more challenging and expansive [7], because bug reports are increasing at an exponential rate as the size and complexity of a software is increasing [9]. It is reported that "On average, Mozilla received 170 and Eclipse 120 new bug reports on each day from January to July 2000" [10].

Triaging bug reports are significantly important for any bug-tracking system. To triage bug reports most of the bug-tracking systems are using an attribute called *priority*. The classification of the priority of bug reports is based on its importance and urgency. In Bugzilla, the priority of a bug report can be defined from P1-P5, where P1 has the top priority level and P5 has the least priority level. Reporters help developers by prioritizing the bug reports. As a result, developers may resolve the most severe bug reports on priority basis [11].

Although reporters assign priority while reporting bugs, in most of the cases, this field is left blank. Because assigning priority to a bug report requires a rich experience and its technical understanding. Therefore, the priority of a bug report may be assigned incorrectly due to different background knowledge of reporters. The missing information and incorrect assignment of priority levels of a bug report may delay its resolution. Consequently, it is manually checked and adjusted by developers. The manual prioritization of bug reports is a time-consuming task and can increase the developers' workload and bug fixing time [12]. Therefore, the decision about priority should be automated.

In order to automate the priority prediction, a number of automated approaches have been proposed to predict the priority of bug reports [1], [13], [14]. However, their performance is not accurate and none of them consider the emotions of the reporter for priority prediction. It is observed

that the count of negative emotions of the reporters in *severe* bug reports is higher than in *non-severe* bug reports.

To this end, in this paper, we propose an emotion-based approach to predict priority of bug reports. To facilitate the automated task, we employ the emotion analysis of bug reports. The emotion words in the summary of a bug report can explain the positive and negative feelings of the reporter.

We extract the history data of bug reports from Bugzilla and apply natural language processing techniques on each bug report to preprocess it. From the preprocessed bug reports, we perform feature modeling to identify the useful features of each bug report for training. As hidden emotion may affect the priority of bug reports, the proposed approach performs an emotion analysis to identify emotions-words from each bug report using emotion-based corpus [10] and assigns it an emotion-value. Finally, we train and test a classifier with the resulting features. The cross-project evaluation suggests that the proposed approach is accurate and improve improves the F1-score that varies from *5.12%* to *7.31%*.

Following are the main contributions:

- An automated emotion-based approach is proposed to predict the priority of bug reports which help developers to focus on bugs resolution by avoiding their manual prioritization.
- Evaluation results of the proposed approach on the history data suggest that the proposed approach is accurate in priority prediction of bug reports.

The remaining sections of the paper are arranged as follows: The background study is explained in Section II. Section III presents the detailed description of the proposed approach. Section IV describes the evaluation process of the proposed approach and its results. Section V and Section VI presents the threats and discusses the related works respectively. Finally, Section VII concludes the paper and suggests future work.

II. BACKGROUND KNOWLEDGE

A. BUG REPORT

Users write bug reports on bug tracking systems if they find any problems in the software product. A sample bug report is shown in Fig. 1.

- This bug report was reported on 18 January 2010 (5) and was fixed on 11 August 2010.
- The title (1) defines the summary (3) of the bug report.
- This report also includes some data fields, e.g., status, severity, product, and cc. Such fields are common for all reports in a different bug.
- The priority (*severity* and *priority* are same on Android tracking systems) of the bug defines its importance.

The priority level of bug reports is different in various bug tracking systems. For the priority prediction of bug reports, we need to be careful about severity because, Eclipse defines priority as P1, P2, P3, P4 or P5. However, Android[1] and JBoss[2] define *severity* as *priority*.

[1]https://source.android.com/setup/report-bugs
[2]https://issues.jboss.org/

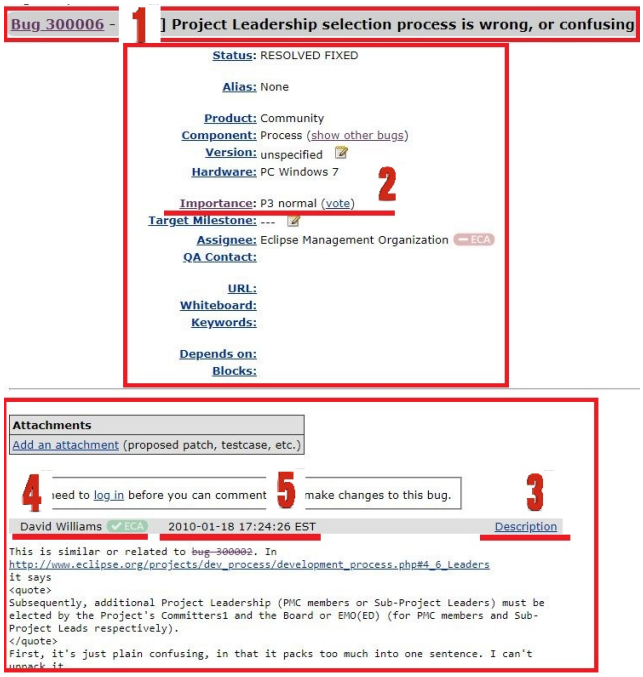


FIGURE 1. Summary of bug report (#300006).

B. BUG REPORTING AND TRIAGING

Each bug report contains information about how a bug could be resolved using related information. A bug report consists of bug-id, title (Text fields include of summary and long description), component, product, resolution, status, the estimated severity (how serious bug is) and the priority (how important bug is, represented normally as levels P1-P5). These fields are entered by the reporter but may be changed by the triager or developer if needed [3]. When a new bug report is placed in the repository its status is NEW. Then a triager takes a decision about bugs reported in bug repository based on standards procedures.

Based on the investigation, triager makes sure that a new reported bug is not duplicate [14] or not to check the validity of it. The purpose of such decisions is to filter out the valid bug reports. For the development-oriented decisions, the triager analyzes the priority of new bug reports and changes their priority if required. Assigning correct priority is a very important task to resolve more important bugs first. Triager also writes comments for bug reports and assigns them to the appropriate developer for their resolution. Developers fix them and change their status as RESOLVED which is further changed to CLOSED or CONFIRMED by the quality assurance personals.

III. APPROACH

A. OVERVIEW

The proposed approach categorizes the bug reports into five classes: P1, P2, P3, P4 or P5, where P1 has top priority and P5 has the least priority. The overview of the approach is shown in Fig. 2. The priority anticipation of a new bug report

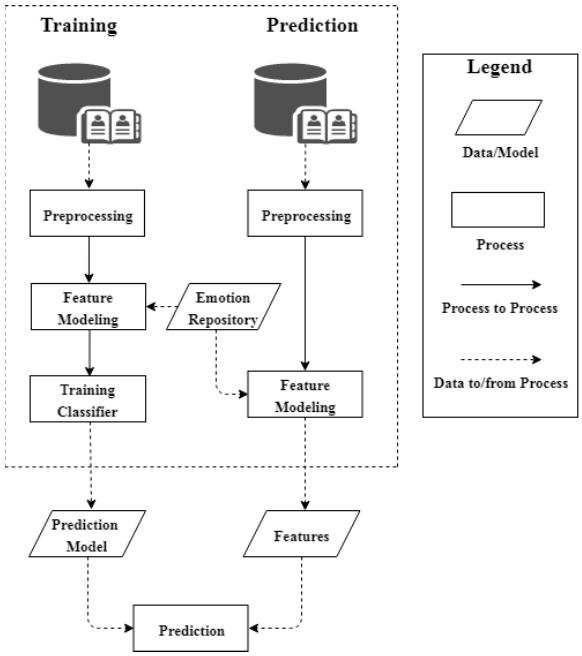


FIGURE 2. Overview of the proposed approach.

br can be categorized into predefined class *c* as a function *f*.

$$c = f(br) \quad c \in \{P_1, P_2, P_3, P_4, P_5\}, \; br \in BR \tag{1}$$

where c represents the classification result: P_1, P_2, P_3, P_4 or P_5, f represents the categorization function of priority anticipation, *br* represents a bug report, and *BR* represents a set of bug reports.

For the priority anticipation of a bug report, first we collect the bug reports' history data from Eclipse. Next, we perform the preprocessing using extracted bug reports. Then, from the preprocessed bug reports, we employ an open source emotion repository [10] to find the emotion words from the summary attribute of bug reports. Next, we calculate and assign an emotion-value to each bug report. Finally, we train and test a classifier to anticipate the priority of bug reports. Each of the key steps is introduced in the following sections.

B. DATA ACQUISITION

Software bugs are generally reported into issue-tracking systems which help in continuous monitoring of reported bugs. We extract and store the bug reports of Eclipse from Bugzilla issue-tracking system.

A bug report *br* is from a set of bug reports (BR) which can be formalized as,

$$br = < d, p > \tag{2}$$

where d represents the textual information and p represented the associated priority of each bug report.

C. PREPROCESSING

Standard preprocessing techniques are applied using Python Natural Language Processing Toolkit (NLTK) [15] to convert a textual information into a set of features. These preprocessing techniques include tokenization, Parts of Speech (POS) tagging, stop-word removal, and lemmatization. Fig. 3 illustrates the following steps involved in preprocessing.

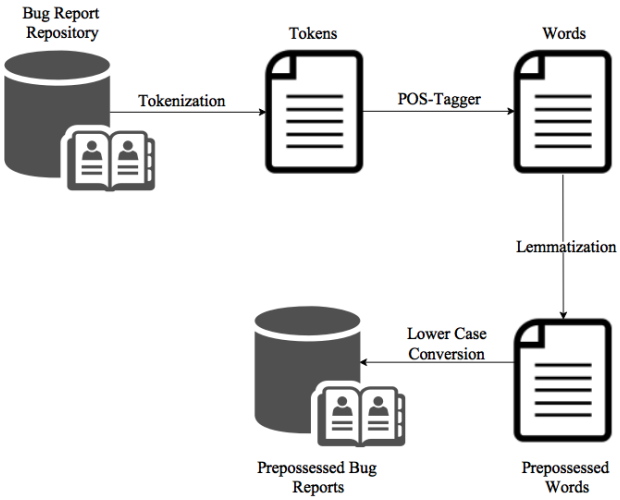


FIGURE 3. Overview of preprocessing.

1) TOKENIZATION

Breaking up a sequence of textual document into words is referred as tokenization. In this process, each word is called a *token* which is tagged with a POS tag after misspelling correction. Note that some natural language processing tools perform POS tagging in a separate step.

2) STOP-WORD REMOVAL

The frequently used words like "the," "in," "am," "are," "is," "I," "he" and "that" which has no meaning in actual are known as stop-words. Such words do not carry much information in the context of a bug report. We remove these words from the set of tokens extracted in the previous step.

3) LEMMATIZATION

Each term appearing in the description of bug report can appear in various forms. We perform stemming of all filtered words to convert into their ground word. For example, there is no difference in between "write" and "writes." In the text mining and information retrieval community, stemming plays a vital role. For example, "working", "worked" and "work" would all be reduced to "work". There are various algorithms available to perform stemming, however we use Porter's stemming algorithm [20] for lemmatization as it is a commonly used stemming algorithm by many researchers. The preprocessing of a bug report can be formalized as,

$$er = < d', p > \tag{3}$$

$$d' = < t_1, t_2, \ldots, t_n > \tag{4}$$

where d' is preprocessed textual description of bug report *br*, p represents its priority and $t_1, t_2, \ldots, t_n$ represents the n terms or tokens involved in d'.

D. EMOTION VALUE CALCULATION

In order to calculate the emotion-value of each bug report, the words from each preprocessed bug report are taken into consideration and compared with the list of words from emotion-word corpus [10]. SentiWordNet is a commonly used corpus [16]–[19] which includes the emotion values of each emotion word. The total number of positive and negative emotion words are *23,147* and *26,440* respectively, which are extracted from the bug reports. In Table 1, we provide top ten words having highest emotion values (positive and negative) from the emotion-word base corpus. The emotion corpus contains positive and negative scores for each word. We use the sum of the positive score and negative score of the emotion words of each bug report to calculate its emotion score.

TABLE 1. Top 10 highest emotion values.

Iteration	Positive Term & Score	Negative Term & Score
TOP 1	Good (**15.375** / 0.125)	Bad (0.875 / **10.625**)
TOP 2	Clear (**9.625** / 1.5)	Wrong (0.75 / **8.125**)
TOP 3	Well (**7.792** / 0.708)	Suffer (0 / **7.75**)
TOP 4	Clean (**7.625** / 3.875)	Dead (2.125 / **7.375**)
TOP 5	Clean (**7.625** / 3.875)	Rough (1.125 / **7.375**)
TOP 6	Better (**6.75** / 0.25)	Hurt (0 / **7.25**)
TOP 7	Light (**6.25** / 5.25)	Lose (0.25 / **7**)
TOP 8	Respect (**4.5** / 0)	Sting (0 / **7**)
TOP 9	Love (**4.375** / 0.125)	Black (0.625 / **6.875**)
TOP 10	Sound (**4.375** / 0.375)	Trouble (0.625 / **6.5**)

To assign an emotion-value to each bug report, we filter out the words of each bug report as emotion words if they are in the corpus, otherwise ignored.

E. FEATURE EXTRACTION

In a classification problem, we extract features from bug reports that can be effective in the characterization of bug reports. In order to build a model to predict the priority of a new bug report, we select the bug reports from Bugzilla. We analyze the bug reports and examine them to identify feature words from the summary of each bug report. Additionally, we calculate the emotion-value of each bug report using emotion words that are involved in the summary of the bug report. Finally, a high dimension matrix is created where each bug report is a row of the matrix and its emotion value and feature words are the columns of the matrix. A feature vector can be defined as,

$$br = < emo, f_1, f_2, \ldots . f_n > \tag{5}$$

where br represents the bug report, emo is the calculated emotion-value of each bug report and f_1, f_2,.. f_n represents the features words.

For each bug report, we use term frequency (TF) to represent a feature in the feature vector. We define a rule to find features from each bug report, if feature not found in bug reports, marked as *0* otherwise marked by its frequency count (N). Following conditions are used to mark the value of the feature to fill the matrix.

$$f_i(br) = \begin{cases} 0, & if\ t_i \notin d' \\ 1, & if\ t_i \in d' \end{cases}$$

where f_i is featureset of preprocessed textual description d' of each bug report br.

F. TRAINING AND PREDICTION

The training and prediction are two main steps of the classification module. The feature vectors produced by feature extraction module are taken into consideration for training and testing.

The proposed approach utilizes the Support Vector Machine (SVM) to captures the relationship between extracted features and priority levels of bug reports. We use support vector machine for classification for the following reasons. First, it scales relatively well to high dimensional data set using kernel trick. Second, the trade-off between classifier complexity and error can be controlled explicitly. Third, the flexible threshold can be applied during the selection of priority levels for the imbalanced dataset. Therefore, we train the support vector classifier using feature sets of each bug reports which are tagged while feature modeling using Equation (5). Then the trained model is used to predict the priority level of bug reports.

1) TRAINING

Given a training data, the proposed approach build a model capturing the relationship between explanatory variable with a dependent value. In our problem setting, a given set of bug reports $BR = <br_1, br_2, br_3, \ldots . br_n>$ form explanatory variable, while the priority level *P1 - P5* are the dependent variables. Each bug report br from BR contains the classification category c as mentioned in Equation (1), emotion-value *emo*, and set of features f as mentioned in Equation (5). Next, we train the support vector classifier for the proposed approach. To find the decision surface for the priority prediction of each bug report, we divide our reports into five categories P1, P2, P3, P4, and P5. The decision plane is called hyper-plane which is used for the training of our approach. The hyper-plane can be formalized as,

$$f(br) = w^\top br + b$$

where function f calculates the hyperplane for the training bug reports br, w is a weight vector which is normal to decision surface and b is bias.

Given the separable data br_i labeled into specified five categories y_i to find a weight vector w, function f can be specified as,

$$f(br_i) = w^\top br_i + b$$

separates the categories for *i = 1, 2, ….. n*. Hence, the separable data is formalized as,

$$f(br) = \sum_i^n y_i \alpha_i \left({br_i}^\top br \right) + b$$

where α_i represents the Lagrangian multiplier which is used to find the possible extremes to the respective classification category. The optimized support vector classifier can be formalized as,

$$max(w)\frac{2}{\|w\|}$$
$$subject\ to\ w^\top br_i + b \underset{\leq -1\ if y_i=-1}{\overset{\geq +1\ if y_i=+1}{}} \quad for\ i = 1 \ldots..n$$

2) PREDICTION

Once the support vectors are defined with the training set, each bug report from the testing dataset may be prioritized by comparing the results of the following equation for each priority class defined in training.

$$w^\top br + b > 1$$

Finally, compare the results of all priority classes to pick the best one.

IV. EVALUATION

In this section, we evaluate the performance of the proposed approach on the bug reports of four main Eclipse projects.

A. RESEARCH QUESTIONS

The evaluation investigates the following research questions:

- **RQ1:** Does the proposed approach outperform the state-of-the-art approach? If yes, to what extent?
- **RQ2:** Is there any strong correlation between the emotion value of bug reports and their priority? If yes, how strong is the correlation.?
- **RQ3:** Does support vector outperform other classification algorithms in predicting priority for bug reports?

To answer the research question (RQ1), we compare the performance of proposed approach with the state-of-the-art approach [1] in order to find out the performance improvement of the proposed approach. DRONE [1] is the latest approach that could help to predict the priority of bug reports and has significant results. Therefore, we select it for the comparison with the proposed approach.

To answer the research question (RQ2), we figure out the relationship between the emotion value and priority of bug reports.

To answer the research question (RQ3), different classification algorithms are taken into consideration and compared with the proposed approach to evaluate the performance (precision, recall, and f1-score) of the proposed approach.

B. DATASET

We use the same dataset as DRONE [1]. We extract the bug reports of four main projects: Java development tools (JDT), Eclipse's C/C++ Development Tooling (CDT), Plug-in Development Environment (PDE) and Platform from Bugzilla.

The history data is extracted from Bugzilla[3] using its Native REST API.[4] We collect the summary, resolution and priority attributes of each bug report by filtering the Bugzilla API's JSON response, submitted from October 2001 to December 2007. Summary briefly describes the bug reported by the users, resolution keeps the status of the bug and priority indicates the importance of the bug that is marked by the developers. The total number of bug reports are 80,000 which includes *25%*, *28%*, *16%*, *31%* bug reports of CDT, JDT, PDE and Platform respectively.

C. PROCESS AND METRICS

1) PROCESS

The evaluation of the proposed approach is performed as follows, we first extract the bug reports (*BR*) of four main Eclipse projects from Bugzilla and preprocessed them as mentioned in Section III. Second, we perform the cross-project technique on the given dataset.

For the i^{th} cross-validation, we select the bug reports *BR* that are not from the project P_j as training dataset *(TR)* and bug reports from project P_j as a testing dataset *(TE)*.

For the i^{th} cross-validation, a step by step process is as follows:

- First, we extract and combine all the bug reports *TR* from *BR* but project P_j.

$$TR_j = \bigcup_{k\in[1,4]\ \wedge\ k\neq j} \mathbf{P_k}$$

- Second, we train a Naïve Bayes classifier (*NB*) on *TR*.
- Third, we train a Multinomial Naïve Bayes classifier (*MNB*) on *TR*.
- Fourth, we train a Linear Regression classifier (*LR*) on *TR*.
- Fifth, we train a proposed classifier (*SV*) on *TR*.
- Sixth, for each bug report from *TE*, we predict its priority using trained *NB*, *MNB*, *LR* and *proposed approach* respectively.
- Finally, we calculate the precision, recall, and F1-score for each classifier for their comparison.

2) METRICS

To evaluate the performance of the proposed approach of the given bug reports, we calculate the priority specific *Precision*, *Recall* and *F1-score* of the approaches:

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$
$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$
$$F1 - score_i = \frac{2 * Precision_i * Recall_i}{Pecision_i + Recall_i}$$

where, $Precision_i$ ($Recall_i$ / $F1 - score_i$) is the precision (recall / F1-score) of the approaches in predicting priority of

[3]https://https://www.bugzilla.org
[4]https://bugzilla.readthedocs.io/en/latest/api/

TABLE 2. Evaluation results of macro and micro analysis.

Approach	Macro			Micro			Error
	Precision	Recall	F1-score	Precision	Recall	F1-score	
Proposed	57.62%	38.78%	46.36%	58.94%	38.14%	46.31%	0.4397
DRONE	53.63%	32.23%	40.26%	54.45%	31.77%	40.12%	0.4790
Improvement	**3.99%**	**6.55%**	**6.10%**	**4.49%**	**6.37%**	**6.19%**	**0.0820**

TABLE 3. Priority-level comparison against drone.

Approach	P1			P2			P3			P4			P5		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Proposed	59.42%	26.49%	36.64%	58.19%	22.75%	32.71%	57.75%	91.38%	70.77%	60.89%	22.86%	33.24%	58.46%	27.21%	37.14%
DRONE	56.35%	18.39%	27.24%	53.83%	14.32%	22.54%	54.54%	93.23%	68.74%	54.94%	13.67%	21.65%	52.59%	19.22%	28.00%
Improvement	**3.08%**	**8.10%**	**9.40%**	**4.36%**	**8.43%**	**10.17%**	**3.21%**	**-1.85%**	**2.04%**	**5.94%**	**9.18%**	**11.59%**	**5.87%**	**7.99%**	**9.14%**

TABLE 4. Project-level comparison against drone.

Approach	CDT			JDT			PDE			Platform		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Proposed	60.99%	40.69%	48.81%	59.86%	41.24%	48.84%	64.22%	34.79%	45.13%	50.70%	35.83%	41.98%
DRONE	56.66%	32.74%	41.50%	54.79%	35.21%	42.87%	58.26%	29.22%	38.92%	48.09%	29.89%	36.86%
Improvement	**4.32%**	**7.94%**	**7.31%**	**5.07%**	**6.03%**	**5.97%**	**5.96%**	**5.57%**	**6.21%**	**2.61%**	**5.94%**	**5.12%**

bug reports whose actual priority is P_i. TP_i is the number of bug reports that are predicted as P_i and they are actually P_i, FP_i is the number of bug reports that are predicted as P_i whereas they are actually not P_i, and FN_i is the number of bug reports that are not predicted as P_i whereas they are actually P_i.

Besides the priority specific precision, recall, and F1-score, we also employ the macro-analysis and micro-analysis for all classes C that are widely employed to evaluate the performance of multi-class classifier. In this perspective, we compute micro-precision P_{micro}, micro-recall R_{micro} and micro-F1 $F1_{micro}$ that can be formalized as:

$$P_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FP_i}$$

$$R_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FN_i}$$

$$F1_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{2 * P_{macro} * R_{macro}}{P_{macro} + R_{macro}}$$

$$P_{micro} = \sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FP_i}$$

$$R_{micro} = \sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FN_i}$$

$$F1_{micro} = \sum_{i=1}^{|C|}\frac{2 * P_{micro} * R_{micro}}{P_{micro} + R_{micro}}$$

We also calculate the error using Hamming-loss which calculates the average number of the relevance of an example to a class which is incorrectly predicted [20]. It normalizes the loss over a total number of classes and the total number of examples using prediction error (an incorrect label is predicted) and missing error (a relevant label is not predicted). The error can be formalized as:

$$Error = \frac{1}{|N| \, . \, |L|}\sum_{i=1}^{|N|}\sum_{j=1}^{|L|}\left(y_{i,j}, z_{i,j}\right)$$

where, N represents the number of bug reports, L represents the number of labels, where $y_{i,j}$ is the target labels and $z_{i,j}$ is the prediction labels.

D. RESULTS

1) RQ1: COMPARISON AGAINST DRONE

The average precision, recall, and F1-score of the proposed approach and DRONE for macro-analysis and micro-analysis are presented in Table 2. The columns of the table represent the average precision, recall, F1-score, and error-rate. Whereas, the rows of the table represent the performance of both approaches. However, the last row represents the improvement achieved by the proposed approach.

From the Table 2, we observe that the proposed approach outperforms DRONE. It improves the F1-score for both macro-analysis and micro-analysis by *6.10%* and *6.19%* respectively and decreases the error by 0.0820.

Moreover, the performance of both approaches for each priority and each project is presented in the Tables 3 and 4 respectively. The columns of Table 3 represent the precision, recall, and F1-score for each priority. However, the columns of Table 4 represent the precision, recall and F1-score for each project. Whereas, the rows of both tables represent the results of precision, recall and F1-score of both approaches.

TABLE 5. Performance comparison of the proposed approach.

Approach	CDT			JDT			PDE			Platform		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Proposed	**60.99%**	**40.69%**	**48.81%**	**59.86%**	**41.24%**	**48.84%**	**64.22%**	**34.79%**	**45.13%**	**50.70%**	**35.83%**	**41.98%**
LR	58.96%	35.09%	43.99%	57.33%	36.86%	44.87%	63.35%	30.49%	41.17%	49.75%	31.69%	38.72%
MNB	54.43%	34.48%	42.22%	52.57%	36.11%	42.81%	58.18%	31.06%	40.50%	48.20%	31.15%	37.84%
NB	53.09%	36.75%	43.43%	50.89%	37.92%	43.46%	55.22%	33.07%	41.37%	45.33%	32.72%	38.01%

However, the last row from both tables represents the average improvement achieved by the proposed approach.

From the Table 3 and 4, we make the following observations:

- The proposed approach achieves significant improvement in performance on each priority level. The improvement in *F1-score* varies from *2.04%* to *11.59%*.
- The proposed approach achieves significant improvement in performance on each project. The improvement in *F1-score* varies from *5.12%* to *7.31%*.
- The proposed approach fails to achieve significant improvement in recall on priority level *P3*. One possible reason is that the bug reports with priority *P3* contain fewer positive or negative words whereas the proposed approach improves the state-of-the-art by exploiting such strong positive/negative words.

We also apply ANOVA analysis on the *F1-score* (given in the columns of Table 4 labeled as *F1-score*) of cross-project evaluation. The results are presented in Table 6 where $F > F_{cric}$ and $P_{valus} < (\alpha = 0.05)$. Consequently, the ANOVA analysis confirms that the single factor (different prediction approaches) do lead to a significant difference in performance.

TABLE 6. ANOVA analysis of the proposed approach against drone.

Source of Variation	SS	df	MS	F	P-value	F-crit
F1-score						
Between Groups	0.007569	1	0.007569	8.383639	0.027497	5.987378
Within Groups	0.005417	6	0.000903			
Total	0.012986	7				

Based on the preceding analysis, we conclude that the proposed approach outperforms the state-of-the-art.

2) RQ2: INFLUENCE OF EMOTION ANALYSIS

The percentages of positive, negative and neutral bug reports against each priority-level are presented in Table 7. The rows and columns of the table represent the emotion's type and priority-level respectively.

From the Table 7, we observe that a bug report having negative emotion may get higher priority level. Whereas, bug reports having positive emotion may get lower priority level.

We also compute the Pearson correlation coefficient (r) to access the strength of the relationship between the two variables (emotion and priority). Results ($r = 0.405$) suggest that there is strong positive correlation between emotion and priority of bug reports.

TABLE 7. Priority-wise emotion analysis of dataset.

Bug Reports	P1	P2	P3	P4	P5	Total
Negative	61.71%	53.87%	55.71%	28.64%	29.51%	49.28%
Positive	24.23%	27.89%	18.26%	43.82%	56.67%	28.89%
Neutral	14.06%	18.24%	26.02%	27.54%	13.82%	21.83%

3) RQ3: PERFORMANCE COMPARISON OF CLASSIFICATION ALGORITHMS

We apply machine learning algorithms (naive bayes, multinomial naive bayes, linear regression and support vector machine) in order to evaluate the best classifier to predict the priority of bug reports. The reason to choose these classifiers is their importance in terms of usage and competitive performance [21].

The performance results of each classifier are shown in Table 5. The columns of the table represent the average precision, recall, and F1-score for each project. Whereas, the rows of the table represent the results of each classifier. We also compare their resulting performance in Fig. 4 that visualizes the difference.

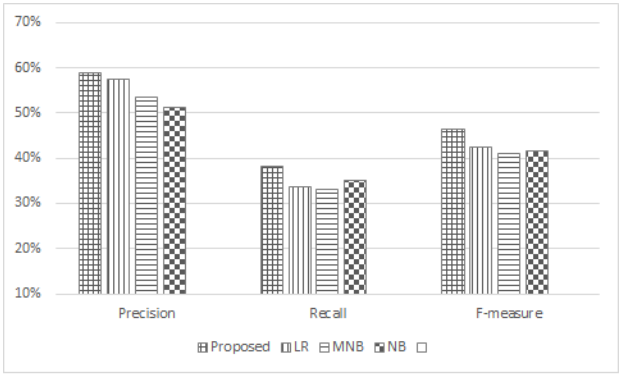


FIGURE 4. Performance comparison of algorithms.

From Table 5 and Fig. 4, we observe that the proposed approach (SVM) outperforms other machine learning algorithms. It achieves the best performance (precision, recall, and F-score) on each of the subject applications.

V. THREATS

THREATS TO VALIDITY

A threat to construct validity is the suitability of our evaluation metrics. We have used the precision, recall, and F1-score for the evaluation of classification algorithms. Because, these metrics are standard and also adopted by many researchers [1], [22].

A threat to construct validity is related to the usage of SentiWordNet corpus for emotion analysis. There is a number of other sentiment corpora, however, we select SentiWordNet due to its excessive use in research. Other sentiment corpora may decrease the performance of the proposed approach.

A threat to internal validity is related to the implementation of the approaches. To mitigate the threat, the implementation and results are checked. However, there could be some unnoticed errors.

A threat to external validity is related to the generalizability of our results. We have only considered and analyzed the bug reports that are related to the four main projects of Eclipse. Also, we exclude those bug reports that do not contain information about their priority field.

A threat to external validity is the proposed approach may not perform well or may not work for the bug reports written in other languages. The proposed approach is trained and evaluated on bug reports which are written in English.

A threat to external validity is a small number of bug reports. Therefore, we use traditional machine learning algorithms to evaluate the proposed approach. Deep learning algorithms have a number of parameters to be adjusted and usually require a large training data. Without those settings, the performance can be influenced.

VI. RELATED WORK

Although a number of approaches have been proposed in order to prioritize bug reports and provide good results. However, the existing studies are not considering the emotion involved in bug reports. To this end, in this paper we proposed an automated approach to prioritize the bug reports with emotion analysis which is different from state-of-the-art approaches.

Some state-of-the-art approaches on bug reports classification that define their importance and identify their priority are following:

SEVERIS (SEVERity ISsue assessment) is an automated algorithm proposed by Menzies and Marcus [22] to assign severity levels to bug reports. They are first to predict the severity of the various bugs reported in NASA. Their approach provides fine-grained severity level out of 5 severity levels used in NASA. They used information-gain to get feature-words from the reports. Top-k feature words are used for feature modeling of the bug reports. The feature vectors are used to train a machine learning classifier which can predict the severity of future bug report.

Menzies and Marcus [22] work is extended by Lamkanfi and *et al.* [23]. They adopted the approach and predict the severity of reports in open-source bug repositories. They analyzed the textual description of bug reports, that are from GNOME, Eclipse and Mozilla, to predict their severity. They used 5 severity labels out of 6 severity labels and grouped them into two categories: severe and non-severe for the severity prediction.

Chaturvedi and Singh [24] demonstrated the applicability of various machine learning algorithms to predict the severity levels of bug reports using textual summary.

Tian *et al.* [1] proposed an automated classification approach to predict the priority of bug reports. They utilized machine learning algorithm for priority classification and achieve the average F1-score up to *29%*.

Abdelmoez *et al.* [25] used Naive Bayes classifier to build a model that could predict the bug reports in order to prioritize and fix them according to their mean-time.

Dommati *et al.* [26] focused on feature extraction and noise reduction of bug reports using Naïve Bayes, Multinomial Naïve Bayes.

Supervised and unsupervised classification and multivariate visualization technique was proposed by Podgurski *et al.* [27] to prioritize the software failures and diagnosing their causes. The resulting classification is then used to assess the operational frequency and severity of failures caused to diagnose those defects.

An Artificial Neural Network (ANN) based framework was proposed by Yu *et al.* [14] to predict the priorities of five different product bugs reported by an international health-care company. Experiments on threefold cross-validation suggest that proposed approach is better in term of precision, recall and f1-score.

Kanwal and Maqbool [13] proposed a machine learning based recommender to assign automatic priorities of reported bugs. They utilized Support Vector Machine (SVM) to train their classification-based approach on Eclipse bug reports. Evaluation of proposed recommender use precision, recall, and f1-score for automatic bug priority assignment.

The studies related to software repositories [23], [28]–[32] such as source code repositories [33], email archives [34], duplicate detection [35] of bug reports, assigning an appropriate developer [36] and prediction of bug fixing-time [37] also classify the bug reports. However, these are different from the proposed approach.

VII. CONCLUSION

A large number of bug reports usually contain incorrect priority level which is assigned by reporters. Developers check and reassign priority to such bug reports which is a time-consuming task and requires a lot of manual efforts. To this end, in this paper, we proposed an emotion-words based automated approach to predict the priority levels of bug reports. The proposed approach combines the natural language processing techniques and machine learning algorithms to solve the said problem. The proposed approach helps users and developers by assigning an appropriate priority level to future bug reports in an automated way and saves

the valuable time of developers. The cross-project evaluation is performed on the history-data of the four main projects of Eclipse. The performance results suggest that the proposed approach outperforms the state-of-the-art.

ACKNOWLEDGMENTS

The authors would like to say thanks to the associate editor and the anonymous reviewers for their valuable suggestions.

REFERENCES

[1] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2013, pp. 200–209, doi: 10.1109/ICSM.2013.31.

[2] (Jan. 2018). (Bugzilla Issue Tracker). [Online]. Available: https://www.bugzilla.org/

[3] (Jan. 2018). *Mantis Bug Reporting System*. [Online]. Available: https://code.google.com/archive/p/support/wikis/IssueTracker.wiki

[4] (Jan. 2018). *Google Issue Tracker*. [Online]. Available: https://www.mantisbt.org/

[5] (Jan. 2018). *Github Issue Tracker*. [Online]. Available: https://github.com/features

[6] (Jan. 2018). *Jira Issue Tracker*. [Online]. Available: https://www.atlassian.com/software/jira

[7] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for Android applications," in *Proc. 10th Joint Meeting Found. Softw. Eng. (ESEC/FSE)*, 2015, pp. 673–686, doi: 10.1145/2786805.2786857.

[8] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "Dretom: Developer recommendation based on topic models for bug resolution," in *Proc. 8th Int. Conf. Predictive Models Softw. Eng. (PROMISE)*, 2012, pp. 19–28, doi: 10.1145/2365324.2365329.

[9] C. F. Kemerer, "Software complexity and software maintenance: A survey of empirical research," *Ann. Softw. Eng.*, vol. 1, no. 1, pp. 1–22, Dec. 1995, doi: 10.1007/BF02249043.

[10] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.* vol. 47, no. 2, pp. 145–180, 2016.

[11] J. K. Anvik, "Assisting bug report triage through recommendation," Ph.D. dissertation, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, 2007. [Online]. Available: https://open.library.ubc.ca/cIRcle/collections/24/items/1.0051337

[12] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?" in *Proc. 12th Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2013, pp. 112–116, doi: 10.1109/ICMLA.2013.114.

[13] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *J. Comput. Sci. Technol.*, vol. 27, no. 2, pp. 397–412, Mar. 2012, doi: 10.1007/s11390-012-1230-3.

[14] L. Yu, W.-T. Tsai, W. Zhao, and F. Wu, "Predicting defect priority based on neural networks," in *Advanced Data Mining and Applications*, L. Cao, J. Zhong, and Y. Feng, Eds. Berlin, Germany: Springer, 2010, pp. 356–367.

[15] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *Proc. ACL Workshop Effective Tools Methodol. Teach. Natural Lang. Process. Comput. Linguistics (ETMTNLP)*, 2002, pp. 63–70, doi: 10.3115/1118108.1118117.

[16] M. Z. Asghar, A. Khan, S. Ahmad, I. A. Khan, and F. M. Kundi, "A unified framework for creating domain dependent polarity lexicons from user generated reviews," *PLoS ONE*, vol. 10, no. 10, p. e0140204, 2015.

[17] J. Steinberger *et al.*, "azquez, "Creating sentiment dictionaries via triangulation," in *Proc. 2nd Workshop Comput. Approaches Subjectivity Sentiment Anal. (WASSA)*, 2011, pp. 28–36. [Online]. Available: http://dl.acm.org/citation.cfm?id=2107653.2107657

[18] M. Z. Asghar *et al.*, "Medical opinion lexicon: An incremental model for mining health reviews," *Int. J. Acad. Res.*, vol. 6, pp. 295–302, Jan. 2014.

[19] K. Denecke, "Using sentiwordnet for multilingual sentiment analysis," in *Proc. IEEE 24th Int. Conf. Data Eng. Workshop*, Apr. 2008, pp. 507–512.

[20] R. E. Schapire and Y. Singer, "BoosTexter: A boosting-based system for text categorization," *Mach. Learn.*, vol. 39, nos. 2–3, pp. 135–168, May 2000.

[21] A. Khan, B. Baharudin, L. H. Lee, K. Khan, and U. T. P. Tronoh, "A review of machine learning algorithms for text-documents classification," *J. Adv. Inf. Technol.*, vol. 1, no. 1, pp. 4–20, 2010.

[22] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2008, pp. 346–355.

[23] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proc. 15th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2011, pp. 249–258.

[24] K. K. Chaturvedi and V. B. Singh, "Determining bug severity using machine learning techniques," in *Proc. CSI 6th Int. Conf. Softw. Eng. (CONSEG)*, Sep. 2012, pp. 1–6.

[25] W. Abdelmoez, M. Kholief, and F. Elsalmy, "Bug fix-time prediction model using Naïve Bayes classifier," in *Proc. 22nd Int. Conf. Comput. Theory Appl. (ICCTA)*, Oct. 2012, pp. 167–172.

[26] S. J. Dommati, R. Agrawal, R. M. R. Guddeti, and S. S. Kamath, "Bug classification: Feature extraction and comparison of event model using Naïve Bayes approach," *CoRR*, vol. abs/1304.1677, Apr. 2012. [Online]. Available: https://arxiv.org/abs/1304.1677?context=cs.LG

[27] A. Podgurski *et al.*, "Automated support for classifying software failure reports," in *Proc. 25th Int. Conf. Softw. Eng. (ICSE)*, May 2003, pp. 465–475. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776872

[28] T. Kremenek and D. Engler, "Z-ranking: Using statistical analysis to counter the impact of static analysis approximations," in *Proc. 10th Annu. Int. Static Anal. Symp.*, 2003, pp. 295–315.

[29] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *J. Syst. Softw.*, vol. 117, pp. 166–184, Jul. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121216000765

[30] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Jul. 2014, pp. 97–106.

[31] M. Iliev, B. Karasneh, M. R. V. Chaudron, and E. Essenius, "Automated prediction of defect severity based on codifying design knowledge using ontologies," in *Proc. 1st Int. Workshop Realizing AI Synergies Softw. Eng. (RAISE)*, 2012, pp. 7–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2666527.2666529

[32] T. Zhang, G. Yang, B. Lee, and A. T. S. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proc. 30th Annu. ACM Symp. Appl. Comput. (SAC)*, 2015, pp. 1553–1558, doi: 10.1145/2695664.2695872.

[33] C. C. Williams and J. K. Hollingsworth, "Automatic mining of source code repositories to improve bug finding techniques," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 466–480, Jun. 2005.

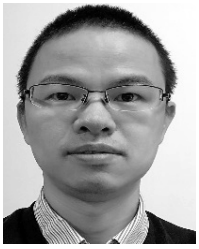
[34] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 25–35.

[35] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proc. 30th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 461–470, doi: 10.1145/1368088.1368151.

[36] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2006, pp. 1767–1772, doi: 10.1145/1141277.1141693.

[37] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. 4th Int. Workshop Mining Softw. Repositories (MSR)*, May 2007, p. 1, doi: 10.1109/MSR.2007.13.

QASIM UMER received the B.S. degree in computer science from Punjab University, Pakistan, in 2006, the M.S. degree in .net distributed system development and the M.S. degree in computer science from the University of Hull, U.K., in 2009 and 2012, respectively. He is currently pursuing the Ph.D. degree in computer science with the Beijing Institute of Technology, China. He is particularly interested in machine learning, data mining, and software maintenance.



HUI LIU received the B.S. degree in control science from Shandong University in 2001, the M.S. degree in computer science from the Shanghai University in 2004, and the Ph.D. degree in computer science from Peking University in 2008. He is currently a Professor with the School of Computer Science and Technology, Beijing Institute of Technology. He is particularly interested in intelligent software engineering, software refactoring, software evolution, and software quality. He is also interested in developing practical tools to assist software engineers.

YASIR SULTAN received the B.S. degree in information technology from Government College University in 2016. He is currently pursuing the M.S. degree in computer science with the Beijing Institute of Technology, China. He is particularly interested in software engineering, data mining, and machine learning.