IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 28, 2019, accepted March 31, 2019, date of current version April 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2909746

Deep Neural Network-Based Severity Prediction of Bug Reports

WAHEED YOUSUF RAMAY[1], QASIM UMER[2], XU CHENG YIN[1], CHAO ZHU[1], AND INAM ILLAHI[2]

[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
[2]School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

Corresponding authors: Xu Cheng Yin (xuchengyin@ustb.edu.cn) and Chao Zhu (chaozhu@ustb.edu.cn)

ABSTRACT Software maintenance is an essential phase of software development. Developers employ issue tracking systems to collect bugs for software improvement. Users submit bugs through such issue tracking systems and decide the severity of reported bugs. The severity is an important attribute of a bug that decides how quickly it should be solved. It helps developers to solve important bugs on time. However, manual severity assessment is a tedious job and could be incorrect. To this end, in this paper, we propose a deep neural network-based automatic approach for the severity prediction of bug reports. First, we apply natural language processing techniques for text preprocessing of bug reports. Second, we compute and assign an emotion score for each bug report. Third, we create a vector for each preprocessed bug report. Forth, we pass the constructed vector and the emotion score of each bug report to a deep neural network based classifier for severity prediction. We also evaluate the proposed approach on the history-data of bug reports. The results of cross-product suggest that the proposed approach outperforms the state-of-the-art approaches. On average, it improves the *f-measure* by *7.90%*.

INDEX TERMS Bug reports, deep learning, severity prediction, software maintenance.

I. INTRODUCTION

Software maintenance is an essential phase of software development. One of the most important tasks of the maintenance process is bug resolution [1], where bug reports represent the bugs that users encountered while using released software systems. Developers are interested to collect such bugs for the improvement of software systems. They utilize bug tracking systems (e.g., *Bugzilla* [2] and *JIRA* [3]) that help developers to manage bug reporting and bug triaging [4].

Users often submit bug reports through issue tracking systems. A bug report describes the particular situation under which a software bug occurred and contains information of bug regeneration. A typical bug report contains multiple attributes: bug-id, submission date, status, priority, severity, summary, and description. However, the severity is an important attribute of a bug report that decides how quickly it should be resolved. In Bugzilla, severity of a bug report may varies from *trivial*, *minor*, *normal*, *major*, *critical* to *blocker*. Users decide and manually assign the severity to bugs at reporting time which is a tedious task that requires domain knowledge. It may lead to an incorrect assessment of severity due to different reasons e.g., inexperienced users.

Researchers have proposed different approaches [5]–[10] to automate the severity prediction of bug reports. Most of the approaches focus on traditional classification algorithms, i.e., j48, decision trees, naive Bayes and support vector machine. However, the performance of such approaches require significant improvement, and none of them incorporate the reporters' emotion in severity prediction [11]. Umer *et al.* [11] reported that the count of negative emotions of reporters is higher in severe bugs as compared to non-severe bugs. Hence, reporters while writing bugs are expressive. For example; the words/phrases (e.g., *pathetic interface*, *worst*, *incorrect*) used by reporters may indicate the urgency of a bug report. Such expressions (emotions) could help in severity prediction of bug reports.

To this end, in this paper, we propose an automatic approach for the severity prediction of bug reports using a deep learning classification algorithm. We apply natural language processing techniques for text preprocessing of bug reports. Given the preprocessed bug reports, we perform an emotion analysis and assign an emotion score to each bug report as users are often expressive while experiencing bugs

The associate editor coordinating the review of this manuscript and approving it for publication was Santhosh Kumar Gopalan.

and writing bug reports that may influence the selection of severity level. Furthermore, we create a vector for each bug report based on the preprocessed text. Finally, we pass the constructed vector and the emotion score of each bug report to a deep neural network based classifier that predicts the severity of bug reports. We also evaluate the proposed approach on the history-data of bug reports. Results of cross-product suggest that the proposed approach outperforms the state-of-the-art approaches. On average, it improves the *f-measure* by *7.90%*.

This study makes the following contributions:

- An automated deep neural network based approach for severity prediction of bug reports. To the best of our knowledge, it is the first deep learning based approach to predict the severity of bug reports.
- Evaluation results of the proposed approach on the history-data suggest that the proposed deep learning approach is accurate in severity prediction of bug reports and outperforms the state-of-the-art approaches.

The remaining sections of the study are organized as follows: Section II defines the proposed approach in detail. Section III describes the evaluation process of the proposed approach and its results. Section IV explains the threats. Section V discusses the related work respectively. Section VI concludes the paper and suggests future work.

II. PRELIMINARY STUDY

A. EMOTION ANALYSIS

Recently, researchers are facing a challenge to make the machines emotionally intelligent in software engineering [12]. The software engineering related text (bug reports in our case) may contain emotion words. For example, a bug report can be declared either *emotionally positive* if it contains positive words (e.g., *good*, *well*, *right*) or *emotionally negative* if it contains words (e.g., *bad*, *wrong*, *suffer*).

To this end, we analyze the emotional words of bug reports based on bug severity levels (*trivial*, *minor*, *normal*, *major*, *critical*, and *blocker*). We observed that the words *break*, *wrong*, and *incorrect* having highest negative scores. In our case (binary classification), the words (e.g., *break*, *crash*, and *error*) indicate the *severe* bugs reports. Whereas, the words (e.g., *warn*, *minor*, and *incorrect*) indicate the *non-severe* bug reports. Note that we utilize *Senti4SD* repository for emotion analysis as explained in Section III-E.

III. APPROACH

A. OVERVIEW

An overview of the deep neural network based severity prediction of bug reports is presented in Fig. 1. The proposed approach predicts the severity of bug reports as follows:

- First, we extract the history-data of bug reports from open-source projects.
- Second, we preprocess the bug reports using natural language processing techniques.
- Third, we compute and assign an emotion score to each bug report.

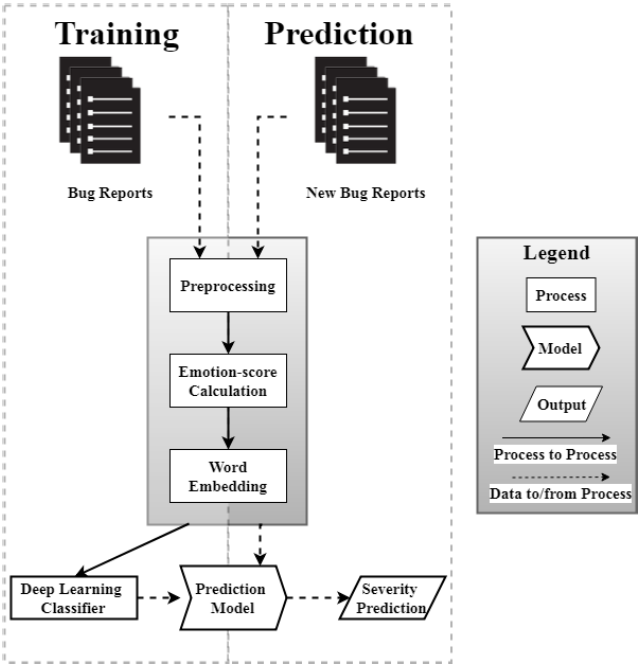

FIGURE 1. An overview of the proposed approach.

- Fourth, we create a vector (word-embeddings) for each preprocessed bug report.
- Finally, we train a deep learning based classifier for severity prediction. We input the emotion score and the vector of each bug report to the classifier for its severity prediction.

The following sections introduce each of the key steps of the proposed approach.

B. ILLUSTRATING EXAMPLE

We use the given example to illustrate how the proposed approach predicts the severity of bug reports. It is an *Eclipse* bug report (*#437094*) collected from *Bugzilla* [2].

- **Product** = "*ECP*" is the name of the affected product.
- **Textual Information** = "*The EMFFilter must be updated to filter out new models added to luna*" is the description of the bug report. It may contain the details on how the bug can be regenerated.
- **Severity** = "*critical*" is the severity of the bug report that indicates how quickly a bug report should be resolved.

The following sections present the details on how the proposed approach performs for the illustrating example.

C. PROBLEM DEFINITION

Software developers often employ bug tracking system for software maintenance. We collect and save the bug reports of *Mozilla* and *Eclispe* from Bugzilla where severity of bug reports varies from *trivial*, *minor*, *normal*, *major*, *critical* to *blocker*. In the proposed approach, we specify the severities

trivial, *minor*, and *normal* as *non-severe*. Whereas, *major*, *critical*, and *blocker* are specified as *severe*.

Given a set of bug reports $\mathbb{R}$, a bug report r can be formalized as

$$r = < t, s > \tag{1}$$

where t and s are the textual description and severity of r, repectively.

For the illustrating example presented in this section, we have

$$r_e = < t_e, s_e > \tag{2}$$

where,

t_e = *"The EMFFilter must be updated to filter out new models added to luna"*, and

s_e = *severe*

The proposed approach predicts the severity of the new bug report either *severe* or *non-severe*. *severe* indicates that the new bug is more important and urgent than *non-severe* existing bugs. Whereas, *non-severe* indicates that the new bug report may be postponed to first solve the existing *severe* bugs. Consequently, the severity prediction of a new bug report r can be defined a mapping f as

$$f : r \rightarrow c$$
$$c \in \{severe,\ non{-}severe\}, \quad r \in \mathbb{R} \tag{3}$$

where c is a predicted severity either *severe* or *non-severe*, r is a bug report, and $\mathbb{R}$ is a set of bug reports.

D. PREPROCESSING

In this section, we briefly introduce each of the key steps of natural language processing that we apply to preprocess the text of bug reports. We perform tokenization, spell correction, stop-word removal, word inflection, and lemmatization to clean the textual description of bug reports. To this end, we employ *Natural Language Toolkit (NLTK)* [13] and *TextBlob* [14] to apply natural language processing techniques.

1) TOKENIZATION

The textual description of bug reports usually combines word and meaningless symbols e.g., punctuation. Such symbols do not contribute to the severity classification of bug reports. Tokenization filters out the meaningless symbols and divides the remaining text into tokens.

2) SPELL CORRECTION

The unstructured attributes (e.g., summary and description) of bug reports contain text written by users that may have spelling mistakes or typo-errors. Therefore, we correct such mistakes in this step of preprocessing.

3) STOP-WORD REMOVAL

The text of a document often contains constructive terms (e.g., prepositions) and other language structures to make sentences. Such terms are known as stop-words. The occurrence of stop-words in bug reports may increase the dimensionality of data that may decrease the efficiency of classification algorithms. In this perspective, we subtract stop-words from the preprocessed bug reports.

4) WORD INFLECTION AND LEMMATIZATION

Word inflection transforms words into their singular form and lemmatization shifts the comparative and superlative terms into their basic term. For example; inflection transforms the word *bugs* into *bug* and lemmatization shifts the word *computation* into *compute*. We perform both word inflection and lemmatization to avoid the repetition of words that share the same basic term. Finally, we convert all the preprocessed words into lowercase.

A preprocessed bug report r can be represented as

$$r' = < w_n, s > \tag{4}$$
$$w_n = < w_1, w_2, \ldots, w_n > \tag{5}$$

where $w_1, w_2, \ldots, w_n$ represent the tokens from preprocessed bug report r.

For the illustrating example presented in Section III-B, Table 1 represents the effect of each preprocessing step on r_e. A preprocessed r_e can be represented as

$$r'_e = < emffilter, update, filter, \ldots, luna, \textbf{severe} > \tag{6}$$

where *emffilter*, *update*, *filter*, ..., *luna* are the preprocessed words from r_e and *severe* is the severity of r_e.

E. EMOTION SCORE CALCULATION

We compute the emotion score for each bug report as users are often expressive while experiencing bugs and writing bug reports. There are a number of tools available to compute the emotion score of the written text e.g., *SentiWordNet* [15]. However, to the best of our knowledge, *SentiStrengthSE* [16], *SentiCR* [17], *Senti4SD* [18] and *EmoTxt* [19] are widely used classification tool for software engineering text. We utilize a repository *Senti4SD* for two reasons: it is most recent and commonly used repository for emotion analysis in software engineering domain [20], [21] and according to Calefato *et al.*, it outperforms the *SentiStrength*, *SentiStrengthSE*, and *SentiCR* in classifying software engineering documents [18].

We pass the each bug report to the *Senti4SD* to compute its emotion score. Note that we pass the textual description of each bug report regardless of its severity level. *Senti4SD* returns the emotion score of the given bug report. A bug report r' with its emotion score *es* can be represented as

$$r'' = < es, w_1, w_2, \ldots..w_n, s > \tag{7}$$

To compute the emotion score of the given bug report, Senti4SD calculates the semantic features of the report as the distance between its vector representation and four prototype distributed semantic vectors. The vectors represent the three polarities and subjectivity. Senti4SD computes a

TABLE 1. An example of preprocessing.

Original Bug Report	The EMFFilter must be updated to filter out new models added to luna.
After Tokenization	'The', 'EMFFilter', 'must', 'be', 'updated', 'to', 'filter', 'out', 'new', 'models', 'added', 'to', 'luna'
After Stop-words Removal	'EMFFilter', 'updated', 'filter', 'new', 'models', 'added', 'luna'
After Word Inflection	'EMFFilter', 'updated', 'filter', 'new', 'model', 'added', 'luna'
After Lemmatization	'emffilter', 'update', 'filter', 'new', 'model', 'add', 'luna'

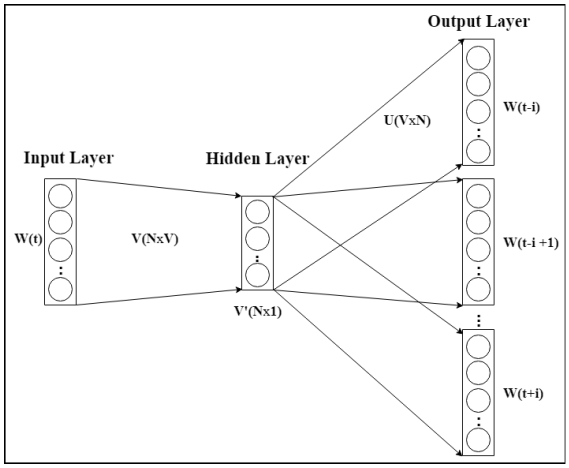

FIGURE 2. Word2Vector model.

vector sum of the words with their polarities according to the SentiStrength lexicon [22].

For the illustrating example presented in Section III-B, we compute its emotion score. A bug report r'_e with its emotion score can be represented as

$$r''_e = < positive, emffilter, update, filter, \ldots, luna, \textbf{severe} > \quad (8)$$

where *positive* represents the emotion score of r'_e.

F. WORD EMBEDDINGS

To input the textual description of bug reports to a deep learning based classifier, we transform words into fixed-length numerical vectors using word2vec (a skip-gram model) proposed by Mikolov *et al.* [23]. It is an efficient method for learning high quality distributed vector representations. It captures a large number of precise syntactic and semantic word relationships and returns k-dimensional vector. It is essentially a neural network that predicts the surrounding context words given the central target word (as shown in Fig. 2). Given a trained network, we can exploit the hidden layer to transform words into numerical vectors.

For each bug report r'', we pass the preprocessed words $w_1, w_2, \ldots, w_n$ from Eq. 7 and transform each word into a fixed-length numerical vector.

$$w_i = < w_1, w_2, \ldots..w_n > \quad (9)$$

$$= < v_1, v_2, \ldots..v_n > \quad (10)$$

where $w_1, w_2, \ldots, w_n$ is a sequence of words from each preprocessed report and v_i transforms word w_i into a fixed-length numerical vector.

For the illustrating example presented in Section III-B, the preprocessed words *emffilter*, *update*, *filter*, ..., *luna* from r''_e are passed to skip-gram model to convert them into a fixed-length numerical vector.

G. DEEP NEURAL NETWORK BASED CLASSIFIER

The composition of the deep neural network based classifier is presented in Fig. 3. We exploit Convolutional Neural Network (CNN) for severity prediction of bug reports because of the following reasons. First, CNN layers may learn the deep semantical relationships between input words for severity prediction of bug reports. Second, CNN significantly reduces training time due to its capability for parallel computation on modern powerful GPU [24]. Third, CNN may use different filter size filters that avoid the exploding gradient problem of recurrent neural network [25].

We pass preprocessed words $w_1, w_2, \ldots, w_n$ and emotion score *es* of each bug report r''(Eq. 7) to the deep neural network based classifier into two parts. We feed the words w_i into an embedding layer that transforms them into numerical vectors as discussed in Section III-F. Consequently, we feed the converted numerical vectors into a CNN. We use three layers of CNN with settings: *filter* = *128*, *kernel size* = *1* and *activation* = *tanh*. Where, *filter* represents the number of neurons, since each neuron performs a different convolution on the input to the layer (more precisely, the neurons' input weights form convolution kernels), *kernel size* represents the size of the *filter*, and *activation* function represents the final value of a neuron. We forward the output of CNN to a flatten layer [26] that turns the given converted numerical vectors into a one-dimensional vector.

We feed the emotion score *es* directly into another CNN (with the same setting as previous CNN) whose output is forwarded to a flatten layer. Both inputs (preprocessed words and emotion score) are finally merged by the merge layer [27] that combines a list of inputs. The following layers (dense layer fully connects the *128 neurons* to those in the next layer and output layer *2 neurons*) map both inputs into a single output (prediction) that predicts the severity s of r. We use *binary_crossentropy* as the loss function for the proposed model. Where *binary_crossentropy*is a cross-entropy loss that measures the performance of a classification model whose output is a probability value between 0 and 1.

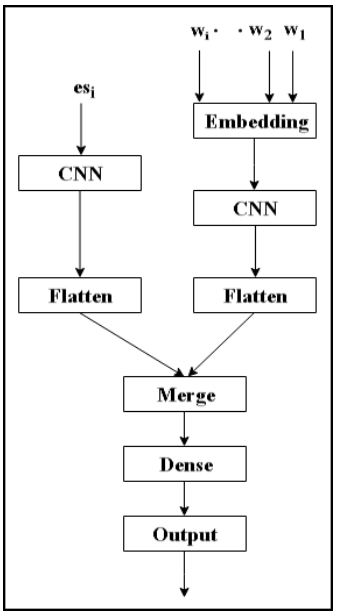

FIGURE 3. Neural network based classifier.

IV. EVALUATION

In this section, first, we construct the research questions to evaluate the proposed approach. Second, we explain the collection process of bug reports. Third, we introduce the metrics and evaluation process. Finally, we discuss the results while answering the research questions.

A. RESEARCH QUESTIONS

We investigate the following research questions to evaluate the proposed approach:

- RQ1: Does the proposed approach outperform the state-of-the-art approaches in severity prediction of bug reports? If yes, to what extent?
- RQ2: How does emotion analysis influence the performance of the proposed approach?
- RQ3: How does the text preprocessing influence the performance of the proposed approach?
- RQ4: Does CNN outperform the traditional machine learning algorithms and LSTM in severity prediction of bug reports?

The first research question (RQ1) investigates the performance improvement of the proposed approach against EWD-Multinomial [28] and severity prediction algorithm (sPredict) [29]. We choose the approaches proposed by Yang *et al.* [28] and Zhang *et al.* [29] for comparison because of the following reasons. First, they provide an automatic severity prediction of bug reports as our approach is. Second, to the best of our knowledge, these are the state-of-the-art approach declared to be more accurate than other related approaches [7], [9], [10].

The second research question (RQ2) examines the influence of emotion analysis. We feed the emotion score of each bug report with its other attributes into the deep neural network for severity prediction of bug reports. RQ2 investigates to what extent emotion analysis affects the performance of the proposed approach.

The third research question (RQ3) examines the influence of the preprocessing on the performance of the proposed approach. We apply natural language preprocessing techniques to clean the textual description of bug reports (mentioned in Section III-D) as most of the written text contain meaningless information e.g., punctuation. RQ3 investigates to what extent preprocessing effects the performance of the proposed approach.

The fourth research question (RQ4) compares the selected deep neural network (CNN) against alternatives. We choose Multinomial Naive Bayes (MNB) and Random Forest (RF) because Yang *et al.* [28] and Lamkanfi *et al.* [7] suggested it as best machine learning algorithms for predicting severity. Whereas, we select Long Short Term Memory (LSTM) because Young *et al.* proved it effective for text classification [30].

B. DATASET

We reuse the dataset created by Lamkanfi *et al.* [31]. They investigated the bug repository of Bugzilla to extract bug reports from *Eclipse* and *Mozilla* projects. They collected bug reports and ignored the duplicate reports and enhancement reports. Both projects *Eclipse* and *Mozilla* contain four products, respectively. From the dataset, we select bug reports of seven open source products. *Platform*, *CDT*, *JDT*, *Core*, *Firefox*, *Thunderbird*, and *Bugzilla*. We ignored bug reports from *GEF* as it contains small number of bug reports. We use summary attribute that defines the bug reports and *severity* attribute that indicates how urgent it is needed to be resolved. The total number of bug reports are *59616* in which approximately *8.39%*, *16.77%*, *16.77%*, *16.77%*, *16.77%*, *16.77%*, and *7.76%* of bug reports belong to each product, respectively.

C. PROCESS

We execute the evaluation of the proposed approach as follows: First, we reuse the bug reports $\mathbb{R}$ of seven open source products from *Eclipse* and *Mozilla* and preprocess each bug report using natural language processing techniques (mentioned in Section III-D). Second, we perform the cross-product validation the given dataset. We divided $\mathbb{R}$ into seven parts based on the project notated as $P_i (i = 1 \ldots 7)$. For the ith cross-validation, we subtract the bug reports that belongs to P_i and select them as testing reports T_e, and the remaining bug reports are combined as training reports T_r. For each cross-validation, the evaluation process as follows:

- First, we combine T_r that is a union of all reports from $\mathbb{R}$ but P_i.

$$T_{r_i} = \bigcup_{i \in [1,7] \wedge j \neq i} P_j \tag{11}$$

- Second, we train a MNB [28] with data from T_r
- Third, we train a RF with data from T_r
- Third, we train a CNN with data from T_r.

- Fourth, we train a LSTM with data from T_r.
- Fifth, we use the trained MNB, RF, CNN and LSTM to predict the severity of each testing report from T_e, and compare their predicted severity with actual severity.
- Finally, we compute the *accuracy*, *precision*, *recall*, *F-measure*, and *MCC* to compare the performance of each algorithm.

D. METRICS

The *accuracy*, *precision*, *recall*, and *f-measure* are well-known metrics that are used to evaluate the performance of classification algorithms. To this end, we compute the severity specific *accuracy*, *precision*, *recall*, and *f-measure* to evaluate the performance of the proposed approach on the given bug reports that can be formalized as,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$F-measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{15}$$

where, *Accuracy*, *Precision*, *Recall*, and *F-measure* are the *precision*, *recall*, and *f-measure* of the approach in predicting severity of bug reports. *TP* is the number of bug reports that are correctly predicted as *severe*, *TN* is the number of bug reports that are correctly predicted as *non-severe*, *FP* is the number of bug reports that are incorrectly predicted as *severe*, and *FN* is the number of bug reports that are incorrectly predicted as *non-severe*.

We also calculates the Mathews Coefficient Correlation (MCC) to measure the quality of the classifier.

$$MCC = \frac{TP*TN - FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{16}$$

E. RESULTS

1) RQ1: COMPARISON AGAINST EWD-MULTINOMIAL

To answer the research question *RQ1*, we compare the *proposed* approach against *EWD-Multinomial* [28] and *sPredict* [29] in severity prediction of bug reports.

The cross-project validation results of the *proposed* approach, *EWD-Multinomial*, and *sPredict* are presented in Tables 2, 3, and 4. For each table, the first column presents the products. Columns 2-6 present the *accuracy*, *precision*, *recall*, *f-measure*, and *MCC* of each approach. Rows 3-9 present the performance results of each approach for each cross-validation, respectively. The last row presents the performance averages of each approach. The average *accuracy*, *precision*, *recall*, *f-measure*, and *MCC* of the *proposed* approach are *88.10%*, *82.64%*, *86.16%*, *84.36%* and *0.286*, respectively. Similarly, the average performance results of the *EWD-Multinomial* are *81.27%*, *76.82%*, *79.70%*, *78.18%*, and *-0.004*, respectively. Furthermore, the average performance results of the *sPredict* are *82.77%*, *76.81%*, *83.79%*, *80.14%*, and *-0.004*, respectively.

TABLE 2. Performance of the proposed approach.

Product	Accuracy	Precision	Recall	F-measure	MCC
CDT	87.81%	80.12%	83.81%	81.92%	0.224
JDT	88.71%	83.89%	88.71%	86.23%	0.398
Platform	88.34%	84.18%	88.35%	86.21%	0.235
Core	89.70%	86.39%	89.70%	88.01%	0.266
Firefox	83.23%	81.61%	83.46%	82.52%	0.33
Thunderbird	89.72%	79.68%	81.94%	80.79%	0.313
Bugzilla	89.18%	82.60%	87.18%	84.83%	0.234
Average	**88.10%**	**82.64%**	**86.16%**	**84.36%**	**0.286**

TABLE 3. Performance of the EWD-Multinomial.

Product	Accuracy	Precision	Recall	F-measure	MCC
CDT	80.62%	80.39%	80.62%	80.50%	-0.003
JDT	86.53%	79.75%	86.53%	83.00%	-0.011
Platform	83.13%	76.03%	83.13%	79.42%	-0.010
Core	83.18%	76.09%	83.18%	79.48%	-0.008
Firefox	74.51%	76.66%	74.51%	75.57%	0.014
Thunderbird	81.91%	76.50%	73.91%	75.18%	0.007
Bugzilla	79.00%	72.34%	76.00%	74.12%	-0.018
Average	**81.27%**	**76.82%**	**79.70%**	**78.18%**	**-0.004**

TABLE 4. Performance of the sPredict.

Product	Accuracy	Precision	Recall	F-measure	MCC
CDT	85.35%	80.60%	86.24%	83.32%	0.006
JDT	84.15%	79.98%	86.10%	82.93%	-0.002
Platform	82.80%	76.00%	83.94%	79.77%	-0.011
Core	83.01%	75.97%	84.08%	79.82%	-0.011
Firefox	79.85%	76.16%	81.27%	78.63%	-0.006
Thunderbird	81.93%	76.40%	84.11%	80.07%	0.002
Bugzilla	82.33%	72.55%	80.79%	76.45%	-0.008
Average	**82.77%**	**76.81%**	**83.79%**	**80.14%**	**-0.004**

TABLE 5. Comparison against the EWD-Multinomial.

	Accuracy	Precision	Recall	F-measure	MCC
Proposed	88.10%	82.64%	86.16%	84.36%	0.286
EWD-Multinomial	81.27%	76.82%	79.70%	78.18%	-0.004
Improvement	**8.40%**	**7.57%**	**8.11%**	**7.90%**	**0.290**

TABLE 6. Comparison against the sPredict.

	Accuracy	Precision	Recall	F-measure	MCC
Proposed	88.10%	82.64%	86.16%	84.36%	0.286
EWD-Multinomial	82.77%	76.81%	83.79%	80.14%	-0.004
Improvement	**6.43%**	**7.59%**	**2.83%**	**5.27%**	**0.290**

The performance improvement of the *proposed* approach upon *EWD-Multinomial* and *sPredict* is presented in Tables 5 and 6. For each table, the first column presents the approaches. Columns 2-6 present their *accuracy*, *precision*, *recall*, *f-measure*, and *MCC*, respectively. The second row and third row present the performance of the *proposed*

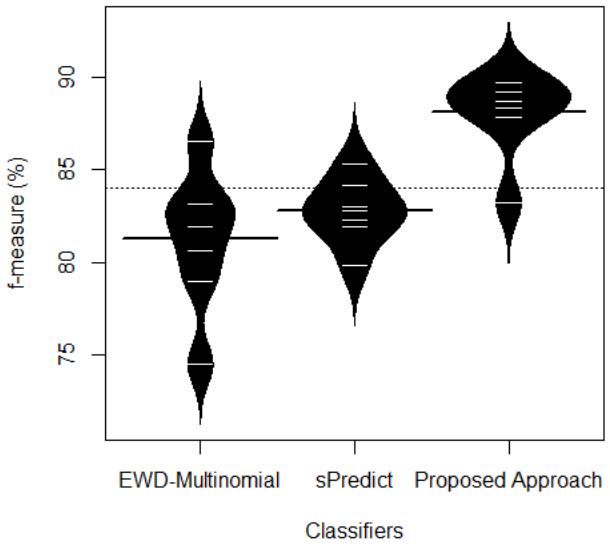

FIGURE 4. Distribution of Accuracy.

approach and the *EWD-Multinomial* or *sPredict*, respectively. The last row presents the improvement of the *proposed* approach upon the *EWD-Multinomial* and *sPredict* in Table 5 and Table 6, respectively.

The accuracy distribution of cross-project validation for the *proposed* approach, *EWD-Multinomial*, and *sPredict* is presented in Fig. 4. The beanplot compares the *f-measure* distributions by plotting one bean for each approach. Across a bean, each short horizontal line represents the *f-measure* on a single project, whereas the long horizontal line represents the average *f-measure*.

From Table 5, Table 6 and Fig. 4, we make the following observations:

- The *proposed* approach outperforms the *EWD-Multinomial* and *sPredict*. The performance improvement of the *proposed* approach in *accuracy*, *precision*, *recall*, *f-measure*, and *MCC* upon *EWD-Multinomial* and *sPredict* is (*8.40%*, *7.57%*, *8.11%*, *7.90%*, and *0.290*) and (*6.43%*, *7.59%*, *2.83%*, *5.27%*, and *0.290*), respectively. One of the possible reasons of the significant improvement in performance is that the *k*-dimensional vector computed with the syntactic and semantic word relationships (mentioned in Section III-F).
- In cross-product validation, the average performance of the *proposed* approach is significantly greater than the best performances of the *EWD-multinomial* and *sPredict*, as shown in Fig. 4.

To validate the significant difference between the *proposed* approach, *EWD-Multinomial*, and *sPredict*, we apply the ANOVA analysis on their evaluation results of *accuracy*, *precision*, *recall*, and *f-measure*. Table 7 shows the results of ANOVA analysis which present $F > F_{cric}$ and $P_{Value} < (alpha = 0.05)$ are true for each *accuracy*, *precision*, *recall*, and *f-measure*. It suggests that the factor

TABLE 7. Results of ANOVA analysis.

Source of Variation	SS	df	MS	F	P-value	F-crit
Accuracy						
Between Groups	0.018028	2	0.009014	12.011373	0.000485	3.554557
Within Groups	0.013508	18	0.000750			
Total	0.031537	20				
Precision						
Between Groups	0.015823	2	0.007911	11.733169	0.000547	3.554557
Within Groups	0.012137	18	0.000674			
Total	0.027960	20				
Recall						
Between Groups	0.014983	2	0.007491	5.916655	0.010596	3.554557
Within Groups	0.022791	18	0.001266			
Total	0.037774	20				
F-measure						
Between Groups	0.013370	2	0.006685	8.223493	0.002905	3.554557
Within Groups	0.014633	18	0.000813			
Total	0.028003	20				

TABLE 8. Influence of emotion analysis.

	Accuracy	Precision	Recall	F-measure	MCC
Enabled	88.10%	82.64%	86.16%	84.36%	0.286
Disabled	85.67%	77.84%	75.52%	76.29%	0.156
Improvement	**2.83%**	**6.16%**	**14.09%**	**10.58%**	**0.130**

(using different approaches) has a significant difference in performance results.

Based on the preceding analysis, we conclude that the *proposed* approach outperforms the *EWD-Multinomial* and *sPredict*.

2) RQ2: INFLUENCE OF EMOTION ANALYSIS

The written text usually express the emotion of the writer. Same is the case, a bug report may contain the emotion of the user who encountered the bug while using software systems. Such emotions may help to automate the severity level assessment of bug reports. In this perspective, we perform an emotion analysis of bug reports.

To answer the research question *RQ2*, we perform performance comparison of *proposed* approach with and without *emotion score* of bug reports. Table 8 presents the evaluation results of the *proposed* approach by enabling and disabling emotion score. The first column presents the emotion analysis input settings. Columns 2-6 present the *accuracy*, *precision*, *rescall*, *f-measure*, and *MCC*. The performance of *proposed* approach upon different settings is presented in the rows of the table. The last row of the table presents the improvement of *proposed* approach upon different input settings for emotion analysis. Fig. 5 also illustrates the performance difference of the *proposed* approach upon different preprocessing input settings by combing the products into two segments.

From Table 8 and Fig. 5, we make the following observations:

- The *proposed* approach with emotions obtains significant improvement in performance. The improvement in *accuracy*, *precision*, *recall*, *f-measure*, and *MCC*

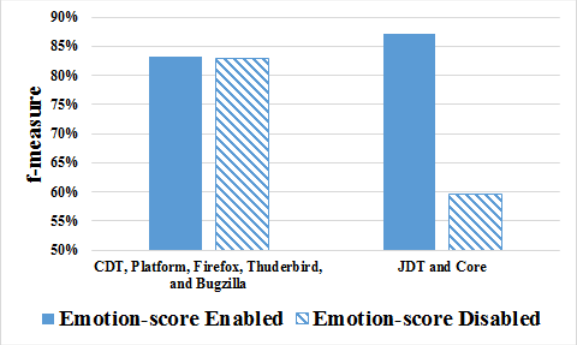

FIGURE 5. Influence of emotion analysis.

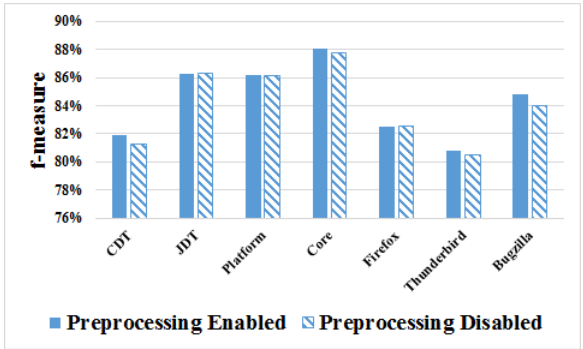

FIGURE 6. Influence of preprocessing.

is *2.83%* = *(88.10% - 85.67%) / 85.67%*, *6.16%* = *(82.64% - 77.84%) / 77.84%*, *14.09%* = *(86.16% - 75.52%) / 75.52%*, *10.58%* = *(84.36% - 74.29%) / 74.29%*, and *0.130* = *0.286 - 0.156* respectively.

- Disabling emotion score significantly decreases the performance in *accuracy* and *f-measure* of the *proposed* approach up to *2.83%* and *10.58%*, respectively. Although emotion score has impact on performance of the proposed approach; however, we observe that the performance on bug reports from *JDT* and *Core* is significantly decreased by disabling emotion score. Therefore, we combine the given bug reports into two segments as shown in Fig. 5. To investigate the reason behind that we analyze the bug reports for emotion words. We observe that *70%* of emotion words belong to *JDT* and *Core*. Whereas, *30%* of emotion words belong to *CDT*, *Platform*, *Firefox*, *Thunderbird*, and *Bugzilla*. However, *45%* of emotion words are common in both segments. The facts suggest that the use of emotion word in a bug report has impact in its severity prediction and may suggest its urgency.
- The decrease in MCC by *0.130* confirms that the *proposed* approach works better with emotion score.

Based on the preceding analysis, we conclude that the emotion analysis of bug reports is an important step to the *proposed* approach.

3) RQ3: INFLUENCE OF PREPROCESSING

The description of bug reports contains irrelevant and unwanted text e.g., punctuation (as mention in Section III-D). Passing irrelevant text as features to the machine learning algorithms increases processing time and require more memory for processing. In this perspective, we perform text preprocessing to improve performance and reduce computational cost.

To answer the research question *RQ3*, we perform performance comparison of *proposed* approach with and without *preprocessing* of bug reports. Table 9 presents the evaluation results by enabling and disabling the preprocessing. The first column presents the preprocessing input settings. Columns 2-6 the *accuracy*, *precision*, *recall*, *f-measure*, and *MCC*, respectively. The performance of *proposed* approach upon different settings is presented in the rows of the table. The last row of the table presents the improvement of *proposed* approach upon different input setting for preprocessing. Fig. 6 also illustrates the performance difference of the *proposed* approach upon different preprocessing input settings.

TABLE 9. Influence of preprocessing.

	Accuracy	Precision	Recall	F-measure	MCC
Enabled	88.10%	82.64%	86.16%	84.36%	0.286
Disabled	85.93%	82.33%	85.92%	84.08%	0.208
Improvement	**2.52%**	**0.37%**	**0.28%**	**0.33%**	**0.078**

From the Table 9 and Fig. 6, we make the following observation:

- The *proposed* approach with the preprocessing obtains significant improvement in performance. The improvement in *accuracy*, *precision*, *recall*, *f-measure*, and *MCC* is *2.52%* = *(88.10% - 85.93%) / 85.93%*, *0.37%* = *(82.67% - 82.33%) / 82.33%*, *0.28%* = *(86.16% - 85.92%) / 85.92%*, *0.33%* = *(84.36% - 84.08%) / 84.08%*, and *0.078* = *0.286 - 0.208* respectively.
- Disabling preprocessing step significantly decreases the performance in *accuracy* and *f-measure* of the *proposed* approach up to *2.52%* and *0.33%*, respectively. Although, the decrease in performance of the *proposed* approach without preprocessing step is minor, however, it cannot be ignored as it reduces the computation time of the algorithm.
- The decrease in MCC by *0.078* confirms that the *proposed* approach works better with preprocessing.

Based on the preceding analysis, we conclude that the text preprocessing of bug reports is also an important step to the *proposed* approach.

4) RQ4: COMPARISON AGAINST ALTERNATIVE MACHINE LEARNING ALGORITHMS

To answer the research question *RQ4*, we apply two well known machine learning algorithms (Multinomial Naive Bayes (MNB) and Random Forest (RF)) and a deep learning

TABLE 10. Influence of machine learning algorithms.

	Accuracy	Precision	Recall	F-measure	MCC
CNN	88.10%	82.64%	86.16%	84.36%	0.286
LSTM	85.98%	78.34%	85.52%	81.69%	0.153
RF	83.24%	77.14%	83.24%	80.07%	0.71%
MNB	79.70%	76.82%	79.70%	78.18%	-0.004

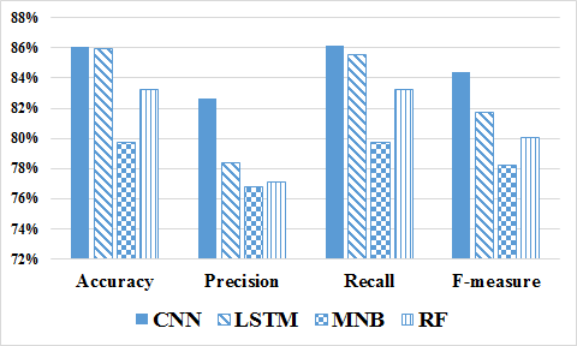

FIGURE 7. Influence of machine learning algorithms.

algorithm (Long Short Term Memory (LSTM) to compare their performances with the *proposed* approach.

Table 10 presents the evaluation results of the algorithms. The first column presents the approaches. Columns 2-6 the *accuracy*, *precision*, *rescall*, *f-measure*, and *MCC*, respectively. The other rows present the performance of *CNN*, *LSTM*, *RF*, and *NBM*, respectively. Fig. 7 also illustrates the performance differences of the algorithms.

From Table 10 and Fig. 7, we make the following observation:

- The *proposed* approach outperforms the MNB, RF and LSTM and obtains the best *accuracy*, *precision*, *recal*, *f-measure*, and *MCC*. One possible reason is that the *proposed* approach build the relationships between hidden vectors.
- The performance of LSTM is significantly higher than the performance of MNB. Consequently, it can be concluded that deep learning approaches outperform the machine learning approaches for severity prediction of bug reports.
- The negative score (*-0.004*) of *MCC* against *MNB* suggests that the *MNB* classifier does not fit for the *proposed* approach.

V. THREATS

In this section, we identify some factors that may affect the performance of the proposed approach. The threats (construct threats, internal threats and external threats) to the validity of this study are as follows.

A. THREATS TO VALIDITY

A threat to construct validity is that the selection of the evaluation metrics. The selected metrics (precision, recall, and f-measure) are most adopted metrics [7], [9], [10], [28], [32]. Therefore, we also adopt these metrics to evaluate the performance of the proposed approach.

Another threat to construct validity is that the utilization of *Senti4SD* to compute the emotion scores of bug reports. We select *Senti4SD* due to its popularity among researchers. It also outperforms the other available tools for emotion analysis [18] (as mentioned in Section III-E). The usage of different emotion calculation repositories may affect the performance of the proposed approach.

A threat to internal validity is that the implementation of the baseline approach. To mitigate the threat, we verify the implementation and results. However, there could be some unseen errors.

A threat to external validity is that the generalizability of our results. We focus on the bug reports from two open-source projects *Eclipse* and *Mozilla* for the evaluation of the proposed approach. The results of the proposed approach are not guaranteed with the inclusion of bug reports from other projects.

Another threat to external validity is that the settings of hyper-parameters for deep learning algorithms. The large number of bug reports as a training set or the adjustment of hyper-parameters may influence the performance of the proposed approach.

VI. RELATED WORK

A. MACHINE LEARNING APPROACHES

Bug reports are generally classified as critical, major, minor and trivial bugs in which critical bugs are more severe and trivial bugs are just inconveniencing to users. To prioritize the severity of the bug reports, a number of machine learning approaches have been proposed. Most of the studies focus on traditional classification algorithms such as Bayesian and Support Vector Machine (SVM).

Menzies and Marcus [8] are the first who proposed an automated solution SEVERIS (SEVERity ISsue assessment) that assigns severity to the bug report. The solution provides fine-grained severity out of 5 severity levels used in NASA for prioritization. They trained a machine learning classifier on the feature vectors with top-k feature words that predict the severity of future bug report.

Lamkanfi *et al.* [7] extended the study of Menzies and Marcus [8]. They applied Menize and Marcus approach on open-source bug repositories to predict the severity of the bug reports. They analyzed the textual description of the bug reports from GNOME, Eclipse, and Mozilla to predict their severity. To avoid the multi-class classification, 5 severity labels out of 6 severity labels are used to group them into two categories i.e., severe and non-severe.

Based on Naive Bayes (NB), decision tree and random forest, Alenezi and Banitaan [33] proposed a solution to execute the priority prediction. They used two feature set based on *tf* weighted words of bug reports and based on operating system and severity classification. The study suggests that the usage of the second feature set performed better than the first feature

set. The study also reveals that random forests and decision trees outperform NB.

Eclipse bug report classification is performed by Gujral *et al.* [34], they have achieved 72% precision and 69% accuracy. Moreover, an algorithm that creates dictionary terms is introduced that predicts severity level by selecting a particular component.

In order to investigate the effectiveness of various classification algorithms, Zhang *et al.* [35] employed multiple machine learning classification algorithms to predict the severity of bug reports. They used 29,204 bug reports and identified that NB multinomial outperforms the NB, 1-nearest neighbor, and SVM.

Using the nearest neighbor approach, Tian *et al.* [36] predicted the severity of bug reports. They used a larger collection of bug reports consisting of more than 65,000 Bugzilla reports. Recently, using SVM Pooja [37] proposed a model for bug severity level that assigns priorities to Firefox crash reports in Mozilla Socorro server based on the frequency and entropy of the crashes.

Kumari *et al.* [38] proposed an approach to assess the severity of bug reports. They used entropy by considering the uncertainty and irregularities in data. They applied KNN, j48, RF, RNG, NB, CNN and MLR as training classifiers and validated them using PITS, Eclipse, and Mozilla projects. The results suggest that their approach significantly outperforms the existing research.

Yang *et al.* [28] constructed an emotion words-based dictionary for verifying bug reports' textual emotion analysis. They modified a machine learning algorithm, the Naive Bayes multinomial, calling the new algorithm EWD-Multinomial. The result shows that the proposed algorithms outperform the severity prediction related approaches [7], [9]. To the best of our knowledge, EWD-Multinomial is the state-of-the-art algorithm to predict the severity of the bug reports. Therefore, we select EWD-Multinomial as a baseline to our approach.

B. DEEP LEARNING

Most of the existing studies focus only on traditional machine learning algorithms e.g., Bayesian and support vector machine. However, recently deep learning has gained tremendous attention of researchers. They have achieved significantly impressive results in the field of computer vision [39], speech recognition [40] and sentiment analysis [41] by using deep learning techniques. Deep learning provides numbers of popular and efficient models, the state-of-the-art studies have not considered deep learning approaches for bug severity of the bug reports. However, some state-of-the-art contributions on bug reports are discussed in the following.

An Artificial Neural Network (ANN) based framework is proposed by Yu *et al.* [42] for an international health-care company. The framework predicts the priorities of five different products of bug reports. Evaluation of threefold cross validation experiments validates, the proposed framework is better in term of precision, recall, and f1-score.

Based on support vector machine NB, KNN and Neural Network, Sharma *et al.* [41] proposed a priority prediction approach. Their approach predicts the priority of the newly arrived bug reports. The study indicates the accuracy of different machine-learning techniques (except NB) can successfully predict the priority of less than 70% bugs from Eclipse and OpenOffice projects.

As a conclusion, researchers have proposed a number of useful machine learning approaches and deep learning approaches to predict the severity of bug reports. Our proposed approach differs from the existing approaches as we are first to predict the severity of bug reports by employing the deep learning algorithms.

VII. CONCLUSION

Users often submit bug reports with their severity level which is an important attribute of a bug that decides how quickly it should be resolved. It helps developers to solve important bugs on time. However, manual severity assessment is a tedious job and could be incorrect. To this end, in this paper, we propose a deep neural network based automatic approach for the severity prediction of bug reports. The proposed approach applies deep learning model, natural language techniques and emotion analysis on the given dataset for the severity prediction of bug reports. The proposed approach automates the severity assessment process and helps users by subtracting the severity assignment step from bug reporting. We perform the cross-project evaluation on the history-data of the open source products of Eclipse and Mozilla. The evaluation results suggest that the proposed approach outperforms the state-of-the-art approaches.

REFERENCES

[1] K. Moran, "Enhancing android application bug reporting," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, New York, NY, USA, Aug. 2015, pp. 1045–1047. doi: 10.1145/2786805.2807557.

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

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

[4] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Proc. 20th Working Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 72–81.

[5] M. Sharma, M. Kumari, R. K. Singh, and V. B. Singh, "Multiattribute based machine learning models for severity prediction in cross project context," in *Computational Science and Its Applications—ICCSA*. Cham, Switzerland: Springer, 2014, pp. 227–241.

[6] 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.

[7] 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.

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

[9] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proc. 7th IEEE Working Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 1–10.

[10] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proc. 19th Work. Conf. Reverse Eng.*, Washington, DC, USA, Oct. 2012, pp. 215–224. doi: 10.1109/WCRE.2012.31.

[11] Q. Umer, H. Li, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.
[12] G. Yang, T. Zhang, and B. Lee, "An emotion similarity based severity prediction of software bugs: A case study of open source projects," *IEICE Trans. Inf. Syst.*, vol. E101.D, no. 8, pp. 2015–2026, 2018.
[13] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *Proc. Workshop Effective Tools Methodologies Teach. Natural Lang. Process. Comput. Linguistics*, vol. 1, Stroudsburg, PA, USA, Jul. 2002, pp. 63–70. doi: 10.3115/1118108.1118117.
[14] TextBlob. (Nov. 2018). *Textblob: Simplified Text Processing*. [Online]. Available: https://textblob.readthedocs.io/en/dev/
[15] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, Feb. 2016, pp. 145–180.
[16] M. R. Islam and M. F. Zibran, "SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text," *J. Syst. Softw.*, vol. 145, pp. 125–146, Nov. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301675
[17] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "SentiCR: A customized sentiment analysis tool for code review interactions," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA, Oct./Nov. 2017, pp. 106–111. [Online]. Available: http://dl.acm.org/citation.cfm?id=3155562.3155579
[18] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Softw. Eng.*, vol. 23, no. 3, pp. 1352–1382, Jun. 2018. doi: 10.1007/s10664-017-9546-9.
[19] F. Calefato, F. Lanubile, and N. Novielli, "EmoTxt: A toolkit for emotion recognition from text," in *Proc. 7th Int. Conf. Affect. Comput. Intell. Interact. Workshops Demos (ACIIW)*, Oct. 2017, pp. 79–80.
[20] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: how far can we go?" in *Proc. 40th Int. Conf. Softw. Eng.*, New York, NY, USA, May/Jun. 2018, pp. 94–104. doi: 10.1145/3180155.3180195.
[21] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Softw. Eng.*, vol. 22, no. 5, pp. 2543–2584, Oct. 2017. doi: 10.1007/s10664-016-9493-x.
[22] L. Ling and S. Larsen, "Sentiment analysis on stack overflow with respect to document type and programming language," Ph.D. dissertation, School Elect. Eng. Comput. Sci., Kth Roy. Inst. Technol., Stockholm, Sweden, 2018.
[23] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. (2013). "Distributed representations of words and phrases and their compositionality." [Online]. Available: https://arxiv.org/abs/1310.4546
[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[25] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." [Online]. Available: https://arxiv.org/abs/1207.0580
[26] Keras. *Flatten Layer*. Accessed: Nov. 1, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/layers/core.py#L467
[27] Keras. *Merge Layer*. Accessed: Nov. 1, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/layers/merge.py
[28] G. Yang, S. Baek, J.-W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proc. Symp. Appl. Comput.*, New York, NY, USA, Apr. 2017, pp. 1280–1287. doi: 10.1145/3019612.3019788.
[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] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.
[31] A. Lamkanfi, J. Péérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, San Francisco, CA, USA, May 2013, pp. 203–206.
[32] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting priority of reported bugs by multi-factor analysis," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Washington, DC, USA, Sep. 2013, pp. 200–209. doi: 10.1109/ICSM.2013.31.
[33] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?" in *Proc. 12th Int. Conf. Mach. Learn. Appl.*, vol. 2, Dec. 2013, pp. 112–116.
[34] S. Gujral, G. Sharma, S. Sharma, and Diksha, "Classifying bug severity using dictionary based approach," in *Proc. Int. Conf. Futuristic Trends Comput. Anal. Knowl. Manage. (ABLAZE)*, Feb. 2015, pp. 599–602. [Online]. Available: https://ieeexplore.ieee.org/document/7154933/authors#authors
[35] 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.*, New York, NY, USA, Apr. 2015, pp. 1553–1558. doi: 10.1145/2695664.2695872.
[36] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Softw. Eng.*, vol. 20, no. 5, pp. 1354–1383, Oct. 2015. doi: 10.1007/s10664-014-9331-y.
[37] P. Choudhary, "Neural network based bug priority prediction model using text classification techniques," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 5, pp. 1315–1319, May/Jun. 2017. [Online]. Available: http://www.ijarcs.info/index.php/Ijarcs/article/view/3559
[38] M. Kumari, M. Sharma, and V. B. Singh, "Severity assessment of a reported bug by considering its uncertainty and irregular state," *Int. J. Open Source Softw. Processes*, vol. 9, no. 4, pp. 20–46, Oct. 2018.
[39] A. Graves, A. Mohamed, and G. E. Hinton. (2013). "Speech recognition with deep recurrent neural networks." [Online]. Available: http://arxiv.org/abs/1303.5778
[40] X. Ouyang, P. Zhou, C. H. Li, and L. Liu, "Sentiment analysis using convolutional neural network," in *Proc. IEEE Int. Conf. Comput. Inf. Technol., Ubiquitous Comput. Commun., Dependable, Autonomic Secure Comput., Pervasive Intell. Comput.*, Oct. 2015, pp. 2359–2364.
[41] M. Sharma, P. Bedi, K. K. Chaturvedi, and V. B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation," in *Proc. 12th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Nov. 2012, pp. 539–545.
[42] 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, Heidelberg: Springer, 2010, pp. 356–367.

WAHEED YOUSUF RAMAY received the B.S. degree in computer science from Islamia University, Pakistan, in 2008, and the M.S. degree in computer science from COMSATS University Islamabad, Pakistan, in 2014. He is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology Beijing, China. He is particularly interested in machine learning, semantic web, and data mining.

QASIM UMER received the B.S. degree in computer science from Punjab University, Pakistan, in 2006, and the M.S. degrees in .net distributed system development and 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.

XU CHENG YIN received the B.Sc. and M.Sc. degrees in computer science from the University of Science and Technology Beijing, China, in 1999 and 2002, respectively, and the Ph.D. degree in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, in 2006. He is currently a Professor and the Deputy Chair with the Department of Computer Science and Technology, School of Computer and Communication Engineering, University of Science and Technology Beijing. His research interests include pattern recognition and machine learning, document analysis and recognition, information retrieval, computer vision, multimedia understanding, and data mining.

CHAO ZHU received the bachelor's degree in automation from Xidian University, Xi'an, China, in 2005, the master's degree in system engineering from Xi'an Jiaotong University, Xi'an, in 2008, and the Ph.D. degree in computer science from the École centrale de Lyon, France, in 2012. He was a Postdoctoral Fellow with the Multimedia Information Processing Lab (MIPL), Institute of Computer Science and Technology (ICST), Peking University, Beijing, China, from 2013 to 2015. He is currently an Associate Professor with the School of Computer and Communication Engineering, University of Science and Technology Beijing. His research interests include deep neural network-based applications, object detection and recognition, feature extraction, and image/video classification.

INAM ILLAHI received the degree from the University of Sargodha, Pakistan, in 2007, and the M.S. degree from the Chalmers University of Technology, Sweden, in 2010. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Beijing Institute of Technology, China. He is particularly interested in software maintenance, crowdsourcing, and machine learning.