IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 4, 2021, accepted August 23, 2021, date of publication August 30, 2021, date of current version September 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3108624

Convolutional Neural Network Based Approval Prediction of Enhancement Reports

JUN CHENG[1], MAZHAR SADIQ[2], OLGA A. KALUGINA[3], SADEEM AHMAD NAFEES[4], AND QASIM UMER[2]

[1]College of Information Engineering, Chaohu University, Chaohu 238000, China
[2]Department of Computer Sciences, COMSATS University Islamabad, Islamabad 45550, Pakistan
[3]Department of English Language for Professional Communication, Financial University under the Government of the Russian Federation, 125167 Moscow, Russia
[4]Department of Computer Science and Information Technology, Virtual University of Pakistan, Lahore 54000, Pakistan

Corresponding author: Qasim Umer (qasimumer@cuivehari.edu.pk)

This work was supported in part by Anhui Key Research and Development Plan under Grant 201904a05020091, and in part by the Key Scientific Research Projects of Chaohu University under Grant xlz-201905.

ABSTRACT For a given software enhancement report, identifying its possible approval status could help software developers by suggesting feature enhancements to compete in the software industry. An automatic solution for the approval prediction of enhancements could assist all the participants in resolving enhancements. The key challenges are the preprocessing of noisy textual information and the state-of-the-art feature models to combine the syntactical and semantic word information available in the given text. To this end, we propose a deep learning based approach for the approval prediction of enhancement reports that incorporates the users' sentiments involved in the text. First, we preprocess the textual information of all enhancement reports to avoid noise. Second, we compute the sentiment of each enhancement report using Senti4SD. Third, we combine the bag-of-words (BOW) representation and traditional word2vec based representation to learn the novel deep representation (a recurrent neural network (RNN) with attention based representation) of preprocessed text. Using an attention mechanism enables the model to remember the context over a long sequence of words in an enhancement report. Fourth, based on sentiment and deep representation, we train a deep learning based classifier for the approval prediction of enhancement reports. Finally, we reuse the 40, 000 enhancement reports from 10 real software applications to evaluate the proposed approach. The cross-application evaluation suggests that the proposed approach is accurate and outperforms the state-of-the-art. The results of the proposed approach improve the precision from 86.52% to 90.56%, recall from 66.45% to 80.10%, and f-measure from 78.12% to 85.01%.

INDEX TERMS Deep learning algorithms, classification, enhancement reports, approval prediction.

I. INTRODUCTION

The users of software applications usually encounter issues and report such issues for maintenance. The reported issues describe bugs (defects) or enhancements (suggestions to improve existing features or to include new features). The enhancement reports (noted as enhancements for short in the rest of this paper) are quite extensive in number for abundant software applications, e.g., enterprise resource planning software. Therefore, companies employ issue tracking systems (e.g., Bugzilla) for the management of enhancements.

The management of enhancements involves human resources (e.g., developers, maintainers, and quality assurance specialists) to read through all enhancements, respond to the essential and meaningful enhancements, and verify and approve their solutions. The manual checking of enhancements to identify the possible approved enhancements is a tedious and time-consuming process. According to Umer *et al.* [1], 76% of the enhancements are rejected due to different reasons, e.g., improper description of the enhancements. Therefore, an automatic solution to identify the approval of enhancements can help all the participants in the management process. To this end, a couple of automated approaches have been proposed for the approval prediction of enhancements [1], [2]. However,

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

the performance of both approaches requires significant improvement.

An enhancement usually contains structured (e.g., priority and severity) and unstructured information (e.g., summary and detailed description). Such information is collected through issue tracking systems. Although state-of-the-art approaches exploit recent preprocessing approaches to filter out the noise (e.g., URLs, hex codes, special characters, and stop-words) from the unstructured text, the adopted feature models (e.g., bag-of-words (BOW) feature representation and word2vec feature representation) do not combine the syntactical and semantic word information available in the text. The concatenation of such information may improve the performance of prediction models [3].

To this end, we propose a deep learning based approach for the approval prediction of enhancements that incorporates the users' sentiments involved in the text. We also propose a novel deep representation of enhancements as a key step of the proposed approach that combines syntactic and semantic information for feature modeling in an unsupervised manner. First, we preprocess the textual information of all enhancements to avoid noise. Second, we compute the sentiment of each enhancement report using Senti4SD. Third, we combine the BOW feature representation and traditional word2vec feature based representation to learn the deep representation of preprocessed text. It uses the attention mechanism that enables the model to learn the context representation over a long word sequence in each enhancement. Fourth, we train a deep learning based classifier for approval prediction of enhancements. Finally, we reuse the history data from real software applications to evaluate the proposed approach. The cross-application evaluation suggests that the proposed approach is accurate and outperforms the state-of-the-art. The results of the proposed approach improve the *accuracy* from 77.90% to 82.15% and *f-measure* from 74.53% to 82.12%.

The paper makes the following contributions:

- A deep learning based approach for approval prediction of enhancement is proposed. To the best of our knowledge, we are the first to exploit a deep learning algorithm in the approval prediction of enhancements.
- A novel deep representation is proposed to represent enhancements for feature modeling that combines the BOW feature representation and traditional *word2vec* representation. It considers the syntactic and semantic features in an unsupervised manner that remembers the context over a long sequence of words.
- The cross-application evaluation of the proposed approach on ten open-source software applications suggests that the proposed approach is accurate and outperforms the state-of-the-art.

We organize the rest of the paper as follows: Section II provides the details of the proposed approach. Section III defines the evaluation process and the evaluation results of the proposed approach. Section IV explains the threats to validity. Section V discusses the related work. Section VI concludes the paper and suggests future directions.

II. APPROACH

A. OVERVIEW

The problem of automatic approval prediction of enhancements can be formulated as a classification problem. The proposed approach categorizes the enhancements into two classes: approved and rejected. Fig. 1 highlights the key steps of the proposed approach, explained as follows:

- First, we collect and reuse the enhancement corpus that contains the summary, description, and status of each enhancement.
- Second, we remove noisy information like URLs, hex codes, and special characters from the text. Because handling such information causes overhead during the training of deep learning model, we also remove the English stop-word and perform spell-check, inflection, and lemmatization.
- Third, we extract a set of unique words from the preprocessed enhancement that have a frequency of at least k-times in the enhancement corpus.
- Fourth, we compute the sentiment of each enhancement using Senti4SD.
- Fifth, we learn each enhancement representation using a deep RNN with attention by combining summary and description as a sequence of words (tokens).
- Sixth, we train deep learning based classifier for approval prediction of enhancements. We pass the sentiment and a novel representation of each enhancement as an input for training.
- We perform cross-application validation to avoid bias for the evaluation of the proposed approach.

B. PROBLEM DEFINITION

An enhancement e from a set of enhancements E can be defined as,

$$e =< d, s > \tag{1}$$

where, d represents the textual information (summary and description) of e and s represents the resolution attribute of e.

The proposed approach suggests the approval of the new enhancement as either *approved* or *rejected*. Consequently, the automatic approval prediction of a new enhancement e can be defined as a function f:

$$f : e \rightarrow c \tag{2}$$

$$c \in \{approved, rejected\}, \quad e \in E \tag{3}$$

where, c is suggested approval status either *approved* or *rejected*.

C. SENTIMENT ANALYSIS

The users of the software applications usually suggest enhancements in writing. Ramay *et al.* reported that the written text involves the sentiment of writers [4]. Therefore,

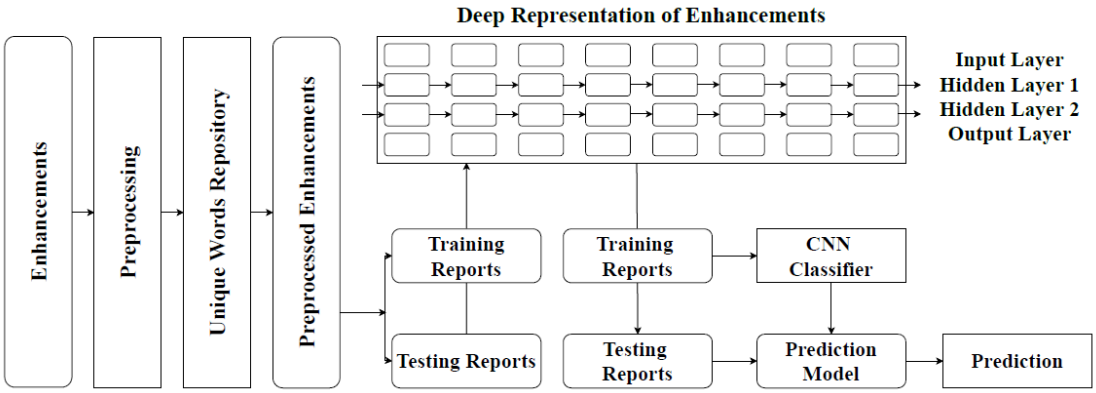

FIGURE 1. Overview of the proposed approach.

we compute the sentiment of each enhancement to identify whether the sentiment involved in the text is positive or negative. Different tools are available to calculate the sentiment from text documents e.g., SentiWordNet [5]. However, a number of researches have been proposed to compute the sentiment from the software engineering text e.g., EmoTxt [6], SentiCR [7], Senti4SD [8], SentiStrengthSE [9], and DEVA [10], whereas the performance of Senti4SD is accurate and outperforms the other software engineering text classification tools. We select Senti4SD for the sentiment analysis due to its significant performance for software engineering text classification.

To compute the sentiment of each enhancement, we input the textual information to Senti4SD. It calculates and returns the sentiment involved in the enhancement based on emotion-words, modifier, and negation in the enhancement [8]. We save the computed sentiment with the corresponding enhancement. An enhancement e with its sentiment can be defined as,

$$e =< d, t, s > \tag{4}$$

where, d, t, and s represent the textual description, sentiment, and resolution attribute of e, respectively.

D. PREPROCESSING

Fig. 2 highlights the key steps of preprocessing. The textual information of enhancements contains noise (e.g., URLs, hex code, special characters, and stop-words) which requires extra processing during the execution of algorithms. Processing of noisy information is an overhead. Therefore, we preprocess each enhancement using Python NLTK package. First, we remove the URLs and hex code, and convert the text into lowercase. Second, each enhancement report is taken to perform tokenization. The tokenization process converts the given text into tokens and assigns them NLP tags. Third, we employ spell-check and remove stop-words (e.g., is, am, and are) and special characters (e.g., @ and -) from the tokenized text. Finally, we apply inflection and lemmatization where inflection transforms plural words into their singular

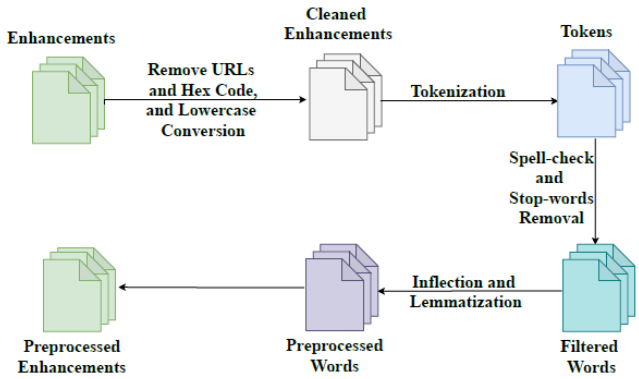

FIGURE 2. Overview of preprocessing.

words (e.g., inflection transforms the words algorithms into algorithm) and lemmatization transforms the comparative and superlative words into their base-words (e.g., lemmatization transforms the word converted into convert). A preprocessed enhancement e with its sentiment can be defined as,

$$e =< d', t, s > \tag{5}$$

where, d', t, and s represent the preprocessed textual description, sentiment, and resolution attribute of e, respectively.

Moreover, we construct a unique words repository using the entire preprocessed enhancement corpus. We remove the words from the repository having a frequency less than k-times in the corpus. Notably, we experiment to figure out the value of k. We observe that the performance of the proposed approach is significant when the threshold of the frequency of unique words is 3.

E. DEEP REPRESENTATION MODEL

A traditional BOW feature representation of an enhancement gives a boolean array or word frequency for each repository word in the enhancement [11]. The BOW does not consider the order of words and semantic similarity among synonyms of the words in the enhancement. Even the bag-of-n-words model feature representation faces the problems of high

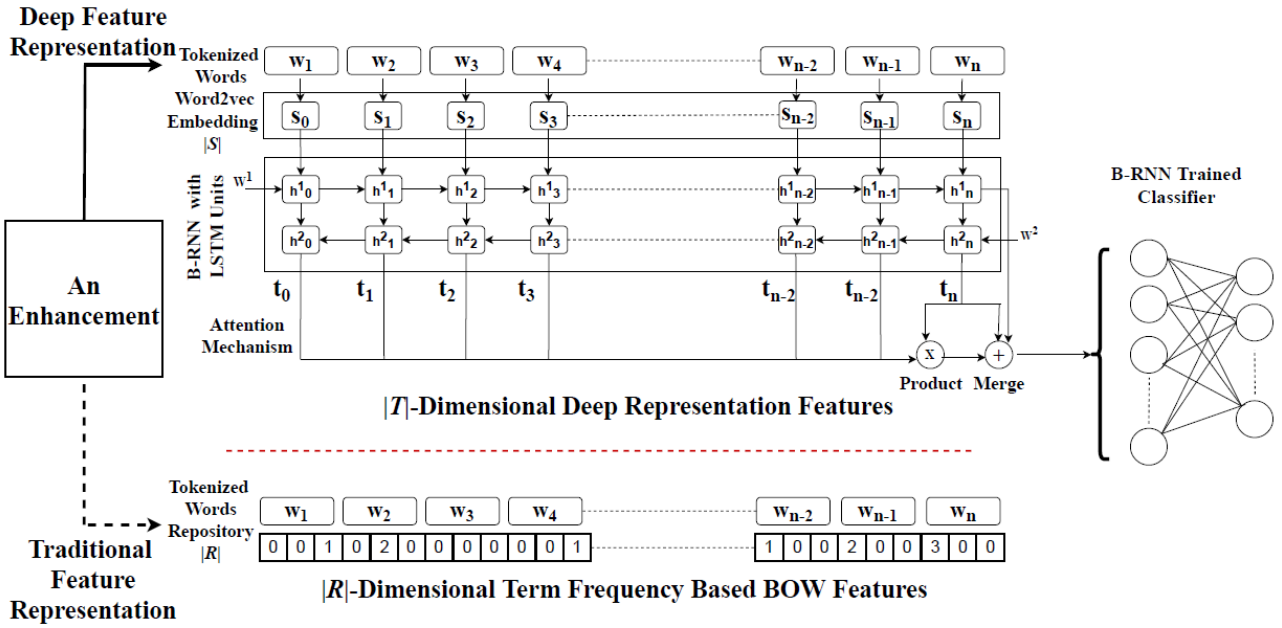

FIGURE 3. Comparison of the deep representation model and the traditional feature representation model.

dimensionality and sparse data [12]. To learn the semantic similarity of the words, a skip-gram based neural network model (word2vec) [13] is proposed. It considers words having the same context in the text have the same semantic meaning. However, word2vec learns a semantic representation of individual words instead of a sequence of words which is the main drawback of word2vec. Consequently, paravec [3] which is an extension of word2vec is proposed. It learns the sequence of words but only for a small context [3]. We observe that a significant improvement is needed in text representation to preserve the sequence of words, the syntax of words, and the semantic relationship of words.

To this end, we proposed a deep representation model for enhancement. Fig. 3 shows the deep representation model of enhancements for feature learning. We use Long Short Term Memory (LSTM) cells [14] in the hidden layer as a memory unit. These cells can memorize the sequence of words and resolve the vanishing gradient problem [15]. Moreover, the proposed deep representation model attends to only extracted words (mentioned in Section II-D) for learning during the classification as an attention mechanism [16]. Notably, the deep representation model memorizes the sequence of words in both forward and backward directions to make the representation more effective for feature learning.

To build the deep representation of each enhancement, we first use the repository to extract the $|R|$-dimensional representation. Second, we learn the $|S|$-dimensional word2vec representation using the extracted $|R|$-dimensional representation. Finally, we determine the deep representation with LSTM cells using the learnt $|S|$-dimensional word2vec representation. It provides the $|T|$-dimensional deep representation of the given enhancement. The deep representation has a RNN which is a sequence network having a hidden layer with n hidden units ($h = h_1, h_2, \ldots, h_n$). The input of RNN is the $|S|$-dimensional word2vec representation ($y = y_1, y_2, \ldots, y_n$) and its output is the $|T|$-dimensional deep representation ($z = z_1, z_2, \ldots, z_n$). Each hidden unit h_i converts the previous state s_{i-1} and a word y_i into the next state s_i and output word z_i. In the RNN, each hidden unit recurrently performs the same function f:

$$f : \{s_{i-1}, y_i\} \rightarrow \{s_i, z_i\} \tag{6}$$

Each state s_i keeps the information of i previous words in the hidden layer h. The output z_n of the last hidden layer h_n represents the complete enhancement. Notably, the function f retains the context of the words in sequence due to the LSTM function [17] that contains built-in memory units to store the contextual information for long text.

Moreover, we employ attention to learn from the important words of the enhancement. We derive the attention vector with the weighted summation of all outputs z_i that can be defined as,

$$a_n = \sum_{i=1}^{n} \alpha_i z_i \tag{7}$$

where, α_i represents the weight of each word y_i that defines the importance of y_i for classification. A bidirectional RNN learns feature representation with input word sequence forward and backward. A complete deep representation of an enhancement (re) can be defined as,

$$re = z_n + a_n + a_n + z_n \tag{8}$$

where, $+$ represents the integration of vectors.

The proposed deep representation model contains 300 LSTM units, 0.2 dropout probability, 0.001 learning rate, and *binary cross-entropy* based loss function with *Adam optimizer*. We set 100 epochs for the training. In contrast to term frequency based BOW representation and word2vec representation, deep representation is much smaller in size. It is reported that the size of deep representation (T) is less than word2vec representation ($4|S|$) (< 1200) when we choose the size of repository as 300 [13] for BOW representation. For example, BOW model produces a feature matrix of size $40,000 * 200,000$ where the enhancements are 40, 000 and the size of the repository is 200, 000. In contrast, the proposed deep representation of the enhancements produces a feature matrix of size $40,000 * 1,200$. Notably, we implement the proposed model in Python Keras Library [18]. To the best of our knowledge, we are the first to apply deep representation to learn the enhancements representation that is further used to learn a deep learning based classifier for approval prediction of enhancements.

F. DEEP LEARNING CLASSIFIER

Fig. 4 shows the composition of the deep learning classifier. We exploit the convolutional neural network (CNN) for approval prediction of enhancement. We choose CNN for the two following reasons. First, CNN is capable of learning the deep semantic relationship between words [4]. Second, it applies different filter sizes to avoid the gradient problem of RNN [19].

We first input the deep representation and sentiment of each enhancement to CNN into two parts. The deep learning classifier contains 3 layers of CNN, filter (number of the neurons) 128, kernel size (size of the filter) 1, loss function *binary-crossentropy*, and activation (the final value of a neuron) *tanh*. Second, we pass the output of the CNN to a flatten layer [20] as input to convert the input into a 1- dimensional vector. Notably, we input the sentiment into a separate CNN with the same setting. Its output is forwarded to a separate flatten layer. Third, the results of both flatten layers are finally merged with merge layer [21] to integrate the results. Finally, we fully connect the neurons between layers using the dense layer and map both inputs (deep representation and sentiment) into a single prediction (output) using the output layer. The output predicts the approval status of the given enhancement.

III. EVALUATION

In this section, we evaluate the performance of the proposed approach (CNN based approval prediction of enhancement (CAAP)) on enhancements from the 10 open-source applications of the Mozilla ecosystem.

A. RESEARCH QUESTIONS

We evaluate the CAAP by investigating the following research questions:

- **RQ1:** Does CAAP outperform the state-of-the-art approaches in the approval prediction of enhancements?
- **RQ2:** How do different inputs (the proposed representation and sentiment) influence the performance of CAAP?
- **RQ3:** How does preprocessing influence the performance of CAAP?
- **RQ4:** Does the proposed deep learning classifier outperform machine/deep learning classifiers in the approval prediction of enhancements?

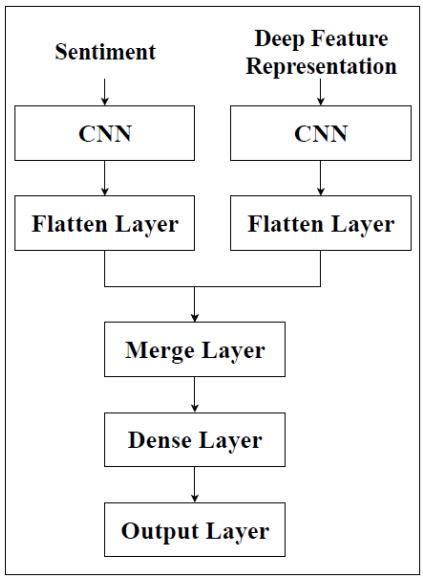

FIGURE 4. Overview of the deep learning based classifier.

The first research question (RQ1) investigates the performance improvement of CAAP against the state-of-the-art approaches. We select three approaches: 1) an automatic approval prediction for enhancements (AAP) [2]; 2) a sentiment-based approval prediction for enhancements (SAAP) [1]; and 3) a machine learning based approval prediction of enhancement reports (MAP) [22] because of the following reasons. First, both AAP, SAAP, and MAP are proposed for the approval prediction of enhancements as our approach CAAP is. Second, to the best of our knowledge, both approaches are proposed recently and are the only approaches that represent the state-of-the-art.

The second research question (RQ2) examines the impact of both inputs (the proposed representation and sentiment). We pass one information at a time to investigate its effect on the performance of CAAP.

The third research question (RQ3) evaluates the performance of CAAP with/without preprocessing to investigate the influence of preprocessing on CAAP.

The fourth research question (RQ4) provides a comparison between the proposed classifier against alternatives. We choose CNN as our classifier because Umer *et al.* [23] recently declared it as the best machine learning algorithm for software engineering documents.

B. DATASET

An overview of the dataset is presented in Table 1. We exploit the only available dataset for enhancements created by

TABLE 1. Description of dataset.

Total number of reports	40,000
Approved reports	9,500 (23.75%)
Rejected reports	30,500 (76.25%)

Nizamani *et al.* [2] and reused by Umer *et al.* [1] and Nafees and Rehman [22]. We collect the enhancements from the history data. They extracted enhancements from Bugzilla using the severity attribute of enhancement. They marked severity attribute as enhancement as an input to the Bugzilla Native REST API to separate enhancements from bugs. We collect enhancements from 10 software applications where each application has more than 2.5% enhancements of the dataset. The total number of collected enhancements in our dataset is 40, 000 in which 12.45%, 4.00%, 3.15%, 18.98%, 2.69%, 18.00%, 5.35%, 20.59%, 10.34%, and 4.45% belong to Bugzilla, Calendar, Camino Graveyard, Core, Core Graveyard, Firefox, MailNews Core, SeaMonkey, Thunderbird, and Toolkit, respectively. Notably, the baseline approaches [1], [2], [22] use the same dataset for the evaluation of their approaches.

C. PROCESS

To evaluate the performance of CAAP, we first preprocess each enhancement as mentioned in Section II-D, compute the sentiment (as mentioned in Section II-C) of each enhancement, and learn its proposed representation as mentioned in Section II-E. Second, we perform cross-application evaluation to reduce the threats to validity in which all enhancements E are divided into 10 portions P_i according to their application where $i = 1, 2, \ldots, 10$. For the i^{th} cross-application evaluation, we collect E that are not from P_i as training set TS and enhancements from P_i as evaluation set ES.

The steps of each i^{th} cross-application evaluation are following:

1) We collect TS from E but P_i.

$$TS_i = \bigcup_{j \in [1,10] \wedge j \neq i} \mathbf{P_j} \tag{9}$$

2) We train a support vector machine (SVM) on TS.
3) We train a long short term memory ($LSTM$) on TS.
4) We train a proposed classifier (convolutional neural network (CNN)) on TS.
5) We train the classifiers from AAP [2], $SAAP$ [1], and MAP [22] on TS, respectively.
6) We predict each enhancement from ES using the trained SVM, AAP, SAAP, MAP, and CNN, respectively.
7) We compute the accuracy (Acc), precision (Pre), recall (Rec), f-measure (FM), Matthews correlation coefficient (MCC), and odds ratio (OR) for each classifier.

To evaluate the performance of CAAP, we select the well-known and most adopted metrics (Acc, Pre, Rec, and FM) that can be defined as:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$Pre = \frac{TP}{TP + FP} \tag{11}$$

$$Rec = \frac{TP}{TP + FN} \tag{12}$$

$$FM = \frac{2 \times Pre \times Rec}{Pec + Rec} \tag{13}$$

where Acc, Pre, Rec, and FM are the accuracy, precision, recall, and f-measure of the approaches to predict the approval status of enhancements. TP is the total number of enhancements that are correctly classified as approved, TN is the total number of enhancements that are correctly classified as rejected, FP is the total number of enhancements that are incorrectly classified as approved, and FN is the total number of enhancements that are incorrectly classified as rejected.

To check the quality and effectiveness of each classifier, we respectively compute MCC and OR.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{14}$$

$$OR = \frac{TP/FP}{FN/TN} \tag{15}$$

D. RESULTS

1) RQ1: COMPARISON AGAINST BASELINE APPROACHES

To answer the research question RQ1, we compare the performance results of CAAP, SAAP, AAP, and MAP. To this end, we perform cross-application validation and present the average evaluation results of all approaches in Table 2. The first column presents the approaches. Columns 2-7 present the Acc, Pre, Rec, FM, MCC, and OR of the approaches, whereas the rows of the table present the performance of CAAP, SAAP, AAP, and MAP, respectively.

We present the FM distribution of cross-application evaluation for CAAP, SAAP, AAP, and MAP in a beanplot (Fig. 5). We plot one bean for each approach to compare the FM distribution of the approaches. In a bean, each horizontal lines represents the FM of the corresponding application, whereas the average FM is presented by a comparatively long horizontal line.

From Table 2 and Fig. 5, we make the following observations:

- CAAP outperforms SAAP, AAP, and MAP. The improvement of CAAP upon SAAP, AAP, and MAP in Acc, Pre, Rec, and FM is up to *(5.46% = (82.15% − 77.90%) / 77.90%, 4.98% = (90.56% − 86.26%) / 86.26%, 20.54% = (80.10% − 66.45%) / 66.45%, 14.06% = (85.01% − 74.53%) / 74.53%), (15.80% = (82.15% − 70.94%) / 70.94%, 88.20% = (90.56% − 48.12%) / 48.12%, 52.31% = (80.10% − 52.59%) / 52.59%, 75.28% = (85.01% − 48.50%) / 48.50%)*, and *(−10.03% = (90.39% − 82.15%) / 82.15%,*

TABLE 2. Performance of *CAAP*, *SAAP*, *AAP*, and *MAP*.

Approach	Accuracy	Precision	Recall	F-Measure	MCC	OR
CAAP	82.15%	**90.56%**	**80.10%**	**85.01%**	**0.492**	**17.005**
SAAP	77.90%	86.28%	66.45%	74.53%	0.487	16.189
AAP	70.94%	48.12%	52.59%	48.50%	0.355	12.491
MAP	**90.39%**	86.52%	65.99%	78.12%	-	-

4.67% = (90.56% − 86.52%) / 86.52%, *21.38% = (80.10% − 65.99%) / 65.99%*, *8.82% = (85.01% − 78.12%) / 78.12%)*, respectively. The possible reasons of the significant improvement in performance are as follows: 1) the deep representation model (mentioned in Section II-E) that does not only learn the semantic representation of individual word but also memorizes the sequence of words to make the representation more effective for feature learning; 2) the proposed deep learning classifier (CNN) is better at extracting local/position invariant features; and 3) Naive Bayes (NB) and SVM proposed by Nizamini *et al.* [2] and Umer *et al.* [1] do not work with variable-high input dimensions.

- Although the accuracy of MAP is higher than the proposed approach, the proposed approach outperforms MAP in Rec as the misclassification rate of MAP is *21.38% = (80.10% − 65.99%) / 65.99%* higher the proposed approach.
- The average results of MCC (*0.492*) $>$ *0* and OR (*17.005*) $>$ *1* are true for CAAP and confirm the quality and effectiveness of CAAP.
- The FM results of CAAP do not drastically fluctuate in contract to SAAP, AAP, and MAP (shown in Fig. 5) and suggest CAAP is more reliable.

Moreover, we perform one-way analysis of variance (ANOVA) and Wilcoxon test to investigate the significant difference between CAAP, SAAP, AAP, and MAP. We employ ANOVA as we apply all approaches to the same set of applications, whereas Wilcoxon test is performed to double-check the significance. Both ANOVA and Wilcoxon test confirm whether the only difference (single factor, i.e., different approaches) leads to the difference in performance. Notably, we compute ANOVA and Wilcoxon test on Excel and Stata with their default setting and do not adjust any parameter. The ANOVA returns the f-ratio value = *49.06987* and p-value = *.00001*, whereas Wilcoxon test returns p-value = *0.0025*. The results of both ANOVA and Wilcoxon test suggest that the factor (using different approaches) has a significant difference at $p < .05$.

Although the CAAP is accurate, we observe many false positives and false negatives. For example, CAAP falsely predicts the rejected enhancement "Option to disallow scripts from hiding toolbars" as approved and accepted enhancement "Automatically decode MacBinary and BinHex files like Mac IE" as rejected. We consider a randomly selected set of *1000* enhancements to investigate the false positives/negatives. We observe that CAAP ignores some of the unique words from the selected enhancements due to the threshold of the frequency of unique words that is 3. For example, the frequency of words (e.g., disallow, MacBinary and BinHex) is less than *3*. Notably, we set the frequency threshold 3 because CAAP works best at the frequency threshold *3*. However, we have not fully understood the rationale for false classifications. In future, we should investigate the rationale for false classification and find out the ways to reduce false classifications.

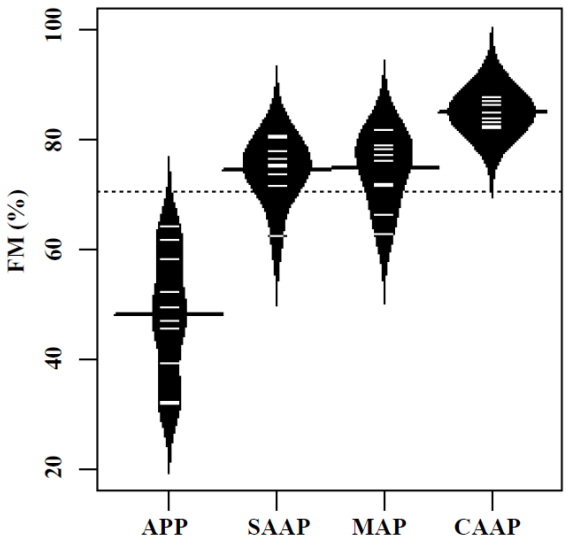

FIGURE 5. Distribution of FM.

Based on the preceding analysis, we conclude that CAAP attains a significant improvement in performance upon SAAP, AAP, and MAP.

2) RQ2: IMPACT OF DIFFERENT INPUTS

To answer the research question RQ2, we compare the performance of CAAP with and without different inputs (sentiment and deep representation (DR)). Evaluation results of CAAP by enabling and disabling different inputs are presented in Table 3. Input settings are given in the first column. Columns 2-7 present the Acc, Pre, Rec, FM, MCC, and OR of CAAP with different input settings, whereas the rows of the table present the performance of CAAP against each input, respectively.

From Table 3, we make the following observations:

- Deep representation (DR) alone is not sufficient for the approval prediction. Although DR alone slightly reduces the performance of CAAP (DR + Sentiment), it helps CAAP in significant improvement in performance upon SAAP and AAP. It reduces the Acc, Pre, Rec, and FM

TABLE 3. Impact of different inputs.

Input	Accuracy	Precision	Recall	F-Measure	MCC	OR
DR + Sentiment	**82.15%**	**90.56%**	**80.10%**	**85.01%**	**0.492**	**17.005**
Sentiment Only	39.77%	51.38%	37.89%	43.62%	0.347	8.253
DR Only	81.94%	87.53%	80.04%	83.62%	0.491	17.003

up to *0.26% = (82.15% − 81.94%) / 81.94%*, *3.46% = (90.56% − 87.53%) / 87.53%*, *0.07% = (80.10% − 80.04%) / 80.04%*, and *1.66% = (85.01% − 83.62%) / 83.62%*, respectively. One of the possible reasons for this reduction is that negative written reports more focus on bugs/problems instead of suggestions/enhancements. Umer *et al.* [1] reported that *71.63%* negatively written reports are rejected where the rejection rate of negatively written reports is *152.48% = (71.63% − 28.37%) / 28.37%*. This finding serves as the rationale to introduce sentiment into the proposed approach.

- Disabling DR (i.e., sentiment only) from the input significantly reduces the performance of CAAP. It significantly reduces the Acc, Pre, Rec, and FM up to *106.56% = (82.15% − 39.77%) / 39.77%*, *76.26% = (90.56% − 51.38%) / 51.38%*, *111.40% = (80.10% − 37.89%) / 37.89%*, and *94.89% = (85.01% − 43.62%) / 43.62%*, respectively. It suggests that sentiment alone is not appropriate for the approval prediction. One of the possible reasons of this reduction is that DR memorizes the semantic relationship among *n*-words (sequence of words) and learns the features in an effective way which is the key of the proposed approach.

To further investigate the impact of inputs in approval prediction, 300 enhancements are randomly selected and manually classified. The manual classification is performed by a team of four software developers and a Ph.D. student. Notably, all participants have rich experience in the management of SE reports. First, they independently classify the enhancements and share their experience to finalize the results. Second, they compare the manual classification with CAAP classification (sentiment only) and observe that 60% and 40% approved enhancements are positive and negative, respectively. Finally, they compare the manual classification with CAAP classification (DR + sentiment) and observe that 18% of approved enhancements (sentiment only classification) are misclassified. For example, rejected enhancement ''Patch: pref to prevent accidental following of links of blocked images'' is classified as approved by CAAP with DR + sentiment. The possible reason is the selection of negative words, e.g., accidental (having frequency $10 > 3$) and prevent (having frequency $41 > 3$) as unique words. The modification in the selection of unique words should be based on the frequency of words and include word polarity (positive/negative). Such modification may provide an effective measurement to reduce misclassification.

Based on the preceding analysis, we conclude that disabling DR significantly affects the performance of CAAP. However, both DR and sentiment are critical for CAAP.

3) RQ3: IMPACT OF PREPROCESSING

The textual information of enhancements contains noise (e.g., URLs, hex code, stop-words, and punctuation). The noise is meaningless and can directly affect the ability to learn any machine/deep learning model. Therefore, preprocessing of textual information is an integral step in machine learning. It improves the performance and reduces the computational cost of the machine/deep learning model.

To answer the research question RQ3, we perform a comparison between the performance results of CAAP by enabling and disabling the preprocessing step. We present the evaluation results of CAAP with and without preprocessing in Table 4. The first column presents the preprocessing input settings. Columns 2-7 present the Acc, Pre, Rec, FM, MCC, and OR of CAAP, whereas the rows of the table present the performance of CAAP upon different preprocessing input settings.

From Table 4, we make the following observations:

- Enabling preprocessing attains significant improvement in performance. It significantly improves the Acc, Pre, Rec, and FM up to *0.83% = (82.15% − 81.47%) / 81.47%*, *3.00% = (90.56% − 87.92%) / 87.92%*, *9.37% = (80.10% − 73.24%) / 73.24%*, and *6.38% = (85.01% − 79.91%) / 79.91%*, respectively. The possible reasons of the improvement are as follows: 1) the textual information of enhancements contains irrelevant and meaningless data, e.g., stop-words and punctuation. Therefore, passing such data to the proposed approach is an overhead. To this end, applying preprocessing may help in performance improvement and computation cost reduction; and 2) the use of Lancaster stemming algorithms instead of Poter stemming algorithm for lemmatization. For example, the output (cri) of a word crying with Poter stemming algorithm has no meaning in sentiment analysis, whereas the output (cry) of the same word with Lancaster stemming algorithm has negative sentiment in emotion analysis.
- Disabling preprocessing results in reduction in MCC and OR. Although it reduces MCC from *0.492* to *0.416* and OR from *17.005* to *16.848*, the results of MCC = *0.416* > *0* and OR = *16.848* > *1* confirm the quality and effectiveness of CAAP without preprocessing.

From the preceding analysis, we conclude that the preprocessing step is critical for the enhancement approval

TABLE 4. Impact of preprocessing.

Preprocessing	Accuracy	Precision	Recall	F-Measure	MCC	OR
Enabled	**82.15%**	**90.56%**	**80.10%**	**85.01%**	**0.492**	**17.005**
Disabled	81.47%	87.92%	73.24%	79.91%	0.416	16.848

prediction, and disabling preprocessing would result in a significant reduction in performance of CAAP.

4) RQ4: COMPARISON AGAINST CLASSIFICATION ALGORITHMS

To answer the research question RQ4, we select SVM and LSTM, because SVM is a best machine learning classifier for software engineering document [1] and LSTM is proven to be effective in natural language processing [24], to compare their performance with CAAP. Notably, we pass same preprocessed enhancements, their sentiment and DR to the selected classifiers for the comparison. We exploit linear SVM with default settings and LSTM with the given settings (dropout = 0.2, recurrent_dropout = 0.2, activation = *sigmoid*, and loss function = *binary-crossentropy*).

We present the evaluation results of classification algorithms in Table 5. Approaches are presented in the first column. Columns 2-7 present the Acc, Pre, Rec, FM, MCC, and OR of the classifiers, whereas the rows of the table present the performance of each classifier, respectively.

From Table 5, we make the following observation:

- The proposed approach CAAP outperforms both machine learning classifier SVM and deep learning classifier LSTM. The performance improvement of CAAP upon SVM in Acc, Pre, Rec, and FM is up to *2.37% = (82.15% − 80.25%) / 80.25%*, *5.52% = (90.56% − 85.82%) / 85.82%*, *4.60% = (80.10% − 76.58%) / 76.58%*, and *5.03% = (85.01% − 80.94%) / 80.94%*, respectively, whereas the performance improvement of CAAP upon LSTM in Acc, Pre, Rec, and FM is up to *1.76% = (82.15% − 80.73%) / 80.73%*, *−1.09% = (90.56% − 91.56%) / 91.56%*, *7.07% = (80.10% − 74.81%) / 74.81%*, and *3.24% = (85.01% − 82.34%) / 82.34%*, respectively.
- The performance of CAAP is better than LSTM. The reason for the decrease is that our input text is long and does not demand sequential preprocessing due to the attention mechanism (mentioned in Section II-E). Another reason is that CNN is better at extracting local/positioninvariant features in contrast to LSTM, and also works well with long input text [25].
- The performance of CAAP is better than SVM. The reason for the decrease is that SVM does not work with variable-high input dimensions as compared to CNN. Although DR (mentioned in Section II-E) greatly reduces the length of feature set, SVM has to deal with variable-high input dimensions which require high computation. Another reason is that CNN does not require feature modeling, which is tedious and time-consuming.

Based on the preceding analysis, we conclude that CNN outperforms other classifiers in the approval prediction of enhancements.

E. THREATS TO VALIDITY

The first threat to construct validity is the possible incorrect labeling of enhancements. Although it is reported that labeling of software engineering reports is not reliable [1], we assume that the reused enhancements are correctly labeled. However, incorrect labeling of enhancements may affect the performance of CAAP.

The second threat to construct validity is the selection of evaluation metrics. The metrics (accuracy, precision, recall, f-measure, MCC, and OR) are selected for the classification of enhancements, because they are the well-known and the most adopted metrics [1], [2], [4].

The first threat to internal validity is the selection of a sentiment analysis repository. We select Senti4SD for CAAP, because it outperforms other available repositories as mentioned in Section II-C. The use of other repositories may affect the performance of CAAP.

The second threat to internal validity is implementation of SAAP and AAP. The implementation and results of the approaches are double checked to mitigate the threat. Some unfold errors may affect the performance of CAAP.

The third threat to internal validity is related to the input settings of the hyper-parameters of CNN. We train the proposed classifier with the setting of a few hyper-parameters as mentioned in Section II-E. The change in other default parameters may affect the performance of CAAP.

A threat to external validity is the reliability of CAAP against other datasets. We evaluate CAAP only on 10 open source applications as mentioned in Section III-C. The inclusion of other inter/intra domain enhancements may affect the performance of CAAP.

IV. RELATED WORK

Many researchers studied the automated classification of software engineering reports (SE-reports) [26]–[34], however, only three studies [1], [2], [22] are focused on enhancements. Most of the state-of-the-art approaches related to the classification of SE-reports mainly predict the severity/priority of SE-reports, incorrect classification of SE-reports, and duplicate SE-reports.

A. PREDICTION OF SEVERITY / PRIORITY

The severity/priority of SE-reports plays a vital role in prioritization. During the last decade, different researches have been conducted for the severity/priority prediction of SE-reports.

TABLE 5. Influence of classification algorithms.

Algorithm	Accuracy	Precision	Recall	F-Measure	MCC	OR
CNN	**82.15%**	**90.56%**	**80.10%**	**85.01%**	**0.492**	**17.005**
LSTM	80.73%	91.56%	74.81%	82.34%	0.484	16.917
SVM	80.25%	85.82%	76.58%	80.94%	0.479	16.794

For the severity prediction, Menzies and Marcus [35] proposed a machine learning based novel approach (SEVERIS) that predicts the severity of SE-reports. Lamkanfi *et al.* [36] and Roy and Rossi [37] collected the reports from Eclipse, GNOME, and Mozilla and found the performance of naive Bayes classifier best for the severity prediction of SE-reports. Chaturvedi and Singh [38] applied different machine learning classifiers on IV and V projects of NASA to predict their severity. Sharma *et al.* [34] took info-gain and chi-square for feature selection and reported that multinomial naive Bayes and k-nearest neighbor perform better for the severity prediction of SE-reports.

For the priority prediction, Abdelmoez *et al.* [39] used the naive Bayes classifier that predicts the priority of SE-reports. They collected data from three large open-source projects, i.e., Mozilla, Eclipse, and GNOME. Based on linear regression, Tian *et al.* [40] proposed an approach (DRONE) for the same purpose and achieved the average F1-score up to 29%. Alenezi and Banitaan [41] used two feature sets (term frequency weighted words based and attribute based) and applied naive Bayes, decision tree, and random forest for the priority prediction. The evaluation results reveal that decision trees outperform the naive Bayes and random forest. Tian *et al.* [42] adopted the nearest neighbor approach to more than 65, 000 Bugzilla reports for the identification of their fine-grained labels. Recently, Choudhary [43] introduced a priority prediction model. The study used a support vector machine to assign priorities of Firefox crash reports based on their frequency and entropy.

B. PREDICTION OF INCORRECT CLASSIFICATION

One of the major challenges in bug prioritization is to deal with the incorrect classification of SE-reports that may delay its resolution.

Antoniol *et al.* [44] proposed an automated classification approach. They used naive Bayes, decision trees, and logistic regression to classify SE-reports of Eclipse and JBoss. They reported that the approach accurately classifies the bugs up to 82%. Herzig *et al.* [45] examined 7000 SE-reports collected from different issue-tracking systems and reported that 33.8% reports are incorrectly classified.

C. PREDICTION OF DUPLICATE SE-REPORTS

Users usually report the same issues multiple times. Automatic identification of duplicate SE reports may save the time and efforts of developers. From this perspective, many studies have been conducted [46]–[48]; however, the following are most significant and influential.

Saric *et al.* [49] proposed a supervised machine learning technique to identify the textual similarity of SE-reports using semantics. Later, Lazar *et al.* [50] improved the performance of Saric *et al.*'s work, whereas Lin *et al.* [33] came up with a different approach that relies on a support vector machine for the duplicate detection of SE-reports. Tian *et al.* [51] also used the support vector classifier for the identification of duplicate SE reports. Feng *et al.* [52] used consumer's profile to detect the duplicate SE reports. Thung *et al.* [47] proposed a support vector based tool (DupFinder) that measures the similarity using summary and description attributes of reports. Notably, Bugzilla uses DupFinder for the identification of duplicate SE-reports.

D. SENTIMENT BASED PREDICTION FOR SE-REPORTS

Jongeling *et al.* [53] presented a comparison study on sentiment analysis tools. They examined the Sentiment analysis tools (SentiStrength, Alchemy, Natural Language Toolkit, and Stanford NLP) and reported the disagreement between tools. They duplicated two existing studies to figure out if the previously published results can be replicated using different sentiment analysis tools. The results reveal that the same results cannot be replicated with a different analysis tool. The study suggests the need for a sentiment analysis tool in the software engineering domain.

Islam and Zibran [9] developed a tool for SE-reports. They used 5600 manually annotated JIRA SE-reports to evaluate the tool. The results reveal that the SentiStrength-SE (a domain dependent tool) outperforms the existing domain-independent tools (SentiStrength, Natural Language Toolkit, and Stanford NLP). Islam and Zibran [10] also introduced a sentiment analysis tool DEVA. The tool performs sentiment analysis and captures the emotion states in SE-reports. They used 1795 JIRA SE-reports for the quantitative evaluation of DEVA. The reported precision and recall of DEVA is more than 82% and 78%, respectively.

Ahmed *et al.* [7] also introduced a sentiment analysis tool for the software engineering domain. They used 2000 manually labeled review comments dataset for one hundred 10-fold cross-validation. They exploited eight supervised learning algorithms (adaptive boosting, decision tree, gradient boosting tree, naive Bayes, random forest, multilayer perceptron, support vector machine with stochastic gradient descent, and linear support vector machine) to evaluate the proposed approach. The evaluation results reveal that gradient boosting tree outperforms the other algorithms and gives the highest mean accuracy, precision, and recall up to 83%, 67.8%, and 58.4%, respectively.

Calefato *et al.* [6] proposed a toolkit (EmoTxt) for the identification of emotion from text. The EmoTxt is trained and evaluated on a gold standard of about 9K question, answers, and comments. The empirical evidence reveals that EmoTxt achieves comparable performance with datasets collected from Stack Overflow and JIRA.

A sentiment analysis tool (Senti4SD) is developed by Calefato *et al.* [8] that supports developers in their communication channels. The tool is trained and validated with a gold standard of Stack Overflow questions, answers, and comments. The dataset is manually annotated for sentiment polarity. The evaluation results show that the tool decreases the misclassification of neutral and positive posts as emotionally negative.

Werder and Brinkkemper [54] developed a tool called MEME-a Method for Emotion Extraction to extract emotion from the software engineering text. The authors collected data from GHtorrent and GitHub to evaluate the tool. The results show that MEME is a better tool as compared to Syuzhet R package emotion analysis.

Marshall *et al.* [55] carried out an experiment to observe the effect of emotional post content on project performance. Three Scrum Sprints consisting of five teams members produced thirteen hundred forum posts. A manual sentiment analysis evaluation technique suggests that emotional posts in software development team communication need intervention.

Williams and Mahmoud [56] presented a preliminary study to detect, classify, and interpret emotions in the tweets of software users. They collected and used 1000 tweets from a wide range of software systems, e.g., Twitter. The evaluation results suggest that naive Bayes and support vector machine can accurately detect general and specific emotions expressed in software-relevant tweets. Moreover, a position paper is presented by Fountaine and Sharif [57] on the effects of emotional awareness on the progress of developers that implicates the impact of emotion in software development.

Graziotin *et al.* [58] shared the experienced consequences of unhappiness (emotions) among developers and found 49 consequences of unhappiness. Graziotin *et al.* [59] also observed the effects of happiness and unhappiness in software development. The study carried out a qualitative analysis and reported the consequences of both happiness and unhappiness that are beneficial to the mental well-being of developers.

Lin *et al.* [60] built a software library based on the developers' opinions collected from Stack Overflow. From the collected dataset, 40K manually labeled sentences are used to examine the accuracy of the tools (SentiStrength, Natural Language Toolkit, Stanford CoreNLP, and EmoTxt) that are famous for identifying the sentiment of software engineering text. The results suggest that Stack Overflow does not deal with emotions; instead, it is a place for developers to discuss technicalities.

Umer *et al.* [23] presented an emotion-based automatic approach (eApp) for the priority classification. They adopted a distinct method and employed emotion analysis for the priority classification. They adopted the support vector machine for the multiclass prioritization of bug reports. The evaluation results show that the performance improvement in F1-score by more than six percent. Umer *et al.* [1] also proposed an approach for the approval prediction of enhancement. They employed sentiment and exploited the support vector machine for the approval prediction. They used cross-application validation for the evaluation of the proposed approach. The results suggest that the proposed approach increases the accuracy from 70.94% to 77.90% and f-measure from 48.50% to 74.53%.

In conclusion, researchers have proposed many approaches for predicting different attributes, e.g., severity, priority, and enhancement approval. However, only three studies [1], [2], [22] focus on the approval prediction of enhancement. Moreover, the studies exploit machine learning algorithms for the approval prediction of enhancement. Our proposed approach differs from the existing approaches in that we are the first to apply a deep learning algorithm for the approval prediction of enhancements. Moreover, we also introduce a novel enhancement representation (input) for the deep learning algorithm.

V. CONCLUSION AND FUTURE WORK

This paper proposes a novel representation of enhancements for feature modeling that considers the syntactic and semantic feature in an unsupervised manner to remember the context over a long sequence of words. We learn the representation of each enhancement using RNN with attention (mentioned in Section II-E) by combining the summary and description of the enhancement as a sequence of words. We also incorporate the sentiment of reporters and compute the sentiment of each enhancement. Given the proposed representation and sentiment of each enhancement, we train a convolutional neural network based classifier (mentioned in Section II-F) for the approval prediction of enhancements. In the end, we perform a cross-application evaluation of the proposed approach on reused 40, 000 enhancements of 10 open-source software applications. The results of the proposed approach improve the precision from 86.52% to 90.56%, recall from 66.45% to 80.10%, and f-measure from 78.12% to 85.01%.

The broader impact of our work is to show that the proposed representation of enhancements could be valuable in the approval prediction of enhancements. In future, we would like to exploit deep learning algorithms on a large number of inter/intra domain enhancements to generalize the results of the proposed approach.

ACKNOWLEDGMENT

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

REFERENCES

[1] Q. Umer, H. Liu, and Y. Sultan, "Sentiment based approval prediction for enhancement reports," *J. Syst. Softw.*, vol. 155, pp. 57–69, Sep. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219301104

[2] Z. A. Nizamani, H. Liu, D. M. Chen, and Z. Niu, "Automatic approval prediction for software enhancement requests," *Automated Softw. Eng.*, vol. 25, no. 2, pp. 347–381, Oct. 2017, doi: 10.1007/s10515-017-0229-y.

[3] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, 2014, pp. II-1188–II-1196. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3045025

[4] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, vol. 7, pp. 46846–46857, 2019.

[5] S. Baccianella, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," in *Proc. LREC*, 2010, pp. 1–5.

[6] 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.

[7] 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. 2017, pp. 106–111. [Online]. Available: http://dl.acm.org/citation.cfm?id=3155562.3155579

[8] 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.

[9] 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

[10] M. R. Islam and M. F. Zibran, "DEVA: Sensing emotions in the valence arousal space in software engineering text," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, Apr. 2018, pp. 1536–1543, doi: 10.1145/3167132.3167296.

[11] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, p. 10, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000794

[12] P. Devanbu, "On the naturalness of software," in *Proc. 6th India Softw. Eng. Conf. (ISEC)*, Piscataway, NJ, USA, 2013, pp. 837–847. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337322

[13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. New York, NY, USA: Curran Associates Inc., 2013, pp. 3111–3119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999959

[14] S. Hochreiter and J. J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 80–1735, 1997.

[15] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Jan. 2014, pp. 338–342.

[16] M.-T. Luong, H. Pham, and C. Manning, "Effective approaches to attention-based neural machine translation," Aug. 2015, *arXiv:1508.04025*. [Online]. Available: https://arxiv.org/abs/1508.04025

[17] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[18] (Jan. 2015). *Keras*. [Online]. Available: https://keras.io/

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*. [Online]. Available: http://arxiv.org/abs/1207.0580

[20] Keras. *Flatten Layer*. Accessed: Nov. 1, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/layers/core.py#L467

[21] Keras. *Merge Layer*. Accessed: Nov. 1, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/layers/merge.py

[22] S. Nafees and F. Rehman, "Machine learning based approval prediction for enhancement reports," in *Proc. Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Jan. 2021, pp. 377–382.

[23] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.

[24] 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.

[25] B. Wang, "Disconnected recurrent neural networks for text categorization," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Melbourne, NSW, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2311–2320. [Online]. Available: https://www.aclweb.org/anthology/P18-1215

[26] S. Youn and D. Mcleod, "A comparative study for email classification," Seongwook, Los Angeles, CA, USA, Tech. Rep. 90089, 2006.

[27] W. Gad and S. Rady, "Email filtering based on supervised learning and mutual information feature selection," in *Proc. 10th Int. Conf. Comput. Eng. Syst. (ICCES)*, Dec. 2015, pp. 147–152.

[28] I. Santos, C. Laorden, B. Sanz, and P. G. Bringas, "Enhanced topic-based vector space model for semantics-aware spam filtering," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 437–444, Jan. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417411009961

[29] Y. Zhang, S. Wang, P. Phillips, and G. Ji, "Binary PSO with mutation operator for feature selection using decision tree applied to spam detection," *Knowl.-Based Syst.*, vol. 64, pp. 22–31, Jul. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095070511400104X

[30] Z. Jin, Q. Li, D. Zeng, and L. Wang, "Filtering spam in Weibo using ensemble imbalanced classification and knowledge expansion," in *Proc. IEEE Int. Conf. Intell. Secur. Inform. (ISI)*, May 2015, pp. 132–134.

[31] R. Moraes, J. F. Valiati, and W. P. G. Neto, "Document-level sentiment classification: An empirical comparison between SVM and ANN," *Expert Syst. Appl.*, vol. 40, no. 2, pp. 621–633, Feb. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417412009153

[32] B. Baharudin, L. H. Lee, and K. Khan, "A review of machine learning algorithms for text-documents classification," *J. Adv. Inf. Technol.*, vol. 1, no. 1, Feb. 2010.

[33] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *J. Syst. Softw.*, vol. 121, pp. 223–233, Nov. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121216000546

[34] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Comput. Sci.*, vol. 70, pp. 632–639, May 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915032238

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

[36] 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.

[37] N. K.-S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2014, pp. 269–276.

[38] 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.

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

[40] 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.*, Oct. 2012, pp. 215–224.

[41] 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.

[42] 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.

[43] 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, 2017. [Online]. Available: http://www.ijarcs.info/index.php/Ijarcs/article/view/3559

[44] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: A text-based approach to classify change requests," in *Proc. Conf. Center Adv. Stud. Collaborative Res. Meeting Minds (CASCON)*, New York, NY, USA, 2008, p. 23, doi: 10.1145/1463788.1463819.

[45] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA, May 2013, pp. 392–401. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486840

[46] S. Banerjee, B. Cukic, and D. Adjeroh, "Automated duplicate bug report classification using subsequence matching," in *Proc. IEEE 14th Int. Symp. High-Assurance Syst. Eng.*, Washington, DC, USA, Oct. 2012, pp. 74–81, doi: 10.1109/HASE.2012.38.
[47] F. Thung, P. S. Kochhar, and D. Lo, "DupFinder: Integrated tool support for duplicate bug report detection," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, New York, NY, USA, Sep. 2014, pp. 871–874, doi: 10.1145/2642937.2648627.
[48] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA, May 2013, pp. 183–192. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487123
[49] F. Šarić, G. Glavaš, M. Karan, J. Šnajder, and B. D. Bašić, "Takelab: Systems for measuring semantic text similarity," in *Proc. 1st Joint Conf. Lexical Comput. Semantics, Main Conf. Shared Task, 6th Int. Workshop Semantic Eval.* vol. 2. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 441–448. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387636.2387708
[50] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, New York, NY, USA, 2014, pp. 308–311, doi: 10.1145/2597073.2597088.
[51] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2012, pp. 385–390.
[52] L. Feng, L. Song, C. Sha, and X. Gong, "Practical duplicate bug reports detection in a large web-based development community," in *Proc. Asia–Pacific Web Conf.* Sydney, NSW, Australia: Springer, 2013, pp. 709–720.
[53] 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. [Online]. Available: https://doi.org/10.1007/s10664-016-9493-x
[54] K. Werder and S. Brinkkemper, "MEME-toward a method for emotions extraction from GitHub," in *Proc. IEEE/ACM 3rd Int. Workshop Emotion Awareness Softw. Eng. (SEmotion)*, Jun. 2018, pp. 20–24.
[55] A. Marshall, R. F. Gamble, and M. L. Hale, "Outcomes of emotional content from agile team forum posts," in *Proc. 1st Int. Workshop Emotion Awareness Softw. Eng.*, May 2016, pp. 6–11.
[56] G. Williams and A. Mahmoud, "Analyzing, classifying, and interpreting emotions in software Users' tweets," in *Proc. IEEE/ACM 2nd Int. Workshop Emotion Awareness Softw. Eng. (SEmotion)*, Piscataway, NJ, USA, May 2017, pp. 2–7, doi: 10.1109/SEmotion.2017.1.
[57] A. Fountaine and B. Sharif, "Emotional awareness in software development: Theory and measurement," in *Proc. IEEE/ACM 2nd Int. Workshop Emotion Awareness Softw. Eng. (SEmotion)*, Piscataway, NJ, USA, May 2017, pp. 28–31, doi: 10.1109/SEmotion.2017.12.
[58] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "Consequences of unhappiness while developing software," in *Proc. IEEE/ACM 2nd Int. Workshop Emotion Awareness Softw. Eng. (SEmotion)*, Piscataway, NJ, USA, May 2017, pp. 42–47, doi: 10.1109/SEmotion.2017.5.
[59] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *J. Syst. Softw.*, vol. 140, pp. 32–47, Jun. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218300323
[60] 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 2018, pp. 94–104, doi: 10.1145/3180155.3180195.

JUN CHENG received the master's degree in computer science and technology from Anhui University, China, in 2012. He is currently a Lecturer with the School of Information Engineering, Chaohu University. His research interests include network security and cloud computing technology.

MAZHAR SADIQ was born in Pakistan. He received the bachelor's and master's degrees in computer science from Pakistan, the Master of Science degree from Blekinge Institute of Technology, Sweden, in 2007, and the Ph.D. degree from the University of Jyvaskyla, Finland, in 2016. After completing the master's degree, he went to Sweden for higher education. Since March 2017, he has been working as an Assistant Professor with COMSATS University Islamabad–Sahiwal, Sahiwal, Pakistan. He has taught human–computer interaction and machine learning courses to bachelor level students. He has supervised many master of computer science students. He is the author of more than ten articles. His research interests include usability, virtual reality, machine learning, and augmented reality.

OLGA A. KALUGINA received the university degree from Mordovian State Pedagogical Institute named after M.E. Evseviev, Russia, in 2005. She is currently an Associate Professor in the area of professional cross-cultural communication and economics with the Financial University under the Government of the Russian Federation. She is interested in environmental economics and sustainable development. Her current research interest includes the development of practical tools for software engineers.

SADEEM AHMAD NAFEES received the B.S. degree in computer science from COMSATS University Islamabad, in 2016, and the M.S. degree from the National University of Sciences and Technology, in 2020. He is currently a Tutor with the Computer Science Department, Virtual University of Pakistan. His research interests include text mining, machine learning, deep learning, and software engineering.

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 from the University of Hull, U.K., in 2009, the M.S. degree in computer science from the University of Hull, in 2013, and the Ph.D. degree from Beijing Institute of Technology, China. He is currently an Assistant Professor with the Department of Computer Sciences, COMSATS University Islamabad, Vehari Campus, Pakistan. His particular research interests include machine learning, data mining, software maintenance, and developing practical tools to assist software engineers.