Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

In Practice

Sentiment based approval prediction for enhancement reports

Qasim Umer, Hui Liu*, Yasir Sultan

School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China

ARTICLE INFO

Article history:
Received 11 January 2019
Revised 12 May 2019
Accepted 15 May 2019
Available online 16 May 2019

Keywords:
Enhancement reports
Machine learning algorithms
Classification

ABSTRACT

The maintenance and evolution of the software application is a continuous phase in the industry. Users are frequently proposing enhancement requests for further functionalities. However, although only a small part of these requests are finally adopted, developers have to go through all of such requests manually, which is tedious and time consuming. To this end, in this paper we propose a sentiment based approach to predict how likely enhancement reports would be approved or rejected so that developers can first handle likely-to-be-approved requests. This could help the software applications to compete in the industry by upgrading their features in time as per user's requirements. First, we preprocess enhancement reports using natural language preprocessing techniques. Second, we identify the words having positive and negative sentiments in the summary attribute of the enhancements reports and calculate the sentiment of each enhancement report. Finally, with the history data of real software application, we train a machine learning based classifier to predict whether a given enhancement report would be approved. The proposed approach has been evaluated with the history data from real software applications. The cross-application validation suggests that the proposed approach outperforms the state-of-the-art. The evaluation results suggest that the proposed approach increases the accuracy from *70.94%* to *77.90%* and improves the F-measure significantly from *48.50%* to *74.53%*.

1. Introduction

Software applications are becoming dynamic nowadays and businesses are frequently changing the requirements of such applications. The users of the software applications may propose many suggestions on how to improve them. Such suggestions, if not related to their bugs, are often specified as enhancement reports (noted as reports for short in the rest of this paper). The number of reports may be quite large for a popular software application that has a large number of users. The management of a plurality of reports is a challenging task for businesses. Therefore, they have adopted issue-tracking systems to collect and track such reports.

Developers and maintainers are involved in the resolution and management of reports. They have to go through all of such reports manually in order to respond to useful and important reports. Developers submit their solutions to adopted reports and the quality assurance specialists verify and approve their solutions by marking them *VERIFIED*. However, most of the reports are rejected for different reasons, e.g., improper description of the reports and business considerations [1]. Our analysis on 40,000 reports of different software applications suggests that up to *76.25%* of the reports are rejected as shown in Table 2.

Although only a small portion of the reports would be approved, developers have to check all of them in case of missing any useful reports. However, it is tedious and time consuming to manually check such a large number of reports. Therefore, an automated approval prediction of the reports will be a great relief for developers. With automated and accurate approval prediction for reports, developers may inspect the more likely to be approved requests first (Nizamani et al., 2017). The benefit is two fold. First, rank based inspection may help to find a larger number of useful suggestions from users with limited human resource. Second, developers may respond more quickly to useful and important reports, which is critical to the success of software applications.

To this end, we propose a sentiment based approach for the approval prediction of reports. We reuse the history data of report related to Mozilla ecosystem which is extracted by Nizamani et al. (2017). Each report from the history data is preprocessed e.g., tokenization, Parts-of-Speech (POS) tagging, spelling correction, handle negation, recognizing modifier words, stopwords removal, word inflection, and lemmatization. From the preprocessed reports, we identify useful features for training, i.e., frequently used words in the reports. We also identify the words having positive and negative sentiments in the summary attribute

* Corresponding author.
E-mail addresses: qasimumer667@hotmail.com (Q. Umer), liuhui08@bit.edu.cn (H. Liu).

https://doi.org/10.1016/j.jss.2019.05.026
0164-1212/© 2019 Elsevier Inc. All rights reserved.

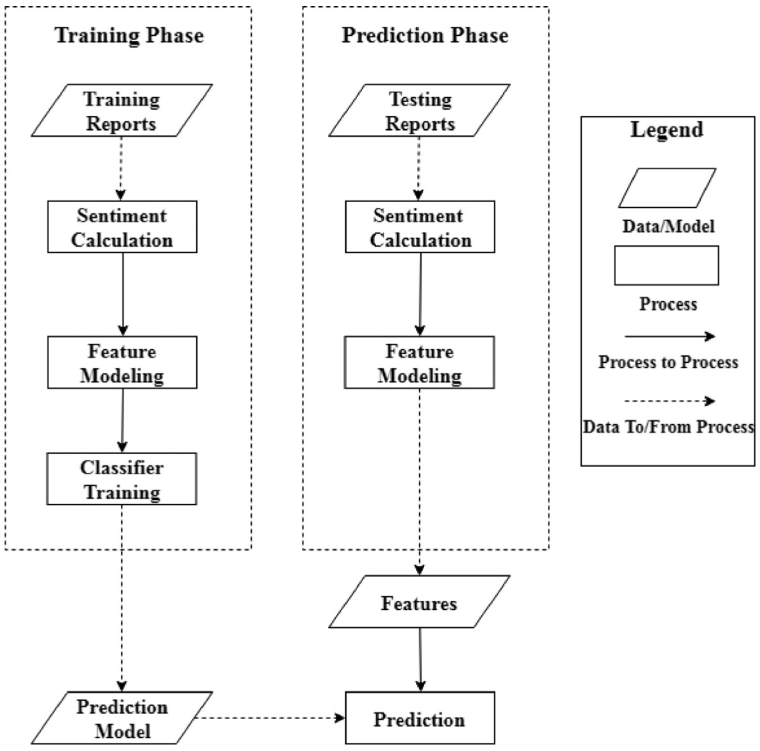


Fig. 1. Overview of the approach.

of the report and calculate its sentiment. Finally, we use the features for training and testing.

Sentiment analysis could help to predicate how likely reports would be approved or rejected because of the following reasons. First, reports written in negative fashion often focus on complaining rather than constructive suggestions, and consequently are rejected frequently. For instance, a report like "*user interface is not attractive, incomplete and need to modify*" complains without any constructive suggestion. By contrast, a report like "*user interface should have a combination of white and blue colors, and navigation can be improved by adding home button*" provides the constructive suggestion in a positive way. The latter has a better chance to be approved because of its insightful and constructive suggestions. Another possible reason for the usefulness of sentiment analysis is that developers, as other human beings, are spontaneously more likely to accept suggestions specified in a positive fashion. Our analysis (shown in Table 2) on the history data confirms the hypothesis: 71.63% of the negatively written reports are rejected, whereas only 28.37% of the positively written reports are rejected.

In order to evaluate the proposed approach, we train a classifier and test the trained classifier on different sets of reports using cross-application validation technique. The results of the cross-application validation suggest that the proposed approach is accurate and it outperforms the state-of-the-art approach. It increases the accuracy from *70.94%* to *77.90%* and improves the F-measure significantly from *48.50%* to *74.53%*.

The main contribution of the paper is as follows:

- A sentiment based approach to predict how likely reports would be approved or rejected so that developers can first handle likely to-be-approved requests. To the best of our knowledge, we are the first to employ the sentiments of the reports in approval prediction of reports.
- Evaluation of the proposed approach on real software applications whose results suggest that the proposed approach is accurate.

The remaining sections of the paper are organized as follows: Section 2 explains the proposed approach in detail. Section 3 reports the evaluation of the proposed approach, results and threats to validity respectively. Section 4 discusses the related works and differences. Finally, Section 5 concludes the paper and suggests future work.

2. Approach

2.1. Overview

The proposed approach classifies reports into two categories: approved or rejected. Fig. 1 represents the overview of the proposed approach. It predicts the approval of reports in two main phases. In the first phase, we preprocess the reports of the real software applications using natural language processing techniques. Sentiment of each report is also calculated in this phase. Each preprocessed report and its sentiment is converted into a feature vector. In the second phase, we use the feature vectors of reports and their corresponding category label for the training of a classifier. The trained classifier is applied to vectors of testing reports for their approval prediction.

The approval prediction of a new report *er* into a defined category *c* could be defined as a function *f*,

$$c = f(er) \quad c \in \{approved, rejected\}, \quad er \in ER \tag{1}$$

where *c* represents the classification result: approved or rejected, *f* represents the classification function of approval prediction, *er* represents a report which is an input of the function, and *ER* represents a set of reports.

In order to predict approval status of the reports, we reuse the history data of report from an issue tracking system which is extracted by Nizamani et al. (2017), preprocess each report and compute the sentiment of each report in order to extract its features. Then, we train the machine learning classifier to predict whether

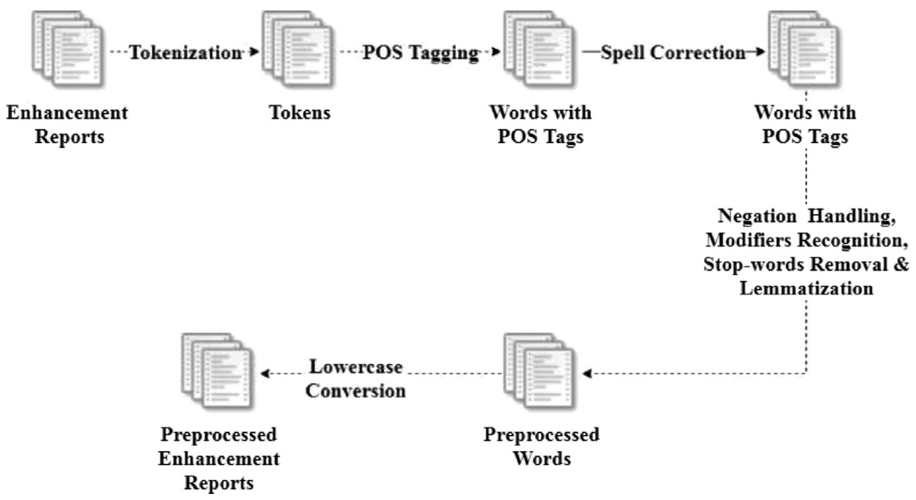


Fig. 2. Preprocessing.

a given report would be approved. In the following sections, we introduce each of the key steps of the approach.

2.2. Data acquisition

The software application issues are generally reported into an issue tracking system which allows users to report the problems, keeps track of registered problems and provides a platform to the developer to resolve the registered problems to maintain a good quality product. We use only reports of real software applications extracted from Bugzilla.[1] A report (er) is formalized as,

$$er = < d, r > \tag{2}$$

where d represents the textual description of a report and r represents its resolution attribute.

2.3. Sentiment calculation

Sentiment calculation of a text is a primary task of sentiment analysis of the writer in natural language processing. To classify the sentiment of a given text of a report whether the expressed opinion of the reporter in the report is positive or negative (Bird, 2002), we calculate the sentiment of each report. There are numbers of sentiment analysis tools available to calculate the sentiment of text document e.g., SentiWordNet (Baccianella et al., 2010). However, to the best of over knowledge, SentiStrengthSE (Islam and Zibran, 2018b), SentiCR (Ahmed et al., 2017), Senti4SD (Calefato et al., 2018), EmoTxt (Calefato et al., 2017), and DEVA (Islam and Zibran, 2018a) tools can be used for the classification of software engineering text. We select *Senti4SD* for two reasons. First, it represents the state of the art. It outperforms the SentiStrength, SentiStrengthSE, and SentiCR in classifying software engineering documents (Calefato et al., 2018). Second, it is widely used in software engineering (Jongeling et al., 2017; Lin et al., 2018), and it is proved efficient and effective.

Senti4SD leverages three different kinds of features, namely lexicon-based, keyword-based, and semantic ones. Lexicon-based features are based on sentiment lexicons. The approach is totally independent of the lexicon chosen and simply requires that a sentiment score is provided for each entry of the input. Keyword-based features are n-grams extracted from dataset. The current implementation of the tool considers uni- and bi-grams. Other than including n-grams *Senti4SD* was designed to include features related to micro-blogging. Semantic features are based on word representation in a distributional semantic model specifically trained on software engineering data. The semantic features capture the similarity between the vector representations of the Stack Overflow documents and prototype vectors representing the polarity classes in a DSM, built using positive, negative, and neutral words from the sentiment lexicon used in the first feature set. The semantic features are computed on a DSM built on Stack Overflow data, using the CBOW architecture implemented by word2vec on a corpus extracted from the Stack Overflow.

We pass the textual description d of each report er to the *Senti4SD* that computes the sentiment of the given report. The sentiments are stored with the corresponding reports. A report with its sentiment is formalized as,

$$er = < d, s, r > \tag{3}$$

where d represents the preprocessed textual description of a report er, s represents the sentiment of a report er, and r represents the resolution attribute.

2.4. Preprocessing

We use *Python Natural Language Toolkit (NLTK)* (Loper and Bird, 2002) for the preprocessing, which includes tokenization, POS tagging, spelling correction, negation handling, modifier words recognition, stop-words removal, word inflection, and lemmatization. Each report has gone through the phases of preprocessing and stored for the feature extraction. Fig. 2 depicts all the phases involved in preprocessing.

First, we tokenize each report into words using white spaces involved in its summary attribute and each token is tagged with a POS tag. Second, spelling check is performed e.g., spelling correction changes *falibility* with *fallibility*. We combine the negation and the modifier of the sentiment words in this step, to calculate the sentiment of the report as mentioned in Section 2.3. Third, the stop-words *(as a, am, the, about, and so on)* and special characters *(hyphen, @, and abbreviations)* are removed sequentially from the list of tokenized words. Fourth, we perform inflection and lemmatization of all filtered words. Inflection of the words converts them into singular form and lemmatization transforms the superlative and comparative words into their root words because both have same meanings. For example, the word *spaces* is converted into *space* after inflection and the word *Deleted* is transformed into *delete* after lemmatization. Note that passing words to lemmatizer

[1] https://www.bugzilla.org/, accessed on 30/11/2018.

without their POS information may lead to incorrect lemmatization. For example; an input (*eating* with and without its POS (*v*)) to lemmatizer returns *eat* and *eating*, respectively. Therefore, we integrate POS information before performing lemmatization. Finally, we fold all the words into lower case and store the preprocessed reports. A preprocessed report is formalized as,

$$er = < d', s, r > \tag{4}$$

$$d' = < t_1, t_2, \ldots, t_n > \tag{5}$$

where d' represents the preprocessed textual description of a report *er*, *s* represents the sentiment of *er*, *r* represents the resolution attribute, and $t_1, t_2, \ldots, t_n$ represents the *n* tokens involved in d'.

2.5. Feature modeling

We create a matrix for the given dataset of reports. The matrix contains the sentiment and features of reports where the features are the list of words from all reports. In matrix, each row comprehensively represents a preprocessed report whereas, columns represent the sentiment and features in it. A report in the matrix is defined as,

$$er = < s, f_1, f_2, \ldots ..f_n > \tag{6}$$

where *er* represents a report, *s* and f_1, f_2, ... f_n represent the sentiment and the features of the report respectively.

Equation (6) is used to model each report. We count the frequency (*N*) of each feature to populate the matrix. In this regard, the following conditions are applied to each report to assign the value of each feature.

$$f_i(er) = \begin{cases} 0, & if \quad t_i \notin d' \\ 1, & if \quad t_i \in d' \end{cases} \tag{7}$$

where $\boldsymbol{f_i}$ is a set of features of preprocessed textual description d' of a report *er*.

2.6. Training and prediction

Support vector machines algorithm can manage the high dimensional feature space and removes the irrelevant features (Khan et al., 2010; Tan et al., 2009). The support vector machine classification depends on the structural risk management principle from the computational learning theory that guarantees the lowest true error (Schölkopf et al., 1999; Vapnik, 1999). Support vector machine algorithm requires binary training set to seek a decision surface that separates both classes into n-dimensional space called hyperplane (Khan et al., 2010; Joachims, 1998). Support vector machine algorithm has a significant impact for the classification of the documents (Khan et al., 2010; Chakrabarti et al., 2002; Lin, 2002). Therefore, we train the support vector classifier using the feature sets of each report which are tagged while feature modeling using equation (6). The trained classifier is tested to predict the approval of reports.

2.6.1. Training

To construct a classification model that assigns an unlabeled report to a class *c* as defined in Eq. (1), a set of reports *ER* = *(er*$_1$, *er*$_2$, ..., *er*$_n$*)* is given. Each report from *ER* contains its classification category *c* as mentioned in Eq. (1), and sentiment *s* and set of features f_i as mentioned in Eq. (6). We train support vector classifier for the approach. We use the approved and rejected reports to seek the decision surface for the approval prediction of reports. The decision surface is a hyperplane which is used for the training of proposed approach. Given the reports $\boldsymbol{er_i}$ labeled into specified binary categories y_i, find a weight vector **w** such that discriminant function separates the categories for $i = 1, 2, \ldots n$ and can be formalized as

$$f(\mathbf{er_i}) = \mathbf{w}^\top \mathbf{er_i} + b \tag{8}$$

where *f* is the discriminant function which calculates the decision surface for the training data **er** of enhancement reports, **w** is the normal to decision surface which is known as weight vector and *b* is bias.

2.6.2. Prediction

Once the vector is defined with the training set, each report from the testing dataset may be approved if

$$\mathbf{w}^\top \mathbf{er} + b > 1 \tag{9}$$

and rejected otherwise.

3. Evaluation

In this section, first we construct the research questions for the evaluation of the proposed approach. Second, we explain the collection process of reports. Third, we define the metrics and evaluation process. Finally, we highlight the findings and threats to validity while answering the research questions.

3.1. Research questions

The evaluation is performed to investigate the following research questions:

- RQ1: Does sentiment of reports influence their possibility to be approved?
- RQ2: Does the proposed approach outperform the state-of-the-art approaches?
- RQ3: Does re-sampling help to improve the performance of the proposed approach? If yes, to what extent?
- RQ4: Does support vector machine outperform other classifiers in terms of enhancement approval prediction?

The first research question (RQ1) investigates the influence of the sentiment of the reports to predict their approval. It evaluates whether the reports which are written in a positive fashion are more likely to be accepted or vice versa.

The second research question (RQ2) provides the comparison of our approach with the state-of-the-art (Nizamani et al., 2017), named as *Nizamani's Approach* in this paper. We observe that Nizamani's approach is related to our domain and also predicts the approval status of reports. To the best of our knowledge, it's the only one that can predict the approval of reports. Therefore, the proposed approach is compared with it.

The third research question (RQ3) investigates the impact of re-sampling. Our dataset is imbalanced and contains almost double rejected reports, therefore, we perform re-sampling. Re-sampling can be performed either over-sampling, under-sampling or changing the value of the classifier threshold. Over-sampling can be performed by adding data into an imbalanced dataset, whereas, under-sample can be performed by reducing the data in an imbalanced dataset. Similarly, changing the value of the classifier threshold to balance the weight of each class may help to balance the dataset. We adopt the under-sampling technique to balance our dataset and use both balanced and imbalanced dataset for our approach to observe the change in performance. In cross-application validation, we use an equal number of approved and rejected reports for each product in order to create the real testing condition.

The fourth research question (RQ4) investigates the impact of different classification algorithms on the proposed approach. To answer RQ4, we compare the proposed approach with other machine learning classifiers to evaluate the performance (accuracy, precision, recall, and F-measure) of the proposed approach.

3.2. Dataset

We reuse the history data of reports (available online[2]) related to Mozilla ecosystem. Nizamani et al. (2017) extracted them from Bugzilla using *Bugzilla Native REST API*. They use severity attribute to access the reports from Bugzilla. The value of severity attribute can vary from *blocker, critical, major, normal, minor, trivial* and *enhancement*. Therefore, they specify the value of severity attribute as *enhancement*[3] in the URL of Bugzilla REST API to filter the reports from bugs reports. We reuse the dataset (Nizamani et al., 2017) of 40,000 reports of software applications in which *23.75%* and *76.25%* reports are *approved* and *rejected* respectively. We only select those software applications that are having reports more than *3%* of the dataset. Therefore, the reports from *Bugzilla, Calendar, Camino Graveyard, Core, Core Graveyard, Firefox, MailNews Core, SeaMonkey, Thunderbird* and *toolkit* are taken for the evaluation of the proposed approach. It contains *12.45%, 4.00%, 3.15%, 18.98%, 2.69%, 18.00%, 5.35%, 20.59%, 10.34%* and *4.45%* of the reports of the applications respectively.

3.3. Process

The evaluation of the proposed approach is performed as follows, the given reports are preprocessed as mentioned in Section 2. Second, we perform cross-application validation. The *ER* of ten products are divided into ten segments S_i where$i = (1,2,3,....,10)$. For the *i*th cross-validation, we select segments of reports *ER* that are not from S_i as Training Data (D_{train}) and reports from S_i as Testing Data (D_{test}).

For the *i*th cross-validation, a step by step evaluation process is as follows:

- First, we extract and combine all the reports D_{train} from *ER* but S_i.

$$D_{train_i} = \bigcup_{j\in[1,10]\ \wedge\ j\neq i} \mathbf{S_j} \tag{10}$$

- Second, we train a Naïve Bayes classifier (*NB*) on D_{train}.
- Third, we train a Multinomial Naïve Bayes classifier (*MNB*) on D_{train}.
- Fourth, we train a Bernoulli Naïve Bayes classifier (*BNB*) on D_{train}.
- Fifth, we train a Logistic Regression classifier (*LR*) on D_{train}.
- Sixth, we train a Random Forest classifier (*RF*) on D_{train}.
- Seventh, we train a proposed classifier (*SV*) on D_{train} with both 1) enabling and disabling sentiment of reports and 2) balanced and imbalanced dataset.
- Eighth, we train Nizamani et al. (2017) classifier on D_{train}.
- Ninth, for each report from S_i, we predict whether it could be approved using trained *MNB, BNB, LR, RF, proposed approach* and Nizamani's approach respectively.
- Finally, we calculate the accuracy, precision, recall, F-measure, Matthews correlation coefficient and odds ratio for each classifier.

We carry out a cross-application validation technique for the performance evaluation to reduce the threats to validity. We divide the reports of each product and use one product for testing and the rest of them are used for training, for each fold of cross-application validation. The machine learning classifiers are trained and tested to evaluate the performance of the proposed approach on the given reports. To this end, we select the well known metrics (accuracy, precision, recall, and F-measure) that are used to evaluate the performance of classification algorithms. We compute the approval specific accuracy, precision, recall, and F-measure of the approaches that can be formalized as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$F{-}measure = \frac{2 * Precision * Recall}{Pecision + Recall} \tag{14}$$

where, *Accuracy, Precision, Recall* and *F-measure* are the accuracy, precision, recall and F-measure of the approaches in predicting approval of reports. *TP* is the number of reports that are correctly predicted as approved,

TN is the number of reports that are correctly predicted as rejected, *FP* is the number of reports that are incorrectly predicted as approved, and *FN* is the number of reports that are incorrectly predicted as rejected.

We also compute the Mathews correlation coefficient (MCC) and odds ratio (OR) to measure the quality and effectiveness of the classifier, respectively.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{15}$$

$$OR = \frac{TP/FP}{FN/TN} \tag{16}$$

In order to answer the research questions, we compute the above mentioned metrics for each of the trained classifier. First, we compare the results of the proposed approach with and without sentiment of reports to answer the RQ1. Second, we compare the results of the proposed approach and state-of-the-art approach to answer the RQ2. Third, we compare the results of the proposed approach with balanced and imbalanced dataset to answer RQ3. Finally, we compare the results of proposed classifier with the existing classifiers to answer the RQ4.

3.4. Results

3.4.1. RQ1: Influence of sentiment

The evaluation results of the proposed approach for *sentiment only* (mentioned in 2.3), *features only* (mentioned in 2.5), and *sentiment with features* are presented in Table 1. From this table, we make the following observation:

- Sentiment alone (i.e., without features) is often insufficient for approval prediction. Compared to the default setting (i.e., enabling both sentiment and features), disabling features decreases accuracy significantly from 77.9% to 47.35%, F-measure from 74.53% to 48.81%, and MCC from 0.487 to 0.361.
- Second, disabling sentiment (i.e., features alone) results in significant reduction in performance as well. It significantly reduces precision from 86.28% to 64.37%, recall from 66.54% to 55.17%, and OR from 16.189 to 13.028.

From the preceding analysis, we conclude that both sentiment and features are critical for approval prediction, and disabling either of them would result in significant reduction in performance.

Table 2 shows the finding of the correlation between sentiment and approval status of reports. It presents that *71.63%* negatively written reports are rejected, whereas only *28.37%* of the positively written reports are rejected. The rejection rate *152.48% = (71.63% -*

[2] https://github.com/shanniz/Bugzilla.

[3] https://bugzilla.mozilla.org/rest/bug?severity=enhancement, accessed on 30/06/2018.

Table 1
Influence of sentiment.

Input	Accuracy	Precision	Recall	F-Measure	MCC	OR
Features + Sentiment	77.90%	86.28%	66.45%	74.53%	0.487	16.189
Sentiment Only	47.35%	60.99%	40.69%	48.81%	0.361	9.179
Features Only	76.99%	64.37%	55.17%	58.97%	0.462	13.028

Table 2
Relation between sentiment and approval status.

Reports	Total	Positive	Negative
Approved	9501 (23.75%)	82.62%	17.38%
Rejected	30,499 (76.25%)	28.37%	71.63%

Table 3
Words frequently associated with sentimentally negative reports.

Word	Probability
Inadequate	91.67%
Malicious	80.67%
Nasty	77.78%
Pain	80.67%
Crucial	90.91%
Funny	80.77%
Stupid	92.91%
Tedious	92.87%
Violate	90.91%
Worried	78.95%

28.37%) / 28.37% of negatively written reports is higher than positively written reports. One possible reason for the significant difference is that negative reports more focus on problems instead of suggestions. On the other hand, a report written in a bad manner may be considered as a bug report. This finding serves as the rationale for us to introduce sentiment into approval prediction of reports.

The performance of a sentiment analysis tool may vary in different dataset and context. To investigate how accurate *Senti4SD* is in our case, we randomly select *100* sample reports and manually check the results of *Senti4SD*. The manual checking is accomplished by three software engineering professionals. All of them are software developers and have rich experience in handling enhancement reports. They classify the sentiment of the reports independently, and then discuss together to remove inconsistencies through a *Skype* conference call. Finally, we compare the sentiments generated by *Senti4SD* against those by the professionals. Results suggest that *Senti4SD* is accurate in our case, and it correctly computes the sentiment for *91%* reports.

We also compute the significance of each word effecting the likelihood of reports having negative sentiment. The list of the words that are most associated with the reports having negative sentiment is presented in Table 3. It is calculated in general for all application instead of each individual application. We employ TF/IDF to measure the significance of the words in a document and sort the significant words with the highest occurrences in the reports. We combine all the reports having negative sentiment to compute the term frequency (TF) and take the complete set of reports to compute the inverse document frequency (IDF). The probability of a word can be formalized as,

$$P(w_i, ns) = \frac{count(d_{ns}, w_i)}{count(d_{total}, w_i)} \quad (17)$$

where, $P(w_i, ns)$ represents the probability of word w_i having negative sentiment ns, $count(d_{ns}, w_i)$ represents the number of reports having negative sentiment that includes w_i, and count (d_{total}, w_i) represents the total number of reports that includes w_i.

Consequently, avoiding words having negative meaning may increase the approval likelihood of reports as the rejection rate of negatively written reports (having more negative words) is higher than the positively written reports.

3.4.2. RQ2: Comparison against state-of-the-art approach

The cross-application validation results of the proposed approach and the Nizamani's approach are presented in Table 4. Results suggest that the average accuracy, precision, recall, F-measure, MCC and OR of the proposed approach are *77.90, 86.28%, 66.45%, 74.53%, 0.487* and *16.189* respectively. Similarly, the performance results of Nizamani's approach to cross-application validation suggest its average accuracy, precision, recall, F-measure, MCC and OR are *70.94%, 48.12%, 52.59%, 0.355* and *12.491* respectively.

The accuracy distribution of cross-application validation for the proposed approach and Nizamani's approach is presented in Fig. 3. The beanplot compares the accuracy distributions by plotting one bean for each approach. Across a bean, each small horizontal line represents the accuracy on a single application, whereas the long horizontal line represents the average accuracy.

From Tables 4 and Fig. 3, we make the following observations:

Table 4
Performance of the Proposed Approach and the State-of-the-art Approach.

	Proposed approach						State-of-the-art approach					
Application	Acc	Pre	Rec	FM	MCC	OR	Acc	Pre	Rec	FM	MCC	OR
Bugzilla	74.66%	75.16%	53.34%	62.40%	0.456	8.821	60.23%	55.64%	75.70%	64.14%	0.324	4.831
Calendar	78.66%	95.79%	63.61%	76.45%	0.420	13.880	54.08%	58.82%	39.26%	47.09%	0.332	6.763
Camino Graveyard	77.90%	97.12%	61.68%	75.44%	0.436	20.451	71.08%	51.63%	23.17%	31.98%	0.366	7.879
Core	78.64%	70.36%	73.08%	71.69%	0.546	12.295	65.64%	53.65%	72.40%	61.63%	0.398	9.651
Core Graveyard	72.70%	90.91%	59.27%	71.76%	0.286	4.706	64.92%	39.75%	76.00%	52.20%	0.222	4.771
Firefox	80.02%	75.94%	71.46%	73.63%	0.576	14.773	85.03%	45.11%	55.13%	49.62%	0.361	17.535
MailNews Core	73.76%	97.92%	68.65%	80.72%	0.506	35.330	81.93%	50.51%	41.50%	45.57%	0.364	18.391
SeaMonkey	83.29%	76.49%	73.83%	75.14%	0.626	21.094	56.56%	29.29%	44.30%	35.26%	0.348	14.667
Thunderbird	79.52%	88.35%	73.52%	80.25%	0.604	19.178	89.79%	40.51%	38.01%	39.22%	0.450	29.796
Toolkit	79.90%	94.75%	66.01%	77.81%	0.414	11.367	80.13%	56.30%	60.48%	58.31%	0.383	10.631
Average	**77.90%**	**86.28%**	**66.45%**	**74.53%**	**0.487**	**16.189**	**70.94%**	**48.12%**	**52.59%**	**48.50%**	**0.355**	**12.491**

Where, Acc = Accuracy, Pre = Precision, Rec = Recall, FM = F-Measure, MCC = Mathews correlation coefficient, and OR = Odds Ratio.

Table 5
ANOVA Analysis on the Comparison.

Source	SS	DF	MS	F	*p*-**value**	***F***$_{crit}$
Accuracy						
Between Groups	0.04199	1	0.04199	4.61545	0.04554	4.41387
Within Groups	0.16379	18	0.00909			
Total	0.20579	19				
Precision						
Between Groups	0.72799	1	0.72799	72.83804	9.63933E-08	4.41387
Within Groups	0.17990	18	0.00999			
Total	0.90789	19				
Recall						
Between Groups	0.09592	1	0.09592	5.04352	0.03751	4.41387
Within Groups	0.34232	18	0.01902			
Total	0.43823	19				
F-measure						
Between Groups	0.33871	1	0.33871	46.07457	2.33899E-06	4.41387
Within Groups	0.13232	18	0.00735			
Total	0.47104	19				

Where, SS = sum of squares, DF = degree of freedom, MS = mean square, F = F-test (used for comparing models), and p-value determines the significance.

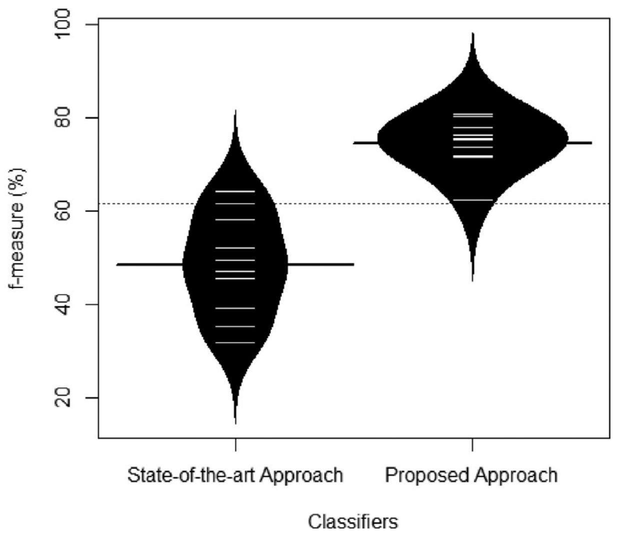


Fig. 3. Distribution of F-measure.

- The proposed approach outperforms Nizamani's approach in both accuracy and F-measure. The proposed approach improves the accuracy by *9.82% = (77.90% - 70.94%) / 70.94%* and F-measure by *53.66% = (74.53% - 48.50%) / 48.50%* respectively. One of the possible reasons for the significant improvement in performance is that the proposed approach considers sentiments of reports. As validated in Section 3.4.1, the sentiment analysis has a significant impact on the approval of reports. Consequently, feeding an additional feature (sentiment of reports) into the classifiers is likely to improve the performance of the classification.
- The average results of MCC ($0.487 > 0.355$) > 0 and OR ($16.189 > 12.491$) > 1 are true for proposed approach and confirm the quality and effectiveness of the proposed approach.
- The minimum F-measure results of the proposed approach is higher than the maximum F-measure results of the Nizamani's approach (shown in Fig. 3).

Although the proposed approach is accurate, it also results in many false positives and false negatives. For example, rejected enhancement report "Should be able to manually open a blocked pop-up" is predicted as *approved* whereas approved report "Firefox fails to show where a webpage was saved from" is predicted as *rejected*. We have not yet fully understood the rationale for the misclassification. It is likely that to avoid such misclassifications, we should exploit additional factors (e.g., cost and potential benefits of implementing the enhancement reports) besides the description of reports. In future, we should further investigate the rationale for misclassifications, and figure out effective measurements to reduce misclassifications.

To validate the significant difference between the proposed approach and Nizamani's approach, we employ (one-way) ANOVA because both approaches are applied to the same projects. It may validate whether the only difference (single factor, i.e., different approaches) leads to the difference in performance. We conduct ANOVA on Excel with its default setting, and no adjustment is involved. Notably, ANOVA on accuracy, precision, recall, and F-measure is conducted independently, where the unit of analysis is a project. Table 5 shows the results of ANOVA analysis which presents $F > F_{cric}$ and *p-value* $<$ *(alpha = 0.05)* are true for precision and F-measure. It suggests that the factor (using different approaches) has a significant difference in precision and F-measure. Whereas, the *p*-values of accuracy and recall not statistically significant. One possible reason is that the accuracy (recall) of the same approach varies dramatically. For example, the accuracy of the proposed approach varies dramatically from 54.08% to 89.79%. As a result, variation within groups is even greater than that between groups, and thus the *p*-value is not significant.

We also evaluate the proposed approach to check the performance of the proposed approach for each class (approved and rejected) separately. The results suggest that the percentage of the approved and rejected reports that are classified correctly is *66.45%* and *86.90%*, respectively.

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

3.4.3. RQ3: Effect of Re-sampling

Re-sampling is used to adjust the class distribution of the dataset which involves the usage of a bias to equal the samples of classes. The usual reason for the sampling is to correct the bias in the original dataset. In order to analyze the effect of re-sampling, we consider both under-sampling and over-sampling approaches. Note that we select *n* samples at random from the majority class for under-sampling, where *n* is the number of samples for the mi-

Table 6
Impact of Re-sampling.

Re-sample	Accuracy	Precision	Recall	F-Measure	MCC	OR
No	77.90%	86.28%	66.45%	74.53%	0.487	16.189
Under-sampling	91.79%	60.13%	98.33%	74.51%	0.734	40.891
Over-sampling	80.37%	91.78%	70.04%	79.82%	0.539	23.637

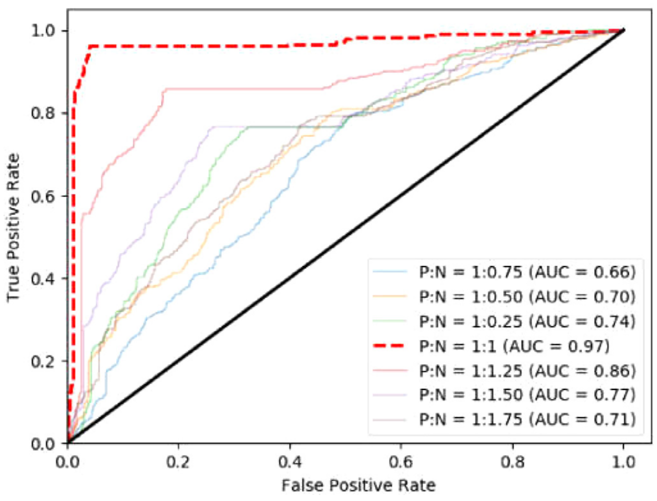


Fig. 4. ROC for various Re-sampling rates.

nority class. Whereas, we exploit synthetic minority over-sampling technique to over-sample the minority class. It finds the k-nearest neighbours for minority class and randomly chooses one of them.

Table 6 shows the average accuracy, precision, recall, F-measure, MCC and OR of the proposed approach with under-sampling and over-sampling are (*91.79%, 60.13%, 98.33%, 74.51%, 0.734*, and *40.891*) and (*80.37%, 91.78%, 70.04%, 79.82%, 0.539* and *23.637*), respectively.

From Table 6, we observe the performance difference between balanced and imbalanced datasets. It suggests that both under-sampling and over-sampling does improve the accuracy and recall of the proposed approach. Whereas, the precision of the proposed approach does not improve with under-sampling. Although, re-sampling is an effective approach to correct the bias for the imbalanced dataset to improve the performance, however, we found *26.15% = 86.28% - 60.13%* decrease in precision in our approach. The possible reason is that the negative data (rejected reports) are reduced with re-sampling, and thus the machine tends to make few negative predictions (more positive predictions). As a result, the precision is decreased.

To evaluate the performance of the proposed approach, we also plot a receiver operation characteristics (ROC) curve for the various under-sampling rates of majority class and over-sampling rates of the minority class (Fig. 4). The ROC curve shows the trade-off between sensitivity (or true positive rate) and specificity (1 - false positive rate). A classifier that gives curve closer to the top-left corner indicates better performance. The average value of the area under the curve (AUC = 0.97) against the ratio 1:1 of approved/rejected reports confirms that the proposed approach has the best measure of separability with equal distribution of input reports.

3.4.4. RQ4: Influence of Classification Algorithms

Bayesian, logistic regression, random forest and support vector machine (proposed approach) are widely used machine learning algorithms in textual classification (Sohrawardi et al., 2014) due to their competitive performance. Support vector machine is commonly observed more accurate than Bayesian and logistic regression (Khan et al., 2010). To investigate which classifier can lead to the best performance in our case, we evaluate the results of Bayesian, logistic regression, random forest and support vector machine classifiers.

The evaluation results of each classifier are given in Table 7. The average accuracy, precision, recall, F-measure, MCC and OR of support vector classifier are *77.90%, 86.28%, 66.45%, 74.53%, 0.487* and *16.189%* respectively. It also summarizes the average accuracy, precision, recall and F-measure of naïve bayes, multinomial naïve bayes, bernoulli naïve bayes, logistic regression and random forest classifiers respectively. The average accuracies of these algorithms are respectively, *70.44%, 76.20%, 70.62%, 75.52%* and *76.88%*. Alternatively, Fig. 5 is a graphical representation of the comparison between the results of average accuracy, precision, recall and F-measure of all classifier.

From Table 7 and Fig. 5, we make the following observation:

- The proposed approach not only outperforms the Bayesian, logistic regression and random forest in terms of accuracy but also improves the precision, recall, and F-measure. The proposed approach achieves the best performance for different reasons. First, support vector machine learns hyperplane in the feature space (Li et al., 2009; Khan et al., 2010) that has the maximum margin for each report (except some outliers). Consequently, support vector machine has better generalization capability on the testing data (unseen data) than the other distance-based and similarity-based algorithms (e.g., random forest) for the classification problem (Cristianini and Shawe-Taylor, 2000). Second, support vector machine can search different combinations of the given features due to the kernel function and does not increase the computational complexity as compared to other machine learning algorithms (Cristianini and

Table 7
Accuracy of the Proposed Approach.

ML Classifier	Accuracy	Precision	Recall	F-Measure	MCC	OR
Proposed Approach	**77.90%**	**86.28%**	**66.45%**	**74.53%**	**0.487**	**16.189**
LR	75.52%	85.90%	65.15%	74.08%	0.486	15.896
RF	76.88%	56.51%	48.38%	52.13%	0.334	7.079
MNB	76.20%	49.49%	54.31%	49.81%	0.322	7.019
NB	70.44%	43.23%	66.43%	50.90%	0.321	6.174
BNB	70.62%	43.32%	66.23%	50.88%	0.335	6.199

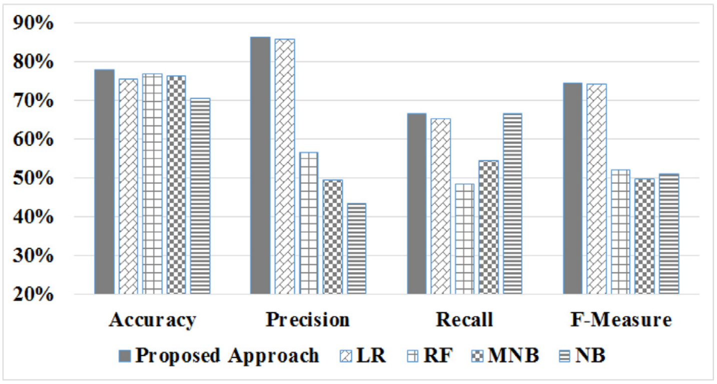


Fig. 5. Impact of classification algorithms.

Shawe-Taylor, 2000). The proposed approach represents reports by very high dimensional and very sparse feature vectors (as mentioned in Section 2.5) for approval prediction. In our natural language classification problem, such representation of reports distributes the approved and rejected reports in distinctly different areas of the feature space. This helps support vector machine for the generalization capability and to search classification hyperplane in approval prediction of reports.

- Logistic regression outperforms the Bayesian. The possible reason for the significant difference in performance is the large number of independent features of reports. Logistic regression (a discriminative model) performs better than the Bayesian (a generative model) when the number of training feature size is large (Ng and Jordan, 2002). Logistics regression also outperforms the random forest. It increases F-measure significantly from *52.13%* to *74.08%*, MCC from *0.334* to *0.486*, and OR from *7.079* to *15.896*. A possible reason is that logistic regression is faster to train and execute on the sparse features (a feature is sparse if it has mostly zero values, i.e., a word count). Whereas, random forest ignores the sparse features. Another possible reason is high dimensional features. Logistic regression performs better due to the curse of dimensionality (Ng and Jordan, 2002; Khan et al., 2010). Whereas, the degree of freedom within trees makes random forest inappropriate for high dimensional features in approval prediction of reports.
- The evaluation results of the logistic regression classifier are very close to the proposed approach. Logistic regression predicts approval of reports significantly better than Bayesian and random forest with independent and high dimensional features vector. It may outperform the support vector machine if the size of the training dataset is increased. In future, we will conduct an experiment with a larger dataset for further comparison.

3.5. Threats to validity

The first threat to the **construct validity** is the potentially inaccurate labeling of reports. The evaluation takes the assumption that all involved reports are correctly labeled. However, according to Herzig et al. (2013), labeling is in general not reliable. Therefore, to evaluate the correctness of the labels of reports, we randomly select *200* reports to manually examine their classification. To this end, we follow the rule "*a report is classified as enhancement report if it requests to implement a new access/getter method, to add new functionality, or to support new object type, specifications, or standards*" (Herzig et al., 2013) to identify the real labels. The results suggest that *7 (3.5%)* reports out of *200* are misclassified. For example; a report *Add setting for moving FST offheap/onheap* is classified as *new feature* but causes a *JVM OutOfMemory* exception. Whereas. according to the rule "*a report is classified as bug if it fixes runtime or memory issues cause by defects*" (Herzig et al., 2013), the report should be classified as *bug*.

The second threat to **construct validity** is the suitability of the evaluation metrics. The accuracy, precision, recall, and F-measure are used for the evaluation of machine learning algorithms. Because these metrics are most commonly used for the evaluation of the classification algorithms. However, to mitigate this threat, we evaluate the proposed approach with MCC and OR to measure its quality and effectiveness.

The third threat to **construct validity** is related to the usage of python package to compute the sentiment of reports. There is a number of corpora for sentiment analysis, however, we select *Senti4SD* as it is specially designed for software engineering domain. Other sentiment analysis corpora may decrease the performance of the proposed approach. For example; we also try *SentiWordNet* to compute the sentiment of the reports, but it reduces the results of the proposed approach.

A threat to **internal validity** is related to the implementation of the proposed and state-of-the-art approaches. To mitigate the threat, the implementation and the evaluation results are double checked. However, there could be some unnoticed errors.

The first threat to the **external validity** is the proposed approach may not work for those reports written languages other than English. Our classifier is trained and evaluated with English reports only. In future, it would be interesting to investigate the possibility to apply the proposed approach to reports in other languages.

The second threat to **external validity** is related to the result's generalizability. We use the reports related to Mozilla ecosystem (mentioned in Section 3.2) for the training and evaluation. Some specific characteristics of such reports may yield the evaluation results. In future, we have the intention to try reports from other software applications to overcome the said threat to validity.

4. Related work

Extensive research has been conducted on the automated classification of the reported documents (Youn and Mcleod, 2006; Gad and Rady, 2015; Santos et al., 2012; Zhang et al., 2014; Jin et al., 2015; Moraes et al., 2013; Khan et al., 2010; Lin et al., 2016; Sharma et al., 2015). However, few of them deal with the enhancement reports. Nizamani et al. (2017) recently proposed an automated classification approach for the enhancement reports to predict their approval status. To the best of our knowledge, it is the only approach to predict the approval of reports. They used the multinomial naïve Bayes classifier for the binary classification and found accuracy up to 89.25% which is considerable for the approval prediction of reports. The proposed approach differs from their approach in that the proposed approach exploits the sentiment of reports. Evaluation results suggest that the proposed approach outperforms the state-of-the-art approach (Nizamani et al., 2017) in both accuracy and recall.

Most of the state-of-the-art classification approaches focus on the categorization of the documents as follows; 1) detection of incorrect classification of documents 2) automated assignment of documents to developers, 3) detection of duplicate documents, and 4) severity prediction of documents to prioritize them.

4.1. Detection of incorrect classification

Bugs and enhancements are commonly reported in issue tracking systems for the management of their resolution. Developers manually separate bugs from non-bugs which is a challenging task for them. For instance, an enhancement may be marked as a bug. As a result, incorrect classification of a bug/enhancement may delay its resolution. Antoniol et al. proposed an approach to automate the problem of incorrect classification of bugs (Antoniol et al., 2008). They applied naïve Bayes, decision trees and logistic regression classifiers on the reports of Eclipse and JBoss and found the accurate classification of bugs up to *82%*. Later, Herzig et al. performed a manual examination of 7000 issue reports collected from different issue tracking systems and found 33.8% reports are incorrectly classified (Herzig et al., 2013). Automated approaches for the detection of incorrect classification is a relief for developer and decrease the chances of misclassification.

4.2. Automated assignment of issues

Users of the different software application often submit their requests to fix the issues, amendments or enhancements to the application. Such requests are manually assigned to the developers for their resolution, which is a complicated task. Many studies (Anvik, 2006; Bhattacharya et al., 2012; Jeong et al., 2009) have been conducted to automate the procedure of issue assignment to the developer. Anvik (2006) proposed a semi-automatic approach for assignment of issues and applied support vector and naïve Bayes classifiers to build the model. Anvik submitted an extension of his work and developed a recommender system for the automatic assignment of issues (Anvik, 2007). Naguib et al. also proposed an automatic issue assignment approach based on the profiles of developers (Naguib et al., 2013). They took reviews, assignments and resolution fields against each developer ID and generated the user's profile to the automatic assignment of issues to the appropriate developer. Canfora and Cerulo (2006) proposed an approach that recommends a new bug report to an appropriate developer, and Weiss et al. (2007) predict the bug fixing time taken by developers.

4.3. Detection of duplicate issues

There is a high possibility that a new issue may address the same problem as existing issues address. Therefore, such new request is susceptible to be duplicate to the existing ones. Automatic identification of such issues may ease the time-consuming manual issue handling. However, the automatic identification of duplicate issues is not easy. In this perspective, a number of studies have been conducted to identify the duplicate issues (Banerjee et al., 2012; Thung et al., 2014; Alipour et al., 2013). Saric et al. used supervised machine learning techniques to find the textual similarity based on semantics (Šarić et al., 2012). Later, this study was improved by Lazar et al. (2014) to identify duplicate issues by constructing feature sets based on textual similarity. Lin et al. proposed a different approach relied on support vector machine for the same purpose (Lin et al., 2016). Tian et al. trained support vector classifier that can highlight a new request as duplicate if the similar issue is found in existing issues (Tian et al., 2012). Another research was conducted by Feng et al. to detect the duplicate issues using consumer's profile (Feng et al., 2013). Bugzilla has implemented a tool proposed by Thung et al. (2014) named as DupFinder, which applied support vector model to measure similarity using summary and description attributes of the reports. Other different studies related to bug reports include software repositories (Iliev et al., 2012; Zhang et al., 2015; Williams and Hollingsworth, 2005) and detection of bug report duplication (Wang et al., 2008).

4.4. Severity prediction for issues

Severity attribute of an issue represents its nature. The value of the severity may vary from *blocker, critical, major, normal, minor, trivial and enhancement*. Severity plays an important role in the decision of developers for the selection of the issue in order to prioritize them. In this regard, a novel approach (SEVERityISsue) proposed by Menzies and Marcus (2008) based on the support vector classifier to predict the severity. Lamkanfi et al. (2010), Roy and Rossi (2014) filtered the issues of Eclipse, GNOME, and Mozilla and applied naïve Bayes classifier for severity prediction. They found naïve Bayes classifier as optimal for the severity prediction by classifying the issues into two categories: severe and non-severe. Sharma et al. (2015) applied multinomial naïve Bayes and k-nearest neighbor for classification of issues to predict their severity. They took info-gain and chi-square for feature selection. Another study (Chaturvedi and Singh, 2012) applied J48, RIPPER, naïve Bayes, support vector, multinomial naïve Bayes and k-nearest neighbor on IV and V projects of NASA for the severity prediction.

4.5. Sentiments in software engineering

Researchers have achieved excellent results for sentiment analysis in semantic parsing, search query retrieval, sentence modeling, traditional NLP tasks, and sentiment analysis (Ouyang et al., 2015). Some state-of-the-art approaches based on sentiments analysis are discussed in the following.

Jongeling et al. (2017) provided a comparison between sentiment analysis tools. They used sentiment analysis tools (SentiStrength, Alchemy, NLTK, and Stanford NLP) to observe the disagreement between tools can lead to diverging conclusions. They also replicate the two existing studies and figured out that previously published results cannot be replicated with the use of different sentiment analysis tools. They revealed the need for a sentiment analysis tool, especially for the software engineering domain.

Islam and Zibran (2018b) developed a tool for improved sentiment analysis that is specially designed for the software engineering text. They used 5600 manually annotated JIRA issue comments for the quantitative and qualitative evaluation of their tool. The evaluation results suggest that the SentiStrength-SE (domain-dependent tool) outperforms the existing domain-independent tools (SentiStrength, Natural Language Toolkit, and Stanford NLP) in detecting sentiments in software engineering text. Islam and Zibran (2018a) also presented the sentiment analysis tool *DEVA*. It not only performs sentiment analysis for software engineering text but also captures the emotional states. They used 1795 JIRA issue comments (a ground truth dataset) for the quantitative evaluation of *DEVA*. The evaluation results suggest that the precision and recall of *DEVA* is more than 82% and 78%, respectively.

Ahmed et al. (2017) developed a sentiment analysis tool for software engineering domain. They used 2000 manually labeled review comments as a dataset. They performed one hundred 10-fold cross-validations of 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) for the evaluation of their tool. Evaluation results suggest that gradient boosting tree outperforms the other algorithms and provides the highest mean accuracy, precision, and recall up to 83%, 67.8%, and 58.4% in identifying negative review comments, respectively.

Calefato et al. (2017) presented a toolkit *EmoTxt* for the identification of emotion from text. They trained and evaluated *EmoTxt* on a gold standard of about 9K question, answers, and comments. They provided empirical evidence that *EmoTxt* achieves comparable performance with datasets collected from Stack Overflow and JIRA. Calefato et al. (2018) also developed a sentiment analysis tool *Senti4SD* to support developers in their communication channels. They trained and validated their tool with a gold standard of Stack Overflow questions, answers, and comments. Notably, they manually annotated the dataset for sentiment polarity. Results suggest that the tool decreases the misclassification neutral and positive post as emotionally negative. Another tool *MEME-a Method for EMotion Extraction* to extract emotion from the software engineering text is developed by Werder and Brinkkemper (2018). They used GHtorrent and GitHub as data sources to evaluate *MEME*. The evaluation results suggest that *MEME* is a better tool in contrast to Syuzhet R package emotion analysis.

Marshall et al. (2016) examined the effect of emotional post content on project performance. They used over thirteen hundreds forum posts produced by five teams across three Scrum Sprints. They employed manual sentiment analysis techniques for the evaluation and found that emotional posts in software development team communication need intervention. Williams and Mahmoud (2017) also presented a preliminary study to detect, classify, and interpret emotions in the tweets of software users. They used 1000 tweets from a broad range of software systems' Twitter. Their results suggest that naive bayes and support vector machine are accurate to detect general and specific emotions expressed in software-relevant tweets.

Moreover, Fountaine and Sharif (2017) presented a position paper to the effects of emotional awareness on the progress of developers that implicates the effects of emotion in software development. At the same time, Graziotin et al. (2017) uncovered the experienced consequences of unhappiness (emotions) among developer while software development and found 49 consequences of unhappiness. Graziotin et al. (2018) also studied the effects of happiness and unhappiness while developing software. They did qualitative analysis and found the consequences of both happiness and unhappiness that are beneficial to the mental well-being of developers.

Lin et al. (2018) built a software library using developers' opinions collected from Stack Overflow. They used 40K manually labeled sentences from the collected dataset. They investigated the accuracy of the tools (SentiStrength, NLTK, Stanford CoreNLP, and EmoTxt) that are used to identify the sentiment of software engineering text. They reported that Stack Overflow does not deal with emotions, it's a place for developers to discuss technicalities.

Umer et al. (2018) recently proposed an emotion-based automatic approach (*eApp*) for the priority classification. They are the first to employ emotion analysis for priority classification. They used the support vector machine (a machine learning algorithm) for the multi-class prioritization of bug reports. They found performance improvement in *F1-score* by more than six percent. Results of our evaluation suggest that the proposed approach outperforms the state-of-the-art approach and improves *F1-score* by more than *24%*.

As a conclusion, researchers have proposed a number of sentiment analysis tools and sentiment related approaches for software engineering. However, to the best of our knowledge, the proposed approach differs from the existing approaches in that we are first to incorporate sentiments for the approval prediction of enhancement reports.

5. Conclusion and future work

A significant percentage of the enhancement reports of software applications are rejected. Manually checking such reports may cause the wastage of time of the developers. To this end, we proposed an automatic approach to predict whether a given enhancement report is likely-to-be-approved by combining the natural language processing techniques and machine learning algorithms. With the approach, developers can resolve more likely-to-be-approved reports first and can save their time by ignoring those reports which are predicted to be rejected. The evaluation of the proposed approach is performed on top ten open-source software application. Compared to the state-of-the-art approach, the proposed approach has shown improvement in accuracy, precision, recall and F-measure respectively, *8.66%, 67.20%, 23.81%* and *46.20%*.

The broader impact of our work is to show that the sentiment and the description of reports could be beneficial in their approval prediction. Our results encourage future research on the approval prediction of reports. It is likely to contain software-specific elements, especially source code in the reports, and such elements may require special preprocessing. Notable, to this end, we manual sample *300* reports (randomly selected) and find that only 2 of them contain software-specific elements. Therefore, we have not yet applied any special technique to remove the software-specific elements from the reports. Furthermore, we would also like to exploit deep learning algorithms to improve the performance of the proposed approach.

Acknowledgments

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

The work is supported by the National Natural Science Foundation of China (61690205, 61772071).

References

Ahmed, T., Bosu, A., Iqbal, A., Rahimi, S., 2017. Senticr: a customized sentiment analysis tool for code review interactions. In: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 106–111.

Alipour, A., Hindle, A., Stroulia, E., 2013. A contextual approach towards more accurate duplicate bug report detection. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 183–192.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.-G., 2008. Is it a bug or an enhancement?: a text-based approach to classify change requests. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. ACM, New York, NY, USA, pp. 23:304–23:318. doi:10.1145/1463788.1463819.

Anvik, J., 2006. Automating bug report assignment. In: Proceedings of the 28th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 937–940. doi:10.1145/1134285.1134457.

Anvik, J., 2007. Assisting bug report triage through recommendation. University of British Columbia Ph.D. thesis.

Baccianella, S., Esuli, A., Sebastiani, F., 2010. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In: in Proc. of LREC.

Banerjee, S., Cukic, B., Adjeroh, D., 2012. Automated duplicate bug report classification using subsequence matching. In: Proceedings of the 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering. IEEE Computer Society, Washington, DC, USA, pp. 74–81. doi:10.1109/HASE.2012.38.

Bhattacharya, P., Neamtiu, I., Shelton, C.R., 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. J. Syst. Softw. 85 (10), 2275–2292. doi:10.1016/j.jss.2012.04.053. Automated Software Evolution.

Bird, S., 2002. Nltk: the natural language toolkit. In: In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics.

Calefato, F., Lanubile, F., Maiorano, F., Novielli, N., 2018. Sentiment polarity detection for software development. Empirical Softw. Eng. 23 (3), 1352–1382. doi:10.1007/s10664-017-9546-9.

Calefato, F., Lanubile, F., Novielli, N., 2017. Emotxt: a toolkit for emotion recognition from text. In: 2017 Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW), pp. 79–80.

Canfora, G., Cerulo, L., 2006. Supporting change request assignment in open source development. In: Proceedings of the 2006 ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1767–1772. doi:10.1145/1141277.1141693.

Chakrabarti, S., Roy, S., Soundalgekar, M.V., 2002. Fast and accurate text classification via multiple linear discriminant projections. In: Proceedings of the 28th International Conference on Very Large Data Bases. VLDB Endowment, pp. 658–669.

Chaturvedi, K.K., Singh, V.B., 2012. Determining bug severity using machine learning techniques. In: 2012 CSI Sixth International Conference on Software Engineering (CONSEG), pp. 1–6. doi:10.1109/CONSEG.2012.6349519.

Cristianini, N., Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press doi:10.1017/CBO9780511801389.

Feng, L., Song, L., Sha, C., Gong, X., 2013. Practical duplicate bug reports detection in a large web-based development community. In: Web Technologies and Applications. Springer, pp. 709–720.

Fountaine, A., Sharif, B., 2017. Emotional awareness in software development: theory and measurement. In: Proceedings of the 2Nd International Workshop on Emotion Awareness in Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 28–31. doi:10.1109/SEmotion.2017...12.

Gad, W., Rady, S., 2015. Email filtering based on supervised learning and mutual information feature selection. In: 2015 Tenth International Conference on Computer Engineering Systems (ICCES), pp. 147–152. doi:10.1109/ICCES.2015.7393036.

Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P., 2017. Consequences of unhappiness while developing software. In: Proceedings of the 2Nd International Workshop on Emotion Awareness in Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 42–47. doi:10.1109/SEmotion.2017...5.

Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P., 2018. What happens when software developers are (un)happy. J. Syst. Softw. 140, 32–47. doi:10.1016/j.jss.2018.02.041.

Herzig, K., Just, S., Zeller, A., 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 392–401.

Iliev, M., Karasneh, B., Chaudron, M.R.V., Essenius, E., 2012. Automated prediction of defect severity based on codifying design knowledge using ontologies. In: Proceedings of the First International Workshop on Realizing AI Synergies in Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 7–11.

Islam, M.R., Zibran, M.F., 2018. Deva: Sensing emotions in the valence arousal space in software engineering text. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1536–1543. doi:10.1145/3167132.3167296.

Islam, M.R., Zibran, M.F., 2018. Sentistrength-se: exploiting domain specificity for improved sentiment analysis in software engineering text. J. Syst. Softw. 145, 125–146. doi:10.1016/j.jss.2018.08.030.

Jeong, G., Kim, S., Zimmermann, T., 2009. Improving bug triage with bug tossing graphs. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. ACM, New York, NY, USA, pp. 111–120. doi:10.1145/1595696.1595715.

Jin, Z., Li, Q., Zeng, D., Wang, L., 2015. Filtering spam in weibo using ensemble imbalanced classification and knowledge expansion. In: 2015 IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 132–134. doi:10.1109/ISI.2015.7165952.

Joachims, T., 1998. Text categorization with support vector machines: learning with many relevant features. In: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23. Springer Berlin Heidelberg, pp. 137–142.

Jongeling, R., Sarkar, P., Datta, S., Serebrenik, A., 2017. On negative results when using sentiment analysis tools for software engineering research. Empirical Softw. Eng. 22 (5), 2543–2584. doi:10.1007/s10664-016-9493-x.

Khan, A., Baharudin, B., Lee, L.H., Khan, K., Tronoh, U.T.P., 2010. A review of machine learning algorithms for text-documents classification. Journal of Advances In Information Technology, VOL.

Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B., 2010. Predicting the severity of a reported bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 1–10. doi:10.1109/MSR.2010.5463284.

Lazar, A., Ritchey, S., Sharif, B., 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, New York, NY, USA, pp. 308–311. doi:10.1145/2597073.2597088.

Li, Y., Bontcheva, K., Cunningham, H., 2009. Adapting svm for data sparseness and imbalance: a case study in information extraction. Nat. Lang. Eng. 15 (2), 241–271. doi:10.1017/S1351324908004968.

Lin, B., Zampetti, F., Bavota, G., Di Penta, M., Lanza, M., Oliveto, R., 2018. Sentiment analysis for software engineering: how far can we go? In: Proceedings of the 40th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 94–104. doi:10.1145/3180155.3180195.

Lin, M.-J., Yang, C.-Z., Lee, C.-Y., Chen, C.-C., 2016. Enhancements for duplication detection in bug reports with manifold correlation features. J. Syst. Softw. 121 (Supplement C), 223–233. doi:10.1016/j.jss.2016.02.022.

Lin, Y., 2002. Support vector machines and the bayes rule in classification. Data Min. Knowl. Discov. 6 (3), 259–275. doi:10.1023/A:1015469627679.

Loper, E., Bird, S., 2002. Nltk: The natural language toolkit. In: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 63–70. doi:10.3115/1118108.1118117.

Marshall, A., Gamble, R.F., Hale, M.L., 2016. Outcomes of emotional content from agile team forum posts. In: 2016 IEEE/ACM 1st International Workshop on Emotional Awareness in Software Engineering (SEmotion), pp. 6–11. doi:10.1109/SEmotion.2016.011.

Menzies, T., Marcus, A., 2008. Automated severity assessment of software defect reports. In: 2008 IEEE International Conference on Software Maintenance, pp. 346–355. doi:10.1109/ICSM.2008.4658083.

Moraes, R., Valiati, J.F., Neto, W.P.G., 2013. Document-level sentiment classification: an empirical comparison between svm and ann. Expert Syst. Appl. 40 (2), 621–633. doi:10.1016/j.eswa.2012.07.059.

Naguib, H., Narayan, N., Bruegge, B., Helal, D., 2013. Bug report assignee recommendation using activity profiles. In: IEEE International Working Conference on Mining Software Repositories, pp. 22–30.

Ng, A.Y., Jordan, M.I., 2002. On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (Eds.), Advances in Neural Information Processing Systems 14. MIT Press, pp. 841–848.

Nizamani, Z.A., Liu, H., Chen, D.M., Niu, Z., 2017. Automatic approval prediction for software enhancement requests. Autom. Softw. Eng. doi:10.1007/s10515-017-0229-y.

Ouyang, X., Zhou, P., Li, C.H., Liu, L., 2015. Sentiment analysis using convolutional neural network. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 2359–2364. doi:10.1109/CIT/IUCC/DASC/PICOM.2015.349.

Roy, N.K.S., Rossi, B., 2014. Towards an improvement of bug severity classification. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 269–276. doi:10.1109/SEAA.2014.51.

Santos, I., Laorden, C., Sanz, B., Bringas, P.G., 2012. Enhanced topic-based vector space model for semantics-aware spam filtering. Expert Syst. Appl. 39 (1), 437–444. doi:10.1016/j.eswa.2011.07.034.

Schölkopf, B., Burges, C.J.C., Smola, A.J. (Eds.), 1999. Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, USA.

Sharma, G., Sharma, S., Gujral, S., 2015. A novel way of assessing software bug severity using dictionary of critical terms. Procedia Comput. Sci. 70 (Supplement C), 632–639. doi:10.1016/j.procs.2015.10.059. Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems

Sohrawardi, S.J., Azam, I., Hosain, S., 2014. A comparative study of text classification algorithms on user submitted bug reports. In: Ninth International Conference on Digital Information Management (ICDIM 2014), pp. 242–247. doi:10.1109/ICDIM.2014.6991434.

Tan, S., Wu, G., Cheng, X., 2009. Enhancing the Performance of Centroid Classifier by ECOC and Model Refinement. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 458–472. doi:10.1007/978-3-642-04174-7_30.

Thung, F., Kochhar, P.S., Lo, D., 2014. Dupfinder: integrated tool support for duplicate bug report detection. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ACM, New York, NY, USA, pp. 871–874. doi:10.1145/2642937.2648627.

Tian, Y., Sun, C., Lo, D., 2012. Improved duplicate bug report identification. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 385–390. doi:10.1109/CSMR.2012.48.

Umer, Q., Liu, H., Sultan, Y., 2018. Emotion based automated priority prediction for bug reports. IEEE Access 6, 35743–35752. doi:10.1109/ACCESS.2018.2850910.

Vapnik, V. N., 1999. The nature of statistical learning theory.

Šarić, F., Glavaš, G., Karan, M., Šnajder, J., Bašić, B.D., 2012. Takelab: systems for measuring semantic text similarity. In: Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 441–448.

Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J., 2008. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 461–470. doi:10.1145/1368088.1368151.

Weiss, C., Premraj, R., Zimmermann, T., Zeller, A., 2007. How long will it take to fix this bug? In: Proceedings of the Fourth International Workshop on Mining Software Repositories. IEEE Computer Society, Washington, DC, USA, pp. 1–8. doi:10.1109/MSR.2007.13.

Werder, K., Brinkkemper, S., 2018. Meme-toward a method for emotions extraction from github doi:10.1145/3194932.3194941.

Williams, C.C., Hollingsworth, J.K., 2005. Automatic mining of source code repositories to improve bug finding techniques. IEEE Trans. Softw. Eng. 31, 466–480.

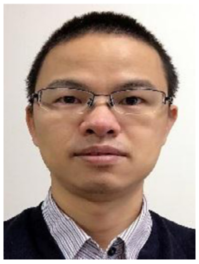
Williams, G., Mahmoud, A., 2017. Analyzing, classifying, and interpreting emotions in software users' tweets. In: Proceedings of the 2Nd International Workshop on Emotion Awareness in Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 2–7. doi:10.1109/SEmotion.2017...1.

Youn, S., Mcleod, D., 2006. A comparative study for email classification. Seongwook Los Angeles", CA 90089.

Zhang, T., Yang, G., Lee, B., Chan, A.T.S., 2015. Predicting severity of bug report by mining bug repository with concept profile. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1553–1558. doi:10.1145/2695664.2695872.

Zhang, Y., Wang, S., Phillips, P., Ji, G., 2014. Binary {PSO} with mutation operator for feature selection using decision tree applied to spam detection. Knowl. Based Syst. 64, 22–31. doi:10.1016/j.knosys.2014.03.015.

QASIM UMER received the BS degree in computer science from Punjab University, Pakistan in 2006, MS degree in.net distributed system development from University of Hull, UK in 2009, and MS degree in computer science from University of Hull, UK in 2012. He is currently perusing PhD degree in computer science from Beijing Institute of Technology, China. He is particularly interested in machine learning, data mining and software maintenance.



HUI LIU is a professor at the School of Computer Science and Technology, Beijing Institute of Technology, China. He received BS degree in control science from Shandong University in 2001, MS degree in computer science from Shanghai University in 2004, and PhD degree in computer science from the Peking University in 2008. He was a visiting research fellow in centre for research on evolution, search and testing (CREST) at University College London, UK. He served on the program committees and organizing committees of prestigious conferences, such as ICSME, RE, ICSR, and COMPSAC. He is particularly interested in software refactoring, AI-based software engineering, and software quality. He is also interested in developing practical tools to assist software engineers.

YASIR SULTAN is a Software Development Engineer at Foxconn Technology Group, China. He received the BS degree in information technology from Government College University in 2016, MS degree in computer science from Beijing Institute of Technology, China. He is particularly interested in software engineering, data mining and machine learning.