


Automatic approval prediction for software enhancement requests

Zeeshan Ahmed Nizamani¹  · Hui Liu¹ ·
David Matthew Chen¹ · Zhendong Niu¹

Received: 17 October 2016 / Accepted: 30 September 2017 / Published online: 26 October 2017
© Springer Science+Business Media, LLC 2017

Abstract Software applications often receive a large number of enhancement requests that suggest developers to fulfill additional functions. Such requests are usually checked manually by the developers, which is time consuming and tedious. Consequently, an approach that can automatically predict whether a new enhancement report will be approved is beneficial for both the developers and enhancement suggesters. With the approach, according to their available time, the developers can rank the reports and thus limit the number of reports to evaluate from large collection of low quality enhancement requests that are unlikely to be approved. The approach can help developers respond to the useful requests more quickly. To this end, we propose a multinomial naive Bayes based approach to automatically predict whether a new enhancement report is likely to be approved or rejected. We acquire the enhancement reports of open-source software applications from Bugzilla for evaluation. Each report is preprocessed and modeled as a vector. Using these vectors with their corresponding approval status, we train a Bayes based classifier. The trained classifier predicts approval or rejection of the new enhancement reports. We apply different machine learning and neural network algorithms, and it turns out that the multinomial naive Bayes classifier yields the highest accuracy with the given dataset. The proposed

✉ Hui Liu
liuhui08@bit.edu.cn

Zeeshan Ahmed Nizamani
nizamanishan@gmail.com; zeeshan_nizamani5@hotmail.com

David Matthew Chen
chen7david@me.com

Zhendong Niu
zniu@bit.edu.cn

¹ School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

approach is evaluated with 40,000 enhancement reports from 35 open source applications. The results of tenfold cross validation suggest that the average accuracy is up to 89.25%.

Keywords Software enhancements · Machine learning · Multinomial naive Bayes · Document classification

1 Introduction

New feature enhancements to software applications are inevitable. Software has to fulfill user needs, and these requirements change over time. Hence new feature enhancements become necessary for success of the software due to evolving user requirements and changing technologies (Rajlich 2014). Development of newer versions of a an application also requires new or improved feature enhancements. A non-trivial software therefore receives a number of suggestions about adding new feature enhancements and improving the existing ones. To request an enhancement in a software application, the suggester writes a report to describe the enhancement. In issue tracking systems, feature enhancement requests are taken as a special kind of issue reports. Consequently, enhancement requests are often called *enhancement reports* as well.

The empirical study of the issue reports statistics shows that the enhancement reports form a sizable portion of the total issue reports filed for an application. We observed the statistics for Thunderbird product between 2000-01-02 and 2005-12-24, in which out of 10,000 bug reports, 1857 reports were enhancements, which account for 18.57% of all types of issue reports. This statistics suggests that enhancement requests represent a meaningful number of all issue reports and a non-trivial software application is subject to a number of enhancements.

Not all of the enhancement reports will finally be approved. By analyzing the enhancement reports from the 35 applications we find that around 75% of the reports are not approved. The problem of low quality reports which leads to rejection of these reports, often arises because the enhancement reporters are not fully aware of the history, environment, functionality and technological limitations associated with the software. This leads to the mismatch between what the developers expect and what the reporters provide (Zimmermann et al. 2010). Consequently, the reporters end up proposing poorly written enhancement reports and the software maintainer ends up evaluating many enhancement reports that are not fixable thus wasting his time and efforts.

Although, the developers have to ultimately evaluate the reports manually before assigning and implementing the enhancements, there are benefits of having an automated approach:

1. First, it can help developers respond to useful enhancement reports quickly. Sizable software applications often receive a large number of enhancement reports, among which only a small part will be adopted. It may take a long time for the developers to read and respond to such a large number of reports. As a result, some important and useful enhancement reports cannot be handled in time. However, if an automated

approach can rank the reports and help developers to pick up a small number of reports that are most likely to be approved, such more valuable reports could be handled promptly.

2. Second, it can help reporters to improve their enhancement reports before submission. By using the automated system, the suggesters can get an idea of whether the recommended enhancement is likely to be approved or not and hence be able to rework the request before submission. This would in turn translate into less but better quality enhancement reports.

To this end, we propose a supervised machine learning based approach to automatically predict the approval of software enhancement requests. The acquired enhancement reports are preprocessed to lower-case the reports' description text, remove non-dictionary words, and lemmatize the words. The lemmatized words are used as features to model the description text of reports as feature vectors. A portion from the set of feature vectors corresponding to the enhancement reports text are used to train a supervised learning classifier. The trained classifier is tested on a different portion of reports from the feature vectors set to evaluate the performance of the approach.

Machine learning based approaches have successfully been applied in classifying different kinds of software documents. Approaches on the software issue reports have been used in software bugs classification (Roy and Rossi 2014b; Herzig et al. 2013; Gopalan and Krishna 2014). These approaches have solved the problem of predicting issue reports as bug or non-bug (Sohrawardi et al. 2014) and duplicate bug report classification (Banerjee et al. 2012; Sun et al. 2011; Lin et al. 2016) using the machine learning algorithms. Popular classification models used in bug handling include decision tree, support vector machine, naive Bayes and neural networks (Lamkanfi et al. 2010). Lamkanfi et al. (2011) proposed an approach to help developers distinguish severe bugs from non-severe and to resolve them first. The authors compared the performance of different machine learning algorithms in severity prediction and conclude that naive Bayes multinomial outperforms the other algorithms on their dataset. The approach scales down multiple levels of severity into two classes; severe and non-severe, classifying new bug reports into one of these two categories. Based on these applications and effective performance of the Bayes based and support vector machine models, we include these classifiers in evaluation of our approach.

Wang et al. (2008) applied natural language processing techniques to suggest a list of most similar existing reports to the new report. The technique for duplicate issue detection using the similarity measures of word frequency has shown effective performance (Lazar et al. 2014). These solutions save the developer time and improve efficiency. The approaches to solve such problems have shown high accuracy in predicting bugs severity, bug or non-bug classification and duplicate bug report identification. Duplicate enhancement requests for an application account for a noticeable portion in the total reports set, but since there have already been a number of studies on duplicate detection, we do not specifically deal with this problem in our study.

A number of important studies and approaches to triage and automate tasks specifically for non-enhancement type bug reports have been conducted and achieved significant performance. The existing applications of machine learning algorithms

on software issue reports that use textual features (Sohrawardi et al. 2014) support the practicality of these algorithms to predict the approval of software enhancement reports. However, to the best of our knowledge, there is no existing approach specifically targeted to predict the approval of the enhancement reports.

The proposed approach is evaluated with 40,000 enhancement reports from 35 open-source software applications. The results of tenfold cross validation suggest that the approach is accurate. The Bayes based approach averaged precision, recall, and f1-score of 84.99, 63.26 and 72.53% respectively. We further compare the approach with re-sampling of the dataset since a large proportion of reports are rejected which makes the dataset imbalanced. The results of re-sampling suggests that the overall performance does not improve by under-sampling the dataset.

The major contributions of this paper include the following:

- An automatic approach to predict the approval of new enhancement reports.
- Evaluation of the approach with data from open-source applications. Evaluation results suggest that the approach is accurate.

Rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the proposed approach. Section 4 details the experimental setup, evaluation and results. Threats to validity and the limitations are discussed in Sect. 5. Finally, Sect. 6 concludes the paper and discusses future work in this direction.

2 Related work

2.1 Machine learning based bugs classification

For text based document classification problem where manually categorized history data is available, a range of supervised machine learning classification models can be used (Murphy and Cubranic 2004; Pingclasai et al. 2013). These classifiers categorize the text documents into predefined classes by building a classification model from history data. The naive Bayes classifier is one of the best text classification algorithms (Hu et al. 2014; Wu et al. 2008), which is a probabilistic learning model. The classifier assumes all the features are fully independent in a given class (Wang et al. 2014). The classifier simplifies learning with this assumption and often produces results comparable to the sophisticated classifiers. Support vector machine is one of the most popular text classification models (Hu et al. 2014). The classifier is a predictive model for the classification problems. Support vector machine classifier categorizes the input data into two classes by determining a hyper-plane that maximizes the separation between the classes (Schölkopf and Burges 1999).

Such accurate and efficient machine learning classifiers make it possible to classify the enhancement reports. These effective classification algorithms provide the basis for our approach. A binary text classifier assigns either of the two classes to each text document. Some of the applications of binary classifiers include the Spam content filtering for emails (Gad and Rady 2015; Santos et al. 2012; Zhang et al. 2014), SMS (Delany et al. 2012) and social media (Jin et al. 2015).

2.2 Automated processing of software issue reports

Machine learning based approaches have successfully been applied in automatically processing different kinds of issue reports. Some of these approaches include issue severity assessment, duplicate issue identification, bug and non-bug detection, automatic identification of files containing bugs and ranking relevant developers to fix a bug. Some of these tasks are influenced by the background and experience of the developers and automated tools can help save the time and efforts.

2.2.1 Severity assessment

Severity of an issue report is important decision factor in software development and maintenance. It helps proper resource allocation and planning for bug fixing and testing. To automatically assess severity of software defect reports, (Menzies et al. 2008) proposed SEVERIS, a tool that uses text mining and machine learning techniques to predict severity of defect reports. The system was case studied with data from NASA's Project and Issue Tracking System (PITS). SEVERIS employs text mining techniques to extract the features of bug reports, while supervised machine learning is trained on the classifications of existing reports and then assign appropriate severity levels to new reports. Lamkanfi et al. (2011) and Roy and Rossi (2014a) applied the machine learning techniques to predict the severity of issue reports on three open-source projects. The goal was to classify the severe and non-severe bugs. They applied naive Bayes based algorithms to predict the bug report severity. According to their study, the naive Bayes produced optimal results and is therefore a suitable binary classifier to classify the bug reports.

2.2.2 Issue classification and bug localization

The problem of bug report misclassification was identified by Antoniol et al. (2008) to distinguish two types of bug reports. The authors built three classifiers using decision trees, naive Bayes and logistic regression to distinguish bugs from non-bugs on Mozilla, Eclipse and Jboss projects, with a precision ranging from 77 to 82%. Ping-clasai et al. (2013) proposed the topic modeling approach to classify the bug reports using three classification methods of decision tree, naive Bayes classifier and logistic regression to get the most accurate model. The results suggested that the naive Bayes classifier was the most accurate and efficient classification model.

The problem of automatically locating the source code files that need to be changed in order to fix the bugs has been addressed by Zhou et al. (2012). They propose BugLocator, an information retrieval based method to rank the files using the textual similarity between the initial bug report and the source code that uses the information about similar bugs previously fixed. The approach uses revised Vector Space Model that uses document length and similar bugs solved before as new features. Xuan and Ming (Xuan et al. 2017) further studied the problem of automatically locating potential buggy source files, proposing a LS-CNN model that enhances the unified features by exploiting the sequential nature of source code. Based on a bug report, the proposed

model combines CNN and LSTM to extract semantic features to automatically identify potential buggy source code.

2.2.3 Automatic bug assignment

The bug assignment is the problem in which a reported bug requiring resolution, needs to be assigned to a developer to resolve the bug. Automatic approaches for assignment or tossing of the bugs reports to developers have been extensively studied (Anvik 2006; Bhattacharya et al. 2012; Jeong et al. 2009). Anvik et al. (2006) proposed a semi-automated machine learning based algorithm for the assignment of reports to relevant developers. The algorithm learns the kinds of bugs each developer resolves from the bug repository and builds a classification model. For a new bug report, the classifier short lists a small number of relevant developers to resolve the bug, achieving a precision levels of 57 and 64% on Eclipse and Firefox respectively. A user activity profile based bug report assignment technique was proposed by Naguib et al. (2013) to assign a bug to appropriate developer. In their formulation, user profile for every developer is generated based on his activities including reviews, assignment and resolutions that suggest the developer's expertise and involvement in the project.

2.2.4 Duplicate issue detection

Bug reporting is often prone to duplication. For a new bug report filed, there are chances for another similar bug report already present in the system, describing the same problem. Such similar bug reports are classified duplicates of each other. Duplicate bugs consume valuable time of developers while being difficult to pinpoint when the subject application is sufficiently large with a huge number of issue reports. Manually going through the pile of existing reports to detect duplication is tedious. In this study, we do not specifically handle this problem as

- (1) Duplicate issue detection usually requires measuring similarity of an issue report with the collection of already existing reports and ranking most similar issues. Since this approach is different from our machine learning based approach, we only address whether a new report would be approved or rejected.
- (2) The problem to detect duplicate issue and rank similar issue reports has been covered by a number of approaches (Banerjee et al. 2012; Thung et al. 2014; Hindle et al. 2016). These approaches have shown better performance and hence can be leveraged in the domain of enhancement reports.

TakeLab system (Saric et al. 2012) automates semantic similarity measure of short text documents using supervised machine learning. Using the TakeLab system, Lazar et al. (2014) presented an improved method to detect duplicate bug reports that uses textual similarity features.

Sun et al. (2011) proposed FactorLCS technique that takes into account the sequential order of the words to detect duplicate issue reports. Enhanced support vector machine model approach using the manifold textual and semantic correlation features is proposed by Lin et al. (2016) for duplicate bug detection. The approach achieves improvements between 2.79 and 28.97% in evaluation. Tian et al. (2012) measured

the text similarity between new bug reports and multiple existing reports to predict whether the new bug report is a duplicate bug. This approach trains a support vector machine classifier with repeated reports to learn the similarities. Feng et al. (2013) proposed profile information of the bug reporter to improve accuracy of the existing approaches in detecting duplicate bugs.

DupFinder (Thung et al. 2014) is an integrated duplicate bug report detection tool, implemented as a Bugzilla extension. The tool uses texts from summary and description fields of a new bug report and recent bug reports present in a bug tracking system, employees vector space model to scale the bugs similarity and lists duplicate bug reports based on the similarity of these reports with the new bug report.

Such machine learning based approaches suggest that it is viable to apply the supervised machine learning classifiers to the problem of predicting approval or rejection of the software enhancement reports. The existing approaches of the software documents classification have shown high performance. However these approaches are not designed to handle the approval prediction of software enhancement reports.

3 Approach

3.1 Overview

The proposed approach in this paper predicts whether a new enhancement report will be approved or rejected. We define the problem of the enhancement report approval prediction as the machine learning based binary classification problem that classifies the new enhancement reports into two classes: *approved* and *rejected*. The classification function f to predict a new enhancement report r into a classification category c , is given by

$$c = f(r); \quad c \in \{approve, reject\}, \quad r \in R \quad (1)$$

where c is the outcome class which can be either approve or reject, f is the classification function that predicts the approval of the report and r is the new enhancement report input to the classifier. The classification function f is obtained by training a supervised learning classifier.

Typically there are more than two types of resolutions for enhancement reports on an issue tracking system. However, only the Fixed resolution issues are approved for implementation. Rest of the reports with other resolution types are not approved due to reasons like duplicate or invalid enhancement. Thus we classify Fixed type reports as approved while the rest as rejected.

Overview of the approach is presented in Fig. 1. The proposed approach to predict the approval of the enhancement reports has two main phases. In the first phase, the enhancement reports are acquired, preprocessed and modeled. Such enhancement reports are extracted from an issue tracking system. The issue tracking systems usually provide interfaces to access the reports from their repositories. The enhancements reports in the corpus are preprocessed with case conversion, non-dictionary words removal and lemmatization. Each report is then converted into a feature vector form.

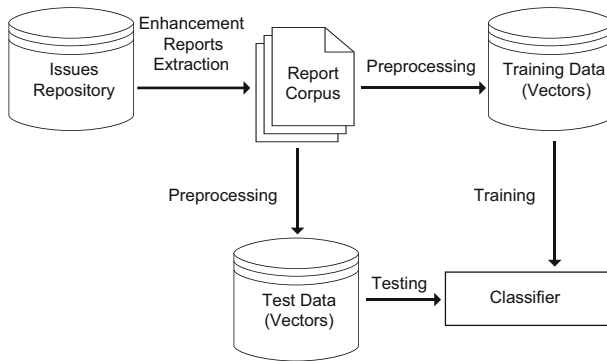


Fig. 1 Supervised prediction approach

In the second phase, a classifier is trained to classify new enhancement reports. The vectors of enhancement reports and their corresponding labels are used to train the classifier. The resulting classifier is then applied to vectors of new enhancement reports to predict the approval or rejection of each test report.

3.2 Data collection and preprocessing

The bug reports data is usually stored and managed on issue tracking systems. An issue tracking system keeps track of software application issues, allowing the users to submit issue reports for software, and also lets developers collaborate on those issue reports by making comments. There are quite a few standard issue tracking systems out there. Some of the most famous and widely used of them include Redmine, Jira and Bugzilla. Jira is a commercial license based application so we did not consider using the system for our evaluation. We chose Bugzilla in our evaluation since it is one of the most popular issue tracking systems for open-source applications.

3.2.1 Raw enhancement reports

The enhancement reports are acquired from Bugzilla. An enhancement report submitted to issue tracking system has many attributes associated with it. Figure 2 shows a graphical view of an enhancement report of an application submitted to Bugzilla. Some of the attributes of an enhancement report include identification number, title, the product for which the report is submitted, time stamps when it is reported and modified, reporter, developer assigned to resolve the issue, resolution, description of the report, and additional comments. The resolution status of the report indicates whether the enhancement has been approved or rejected. The first comment contains the description of the enhancement requested.

In our approach, the text features are selected from the enhancement reports' description for the approval prediction. The lexical description defines what feature enhancement is requested and thus is an essential parameter for approval decision. Thus our approach is based on the lexical description of enhancement request. Approaches on different bug classification tasks using textual features as input have shown reliable



Fig. 2 Sample enhancement report

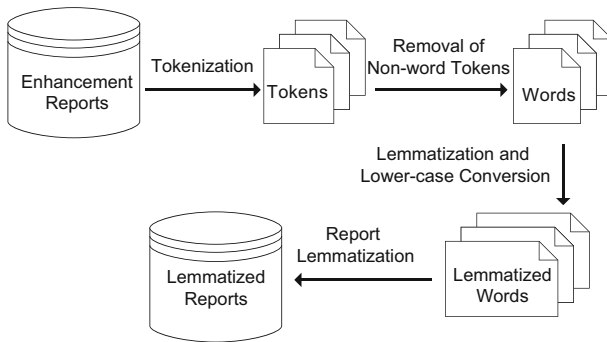


Fig. 3 Enhancement reports preprocessing

performance, suggesting that these text features are effective in the issue reports classification (Lamkanfi et al. 2011; Roy and Rossi 2014a; Xuan et al. 2017). Other factors, e.g., available resources, budget, and business concerns are also essential parameters of approval decision. However, it is challenging to retrieve and quantify such factors from a large number of applications, and thus it is hard to figure out the quantitative relationship between the approval decision and such factors.

3.2.2 Preprocessing

Preprocessing is an important step for text classification since it improves performance of the classification approach (AK Uysal 2014). The steps of preprocessing are summarized in Fig. 3.

A report is first tokenized into individual words. The report text is split on the white spaces and punctuations to generate word tokens. The non-dictionary words are removed from the word tokens obtained from tokenization. The resulting words are converted into lower case. The words are then lemmatized to convert each word to its dictionary base form. The letters in the words are normalized to lower-case form so that the classifier treats as the same, a capitalized word and its non-capitalized form. For instance, the word 'Edited' is changed to 'edit' which is both the word's base (lemmatized) and lowercase form. The lemmatized words from all the reports arranged in alphabetical order, are selected as the feature set.

An enhancement report is represented by a set of features to be processed by the classifier. A feature selection scheme is important for the performance of a classifier, that specifies criteria for measuring how informative each word is, to extract the important features. Some machine learning based classifiers including naive Bayes are sensitive to feature selection (Wei and Feng 2011; Chen et al. 2011).

For preprocessing and feature selection, we used the *twinword-lemmatizer*, a lemmatization API¹ to get the features from the reports' text. A report is lemmatized and returned by the API in JSON format containing lemmatized word features used in the report. The API performs the preprocessing steps of words tokenization, non-dictionary words removal, lower-case conversion and lemmatization. The punctuations and the numbers used in the original reports are eliminated in the lemmatization process. The lemmatized reports are saved and subsequently used for the feature vector modeling of reports. The words from all the lemmatized reports are saved in the features vocabulary. The vocabulary contains each unique word occurring in the reports dataset.

Text classification often involves high dimensional and sparse data features (Su et al. 2011). Lemmatization reduces the number of features and thus the size of feature vectors. Reduction of the feature vectors size improves the efficiency of the approach. Using lemma form of words reduces the chances of a word occurring in the training set in one form, but occurring in test report in another form, thus being treated as two different features.

3.3 Vector space model

The textual reports are converted into feature vector space model (Yang et al. 2012), which is a compact representation consisting of all the unique features extracted from the enhancement reports. Each report in the lemmatized reports corpus is converted into a vector according to the feature vector model. The mapping process is applied to both the training and test reports in the corpus.

A feature vector v is defined as,

$$v = \{c, f_1, f_2, \dots, f_n\} \quad (2)$$

¹ <https://twinword-lemmatizer1.p.mashape.com/extract/>, verified 03/03/2016.

where v is the feature vector representing an enhancement report d , c is the class label that takes either 1 or 2 value identifying whether the report is approved or rejected respectively, and f_1, f_2, \dots, f_n are the set of features each corresponding to the words in the features vocabulary. We use term frequency (TF) to represent features in a feature vector. If a feature is present in the report, it is represented by the number of times (N) it appears in the report, otherwise it is represented by 0 according to the following condition

$$f_i = \begin{cases} 0, & \text{if } i\text{th feature is absent} \\ N, & \text{if } i\text{th feature appears; } N > 0 \end{cases} \quad (3)$$

The feature number i is the feature's respective index number in the lemmatized unique words list created from the lemmatized reports corpus. The input vectors to the classifier are report feature matrix where the columns are the features and rows are the report vectors (Chen and Lü 2006). We model all the reports according to the Eq. 2.

3.4 Multinomial naive Bayes classifier

The classification method has a significant impact on the accuracy of the text classification approach (Tan et al. 2011). Given a set of reports (d_1, \dots, d_n), each report represented by the features vector v and belonging to a known class c , the aim is to construct a classification model f that allows to assign new unlabeled enhancement report to a class c (Zhang et al. 2015).

The multinomial model is considered better and efficient classification model than the multi-variate Bernoulli model (Wang et al. 2014). Multinomial naive Bayes keeps track of the frequency of words in the feature vectors representing the reports (Jiang et al. 2013; Eberhardt 2015). For a test report d , represented by feature vector $\langle w_1, w_2, \dots, w_n \rangle$, multinomial naive Bayes uses the equation below to classify the report.

$$c_{MNB}(d) = P(c) \prod_{i=1}^n P(w_i|c)^{f_i} \quad (4)$$

where $P(c)$ is the prior probability that the report d occurs in the class c , n is the number of features, w_i is the i th word occurring in the report d , $P(w_i|c)$ is the conditional probability that the word w_i occurs in the class c , f_i is the frequency count of word w_i in the report d , C is the set of all possible class labels c and $c_{MNB}(d)$ is the class label of the report d predicted by multinomial naive Bayes.

$P(c)$ of a class c is the prior probability of the class. If N_{doc} is the total number of training reports belonging to class c and T_{doc} is the total number of reports in the corpus, then $P(c)$ is calculated as,

$$P(c) = \frac{N_{doc}}{T_{doc}} \quad (5)$$

Each report belonging to a category c is grouped into respective class. The multinomial Bayes classifier computes the frequency of the word w_i in all the reports of each class to get the maximum likelihood estimate of the probability for a class as,

$$P(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \quad (6)$$

Murphy and Cubranic (2004) used the set of text words that form the bug report's summary and description fields. Authors used term frequency of words in the text to model them into vector form and applied multinomial naive Bayes classifier to automatically assign bug reports to relevant developers. Their framework treats the dataset as a collection of documents D and each document has a class label c from a set of predefined classes. Although the naive Bayes feature independence assumption does not apply in many real-world situations, yet empirical results suggest the classifier performs optimally considering the assumption not entirely unreasonable (Domingos and Pazzani 1997).

Despite the simplified assumption of features independence, the naive Bayes classifiers are shown to have sound theoretical reasons for their competitive performance (Zhang 2004). Decoupling of the class conditional feature distributions makes it possible for each of the features to be estimated independently as a one-dimensional distribution. Since our dataset has spacial feature space, this property of Bayes based classifier handles the problems of sparseness and high dimensionality, such as the need for datasets that scale directly in proportion with the number of features. Furthermore, the classification accuracy of the naive Bayes classifier is not directly affected by the degree of feature dependencies measured in terms of the class-conditional mutual information between features (Rish 2001). Such characteristics of naive Bayes model apply to our dataset and the binary classification problem, making it optimal in our approach.

4 Evaluation

4.1 Overview

The enhancement reports acquired from Bugzilla are preprocessed and converted to feature vectors. We applied different machine learning classifiers to train on our dataset. The training models learned from the classifiers training are applied to the test dataset to evaluate performance of the approach. We further evaluate the performance of neural networks and deep learning algorithms on our dataset. It turns out that with relatively small size of our dataset, the neural network performs poor than multinomial Naive Bayes, but if the dataset size is large, the algorithm can outperform the multinomial Naive Bayes classifier. The approach based on textual word features outputs effective results since the report text is the gist of requested feature enhancement.

The original enhancements dataset acquired from Bugzilla is imbalanced which may affect classifiers performance. To compare the performance with original imbalanced dataset, we also performed re-sampling of the dataset by under-sampling the

reject class since it is almost three times the approve class. To perform the re-sampling, we made the number of rejected reports in each application nearly equal to approved by removing extra reports. For instance, Bugzilla application had 2197 approved reports in the original dataset, and 3537 rejected. After under-sampling the reject class reports, we only selected initial 2197 rejected reports while discarding the remaining 1340 reports. The dataset after re-sampling has total 21,317 reports with 11,072 rejected reports making it roughly balanced. We compare the original dataset results against re-sampled data in our approach.

4.2 Research questions

While evaluating the proposed approach, we address the following research questions.

- RQ1: How accurate are different machine learning algorithms in predicating approval, and which one is best?
- RQ2: Can re-sampling techniques improve the performance of the approach? If yes, to what extent?
- RQ3: Which words should be avoided to increase the likelihood of approval?
- RQ4: How long does it take to train the classifier and to classify new reports?

The research question RQ1 evaluates performance and reliability of our approach in predicting a new report's approval using different machine learning algorithms, i.e., multinomial naive Bayes, decision tree, random forests, logistic regression, and neural networks. Such techniques are selected for comparison as they are popular and accurate in binary classification of text documents (Lamkanfi et al. 2010; Wu et al. 2008; Hu et al. 2014).

Research question RQ2 evaluates the performance of our Bayes based approach on the re-sampled dataset to assess the effects of balanced classes on classification performance. We perform and compare re-sampling on our full dataset since it is imbalanced with approximately 75% of the reports are rejected, which may affect the classifiers' bias towards overrepresented class.

To help improve writing new enhancement reports, in research question RQ3 we explore the affects of words on resolution of the reports. We estimate the likelihood of different words affecting the reject probability of the enhancement reports.

The research question RQ4 measures the time factor of the approach for training and approval prediction. We include the time measure due to three reasons. First, a larger number of reports may take a long training time. Some supervised machine learning techniques, especially neural networks and deep learning algorithms take a long training time when dataset size is large, so this factor becomes more relevant in the evaluation. Second, for classification, the goal is to classify instantly once a new report is submitted. A quick prediction result can save time going through the report manually. Thirdly, large and complex software systems may receive the enhancement requests frequently and the developers may have limited time. In this case, if the approach is computationally complex, it may consume a lot of computing resources which would be ineffective to handle a large number of enhancement reports.

4.3 Dataset

4.3.1 Data retrieval

We extracted the enhancement reports of open-source software applications from Bugzilla,² a general-purpose bug and issue tracking system. Using the Bugzilla REST API,³ we wrote nodejs based code (available online⁴) to access the enhancement reports by specifying the severity level as *enhancement* in the Bugzilla REST API. The severity type of enhancement in the URL filters the feature enhancements from the bug reports, thus retrieves only enhancement reports. Next, detailed description of an enhancement, which is in the first comment of an enhancement report, is separately extracted using the report ID.⁵ We treat this detailed description as the enhancement report because this field describes the requested enhancement by the suggester. We only used the enhancement reports which have been resolved as either fixed or not fixed, discarding the open reports that are not yet decided.

The data acquired is saved in database with appropriate fields for each of the fields of reports retrieved, including ID, product, resolution, description and others. The data retrieved from Bugzilla are reported between 1997-09-10 to 2016-07-13, for the subject applications.

The *resolution* field returned in an enhancement report from the Bugzilla API specifies whether the enhancement report is approved or rejected. In Fig. 2, resolution is presented as the *Status* field. If the resolution resulted in a change to the code base, the enhancement report is resolved as *fixed*. A report is resolved as *invalid* if it is not a proper enhancement. A report is *expired* if it is in *needinfo* status requiring additional information, and the reporter fails to provide the relevant information for more than six months. Since there are multiple resolutions of the reports, we reduce this multi-class resolution problem to binary classification problem by treating a report as approved if its *resolution* is fixed and categorize rejected otherwise. Note that we do not include the new reports in the dataset whose resolution is not defined since we are not sure whether they will be approved or rejected.

Some of the widely used standard issue tracking systems include Redmine, Jira and Bugzilla. Jira is a commercial license based application so we did not consider using the system for our evaluation. These systems do have many peculiar features of their own but the common life cycle of bug reports is more or less similar. Since our approach is based mainly on the enhancement report text, choosing one issue tracking system over the other would not have a bug affect on performance. We chose Bugzilla in our evaluation since it is one of the most popular issue tracking systems for open-source applications including projects like Firefox. Bugzilla also provides a REST API to easily access the bug reports.

² <https://bugzilla.mozilla.org/>, verified 26/02/2016.

³ <https://bugzilla.mozilla.org/rest/bug?severity=enhancement>, verified 26/02/2016.

⁴ <https://github.com/shanniz/Bugzilla>.

⁵ <https://bugzilla.mozilla.org/rest/bug/426904/comment>, verified 26/02/2016.

Table 1 Top 10 open-source applications with most enhancements used in evaluation

Application	Total reports	Domain	Approved	Rejected
SeaMonkey	7922	Internet application suite	883	7039
Core	7223	Shared web browser components pack	2754	4469
Firefox	6793	Web browser	896	5897
Bugzilla	4696	Issue tracking system	2197	2500
Thunderbird	3934	Email client	398	3536
MailNews Core	2050	Mail and news components	376	1674
Toolkit	1678	API services to XUL applications	380	1298
Calendar	1505	Integrated scheduling calendar	439	1066
Camino Graveyard	1168	Legacy Mac OS X browser-only project	344	824
Core Graveyard	1026	Legacy Core components	259	767

The significance of an issue report on Bugzilla is a composite of its priority and severity levels. The priority for an issue is decided and set by the maintainers or developers who plan to work on the bug. General to issue tracking systems and specific to Bugzilla, the priority can have the values of *Immediate*, *Highest*, *High*, *Normal* and *Low*. Bugzilla allows to set these priorities with values from P1 to P5 respectively. The severity field defines the nature of filed issue report in an issue tracking system. Bugzilla specific types of severity for an issue report are *Blocker*, *Critical*, *Major*, *Normal*, *Minor*, *Trivial* and *Enhancement*. Except for enhancement type, other categories of severity are bugs. The enhancement type issue is a request for a new feature or modification in functionality of an existing feature. The severity field gives a high level view of the nature of an issue report and combined with priority field further signifies importance to resolve the issue.

For the most part, the domain of open-source applications obtained from Bugzilla is desktop applications for Internet. The top applications with most number of enhancement reports along-with the number of approved and rejected reports in our dataset are summarized in Table 1. The full list of applications, their domains and number of reports used in the evaluation are available online.⁶ More details and wiki for these applications are accessible online.⁷

4.3.2 Reports' status

The reports extracted for the evaluation have their status *verified* or *closed*. Since for supervised classification system, the data has to be pre-classified, we extracted the enhancements which have already been resolved by the developers as approved or rejected.

The resolution of an enhancement request is specified in resolution field. A total of 8 types of resolution are possible for the enhancement reports in Bugzilla. They

⁶ <https://github.com/zeeshanniz/enhancement.approval.prediction>, verified 30/08/2017.

⁷ https://wiki.mozilla.org/Bugzilla_Products, verified 30/08/2017.

Table 2 Number of reports for each resolution type

Resolution	Number of reports
FIXED	11,072
INVALID	2656
DUPLICATE	16,233
WONTFIX	7186
WORKSFORME	2540
INCOMPLETE	615
EXPIRED	1329
MOVED	4

include DUPLICATE, EXPIRED, FIXED, INCOMPLETE, INVALID, MOVED, WONTFIX, WORKSFORME. Since only the FIXED enhancements are approved for implementation, leaving the rest unimplemented, we classified a report as fixed if its resolution is fixed, and classified non-fixed for other resolutions. Thus we have only two classes of reports reducing the problem to binary classification. The total number of enhancement reports in our dataset from Bugzilla, for each type of report is shown in Table 2.

4.4 Metrics

We use accuracy, precision, recall and f1-score to evaluate performance of the classifiers. Accuracy measures the proportion of all correct predictions. Precision calculates the number of actual true positive outcomes out of all positive predictions. Recall measures the number of true positives returned by classifier from the total number of true positive cases. F1-score is the average of precision and recall.

Mathematically, these metrics are defined as follows

$$Accuracy = \frac{(TP + TN)}{(TP + FN + FP + TN)}$$

$$Precision = \frac{TP}{(TP + FP)}$$

$$Recall = \frac{TP}{(TP + FN)}$$

$$F1-Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

where *TP* (True Positive) is the number of approved reports predicted as approved. *TN* (True Negative) is the number of rejected reports, predicted correctly. *FN* (False Negative) is the number of approved reports predicted as rejected. *FP* (False Positive) is the number of rejected reports predicted as approved. The experimental evaluation of the approach is performed on the system with the hardware and software specifications shown in Table 3.

Table 3 System configuration for evaluation

Resource	Configuration
Processor	Intel® Core™ i5-7200U CPU @ 2.50G x 4
RAM	16 GB
OS	Ubuntu 16.10, 64-bit
Kernel	4.8.0-46-generic Kernel

4.5 Process

The evaluation is carried out as follows. First, we retrieve all enhancement reports (notated as *ER*) from Bugzilla. Second, based on *ER*, we carry out a tenfold cross validation. We randomly partition *ER* dataset into ten equally sized groups notated as G_i ($i = 1 \dots 10$). For the i th cross-validation, we consider all reports except for those in G_i as the corpus of training data, and treat the reports in G_i as the testing data.

For the i th cross-validation, the evaluation process is as follows:

1. First, we extract all reports $trainingData_i$ from training dataset that is the union of all groups but G_i .

$$trainingData_i = \bigcup_{j \in [1,10] \wedge j \neq i} G_j \quad (7)$$

2. Second, we train a Bayes based classifier (*BasClf*) with data from $trainingData_i$.
3. Third, we train a Support Vector Machine based classifier (*SVMClf*) with data from $trainingData_i$.
4. Fourth, we train a Random Forest based classifier (*RFClf*) with data from $trainingData_i$.
5. Fifth, we train a Logistic Regression based classifier (*RFClf*) with data from $trainingData_i$.
6. Sixth, for each report in G_i , we predict its approval with the resulting Bayes based classifier (*BasClf*), SVM based classifier (*SVMClf*), Random Forest based classifier (*RFClf*) and Logistic Regression based classifier (*RFClf*) and compare the results against its actual (correct) status.
7. Finally, we compute accuracy, precision, recall and f1-score for each of the classifiers.

It should be noted that we do not train different classifiers for different applications because the data from a single application is usually too small for training, and thus inner-application prediction may be less accurate than inter-application prediction. In other words, we would make prediction based on the same resulting classifier for different applications.

Table 4 Ten-fold cross validation (Multinomial Naive Bayes)

Iteration	TP	FP	TN	FN	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	666	138	2697	498	84.09	82.83	57.81	68.09
2	472	144	3162	251	82.83	80.54	65.28	72.11
3	348	109	3258	284	90.17	76.14	55.06	63.90
4	256	44	3341	358	89.94	85.33	41.69	56.01
5	265	52	3381	301	91.17	83.59	46.81	60.01
6	556	90	3011	342	89.19	86.06	61.91	72.01
7	776	107	2825	291	90.04	87.88	72.72	79.58
8	830	122	2698	349	88.22	87.18	70.39	77.89
9	1060	139	2484	316	88.62	88.40	77.03	82.32
10	1692	149	1917	241	90.24	91.90	87.53	89.66
Average	692.1	106.4	2877.4	323.1	89.25	84.99	63.26	72.53

4.6 Results

4.6.1 RQ1: Accuracy of different machine learning algorithms in predicating approval

Multinomial naive Bayes, support vector machine, decision tree and neural networks are among the widely used supervised machine learning classification algorithms in text due to their competitive performance (Wu et al. 2008; Sohrawardi et al. 2014). The results of applying these classifiers on the approach revealed that the multinomial naive Bayes classifier yields most accurate results on our dataset (Table 4).

Naive Bayes algorithm achieves competitive classification performance, even though its basis of conditional independence assumption is not often true. In our dataset, the classifier achieves optimal performance. Zhang (2004) argues that the way local dependence of a feature distributes in each class, and how the local dependencies of all features work together, consistently or inconsistently, dictates whether the dependencies distribute evenly in classes, or they cancel each other out. The naive Bayes algorithm is optimal given dependencies are evenly distributed in classes, or they cancel each other out. Apart from that, the naive Bayes is further evaluated for Spam email detection (Zhang and Li 2007), which show that the algorithm is effective in binary classification problem. These works support our application of naive Bayes and its optimal performance on our binary classification problem.

Despite its simplicity, the C++ implementation of multinomial naive Bayes⁸ classifier produced accurate prediction results on tenfold validation test. The implementation uses multinomial event model and the maximum likelihood estimate with a Laplace smoothing technique to learn the parameters. The classifier misclassifies a few reports

⁸ <http://www.openpr.org.cn/index.php/NLP-Toolkit-for-Natural-Language-Processing/43-Naive-Bayes-Classfier/View-details.html>, verified 13/05/2016.

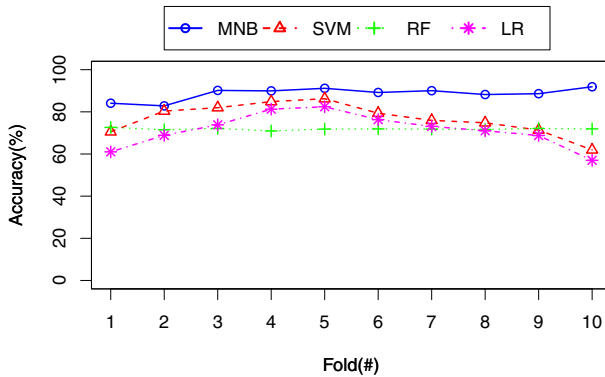


Fig. 4 Accuracy of the approach

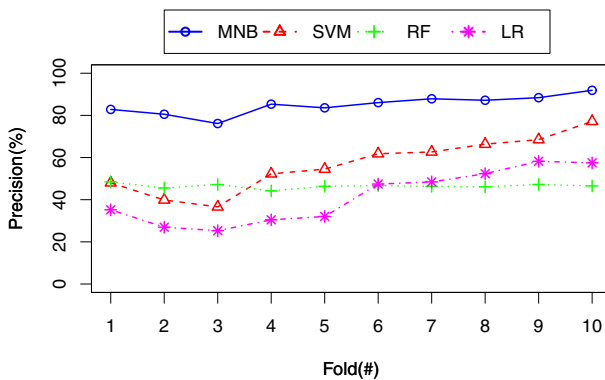


Fig. 5 Precision of the approach

in tenfold cross validation. The misclassifications are mostly false negatives due to the more rejected reports in dataset.

The naive Bayes classifier algorithm is proven effective in many applications including the binary text classification (Hellerstein et al. 2000). Figures 4, 5, 6 and 7 compare the performance of support vector machine, random forests, logistic regression and multinomial naive Bayes classifiers over accuracy, precision, recall and F1 performance parameters respectively showing that the multinomial naive Bayes implementation outperforms the compared algorithms in all performance metrics.

Support vector machine Support vector machine is another classifier shown effective in text classification (Zaghloul et al. 2009). We trained the support vector machine classifier implementation called *SVM^{light}*⁹. The classifier implementation can process hundred thousands of training vectors and handle some thousands of support vectors. The parameter *c* is the trade-off between training error and the support vector margin. The parameter *c* value of 20.0 is used in the classifier training. Table 5 presents the performance of support vector machine classifier on tenfold cross validation. Total

⁹ <http://svmlight.joachims.org>, verified 27/05/2016.

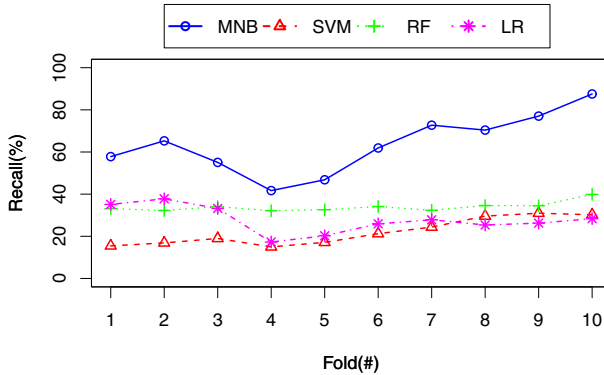


Fig. 6 Recall of the approach

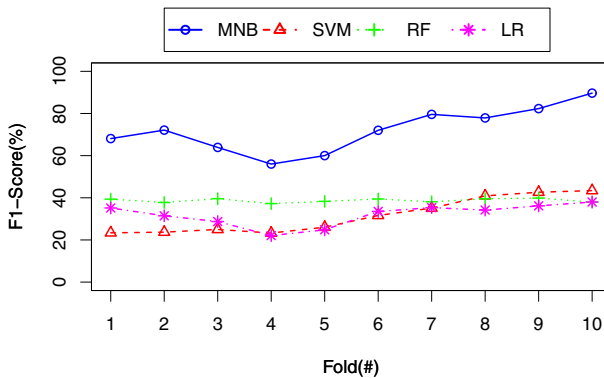


Fig. 7 F1-Score of the approach

number of the reports in one set is 4000. Thus each training dataset has 36,000 reports while a single test set has 4000 reports.

The decision tree¹⁰ algorithm performed poor in the evaluation. The classifier ran out of memory on our system, without generating the classification model while trained on the training dataset in the tenfold validation. We excluded the decision tree classifier from the evaluation due to its poor performance.

Random forest and logistic regression Random forests and logistic regression have been shown prominent in certain software bug handling studies (Xia et al. 2015; Valdivia Garcia and Shihab 2014; Antoniol et al. 2008). For this reason, we assessed the performance of algorithms with same evaluation metrics and tenfold cross validation. We used the same aggregate dataset of all the 35 subject applications' reports text modeled as feature vectors. The results of applying these algorithms on our dataset were not as effective compared to Bayes based classification model. Random forest classifier exploits ensemble learning technique. The standard implementation from *sklearn.ensemble* module for *RandomForestClassifier* was evaluated for cross valida-

¹⁰ <https://github.com/yandongliu/learningjs>, verified 20/05/2016.

Table 5 Ten-fold cross validation (Support Vector Machine)

Iteration	TP	FP	TN	FN	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	150	159	2640	984	70.53	48.00	15.46	23.38
2	96	154	3092	601	80.38	39.87	16.87	23.70
3	93	157	3159	512	81.97	36.59	18.96	24.97
4	72	72	3301	522	84.85	52.27	14.98	23.28
5	66	69	3352	469	86.22	54.49	17.11	26.04
6	141	100	2983	707	79.35	61.81	21.25	31.62
7	195	129	2777	807	75.95	62.65	24.37	35.09
8	237	133	2643	831	74.80	66.35	29.58	40.91
9	261	152	2427	950	71.35	68.49	30.96	42.64
10	389	141	1893	1349	61.95	77.15	30.21	43.41
Average	170.0	126.6	2826.7	773.2	76.73	56.76	21.97	31.50

Table 6 Ten-fold cross validation (Random Forests)

Fold	TP	FP	TN	FN	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
1	370	394	2654	746	72.62	48.42	33.15	39.36
2	360	430	2618	756	71.51	45.56	32.25	37.77
3	380	425	2623	736	72.11	47.20	34.05	39.56
4	359	453	2595	757	70.94	44.21	32.16	37.24
5	364	421	2627	752	71.82	46.36	32.61	38.29
6	381	435	2613	735	71.90	46.69	34.13	39.44
7	361	418	2630	755	71.82	46.34	32.34	38.10
8	386	451	2597	730	71.63	46.11	34.58	39.52
9	385	430	2618	731	72.11	47.23	34.49	39.87
10	358	411	2637	758	71.92	46.55	32.07	37.98
Average	370.4	426.8	2621.2	745.6	71.84	46.47	33.18	38.71

tion scores. Table 6 shows the performance of random forest classifier. The average for accuracy, precision, recall and f1-score are respectively 58.07, 58.50, 70.18 and 63.81% over ten folds cross validation.

Table 7 summarizes the performance of logistic regression classifier from sklearn library. The algorithm outputs an average accuracy of 71.36%. However, it does not perform optimally in terms of precision, recall and f1-score which on average are respectively, 41.38, 27.74 and 31.94% over ten folds cross validation.

To investigate whether there is essential difference between the accuracy of the Bayes based approach and the accuracy of alternative classification techniques, we apply ANOVA analysis on their resulting accuracy in tenfold evaluation (ten different accuracy values for each of the techniques). The results of ANOVA analysis are shown in Table 8. For each of the comparisons present in the table, $F > F_{crit}$ and $Pvalue < (\alpha = 0.05)$. These results suggest that the factor (different clas-

Table 7 Ten-fold cross validation (Logistic Regression)

Fold	TP	FP	TN	FN	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
1	423	778	2018	781	61.02	35.22	35.13	35.17
2	286	775	2468	471	68.85	26.95	37.78	31.46
3	211	623	2742	424	73.82	25.29	33.22	28.72
4	106	242	3145	507	81.27	30.45	17.29	22.06
5	116	246	3181	457	82.42	32.04	20.24	24.81
6	239	265	2813	683	76.30	47.42	25.92	33.52
7	294	313	2630	763	73.10	48.43	27.81	35.33
8	301	274	2540	885	71.02	52.34	25.37	34.18
9	354	254	2397	995	68.77	58.22	26.24	36.17
10	527	390	1753	1330	57.00	57.47	28.37	37.99
Average	285.7	416.0	2568.7	729.6	71.36	41.38	27.74	31.94

Table 8 ANOVA analysis on accuracy

Source of variation	SS	df	MS	F	P value	F critical
<i>MNB versus SVM</i>						
Between groups	0.070590962	1	0.070590962	22.22338	0.000172997	4.413873419
Within groups	0.057175686	18	0.003176427			
Total	0.127766648	19				
<i>MNB versus RF</i>						
Between groups	0.140767420	1	0.140767420	317.9988106	6.90912E-13	4.413873419
Within groups	0.007967997	18	0.000442667			
Total	0.148735417	19				
<i>MNB versus LR</i>						
Between groups	0.1489538	1	0.1489538	40.73390441	5.20036E-06	4.413873419
Within groups	0.065821542	18	0.003656752			
Total	0.214775342	19				

sification techniques) did cause significant difference in resulting accuracy, and the multinomial naive Bayes based approach results in best performance.

The distribution of accuracy over tenfold cross validation for multinomial naive Bayes, support vector machine, random forests and logistic regression is presented in Fig. 8. A beanplot plots the beans, one bean per group to compare the distributions of different groups. A bean is a one-dimensional scatter plot consisting of the data distribution as a density shape. The accuracy of individual folds are represented as horizontal lines within the bean whereas the average accuracy is represented as the longer line across the bean. The shape of the bean is the density, and the longer bold line represents the average accuracy of each classifier over tenfold cross validation.

As observable in Fig. 8, the multinomial naive Bayes classifier exhibits a high accuracy and a small deviation in the values through the different folds of tenfold

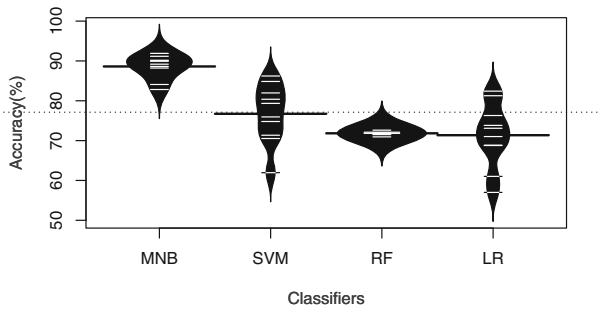


Fig. 8 Accuracy distribution of the approach

validation. Random forest classifier shows a consistent but lower accuracy as compared to multinomial naive Bayes model. Support vector machine classifier and logistic regression do not show consistency in accuracy across different folds of tenfold cross validation and a relatively lower performance as compared to the multinomial naive Bayes classifier. The bean-plot suggests that the lowest accuracy of multinomial naive Bayes is still comparable to the highest accuracy of support vector machine and logistic regression in the tenfold cross validation.

We conclude the preceding analysis that multinomial naive Bayes classifier outperforms other classification models in the approval prediction of enhancement reports. The Bayes is an effective model for text classification and the proposed approach is also based on the text. Another possible reason is that the naive Bayes classifier is more effective for binary classification than multi-class classification (Rish et al. 2001), and the proposed approach classifies the reports into two classes only.

The relatively lower recall rate in multinomial Naive Bayes is a result of more reports being misclassified as rejected. Applying an scaling factor s to the posterior probability of *approve* class to increase its probability can be used to adjust the precision and recall of the approach. A developer can choose to have either higher precision or recall for the reports depending on which is more important to him, by scaling the class weight accordingly. With the scaling factor, posterior probability of a class (e.g. approve) becomes

$$P(\text{approve}) = s * P(\text{approve}) \quad (8)$$

Figure 9 shows the effect of applying the scaling factor s values between 1.001 to 1.019 with step size of 0.003. The average results of the tenfold validation on Bayes based approach show that the precision improves as a result of applying more weight to the approve class posterior probability due to less number of false negative or reject classifications. Thus there is a trade-off between precision and recall with the change in scaling factor value.

Comparing against deep learning algorithms To get more optimal performance, we applied Deep Belief Networks (DBN) for deep learning and compared it with the Multi-Layer Perceptron (MLP),¹¹ a neural network algorithm. The deep learning

¹¹ <http://deeplearning.net/>, verified 10/08/2016.

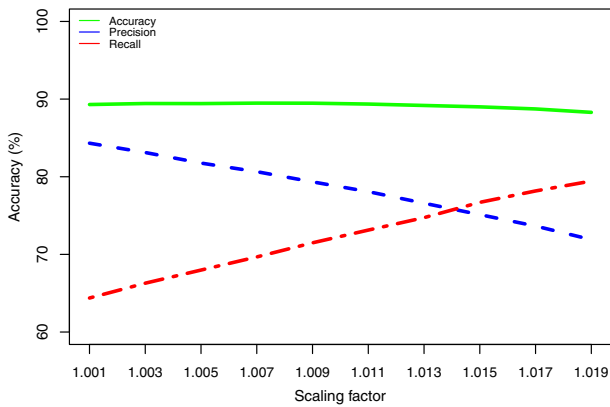


Fig. 9 Effect of scaling the class probability on the performance

algorithms require compact representation of the data for better performance. The *paragraph vector* algorithm (Le and Mikolov 2014) generates fixed length feature representation from text reports of varying lengths. We applied the *paragraph2vec* implementation (Liu et al. 2015) of the algorithm to convert the enhancement reports into feature vectors. Since the algorithm allows to generate feature vectors with any number of dimensions, we tried and found that 100-dimensional feature vectors are applicable without any performance reduction.

The dataset we used for deep learning contains set of pairs (y^i, x^i) , where x^i is the compact representation of the report text (100-dimensions vector) and y^i is the label (0 or 1) associated with the report. We divide the dataset into training and validation sets. The dataset is divided into ten parts. One part forms the validation set while the remaining parts form the training set.

We built a deep belief network (DBN), with an input layer of 100 neurons, three hidden RBM layers of 1000 neurons each, and a logistic regression output layer of 2 neurons. Since a DBN has many possible configurations and parameters, we applied different learning rates and mini-batch sizes according to our dataset and feature vectors. The network was trained with the report vectors from paragraph to vector algorithm. The parameters used in training DBN were; learning rate of 0.1, training epochs were 1000, and batch size of 10. DBN performed poor with the validation error 77.87% on our dataset of 40,000 reports. Furthermore, the performance of the algorithm was not consistently improving as we tried different sized datasets. The classifier takes relatively higher training and classification time of 187.8 and 1.021 seconds respectively.

We built multilayer perceptron (MLP) with an input layer of 100-dimensions, a hidden layer of 1000 sigmoids, and a logistic regression classifier layer which outputs one of the two classes. The first input layer takes the reports feature vectors as input and forwards it to the hidden layer for the model parameters optimization. Finally the output logistic regression layer uses the hidden layer activations for final binary classification. It turned out during evaluations that with the given dataset, the performance of the deep learning approach is lower than the multinomial naive Bayes based approach. Since deep learning approaches usually require large sized dataset to get better performance,

Table 9 Multi-layer neural network performance improvement with respect to dataset size

Dataset size	Error rate (%)
10,000	28.50
15,000	25.46
20,000	25.45
25,000	22.76
30,000	20.56
35,000	20.45
40,000	18.92

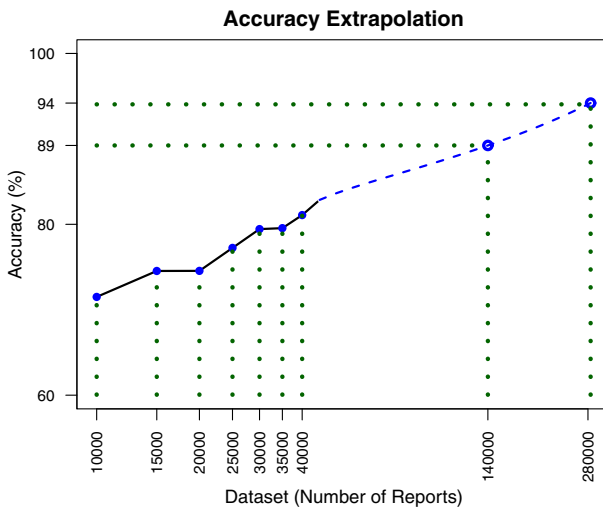


Fig. 10 Validation accuracy extrapolation for neural network based approach

we tested different sized subsets of our original corpus to observe if the results improve with increasing dataset size. The MLP algorithm has shown improvement as we used increasing size of the dataset.

Table 9 shows that the validation accuracy of MLP gradually increases with the increasing dataset size. DBN’s performance, however, does not exhibit consistency with the increasing dataset size. We extrapolated the validation accuracy of MLP for larger dataset size using the forecast function in R. The forecast method uses ARIMA modeling to extrapolate the values. Forecast is a generic function for forecasting from time series models. We used 30 periods for forecasting 30 data-points with intervals of 5000 records.

The result of extrapolation in Fig. 10 (the blue dotted line is the actual dataset accuracy, red line is extrapolated accuracy) shows that for a dataset of 140,000 reports, the accuracy of MLP equals the accuracy of the Bayes based approach. The extrapolation of doubled dataset size of 280,000 reports suggests that the output of the neural networks based approach would be 94.19% accurate. This trend of performance

improvement shows that if the dataset size is sufficiently large, the deep learning technique may outperform the multinomial Naive Bayes based approach.

Although a large sized training data may be difficult to obtain, but based on the improvement trend proportional to increase in dataset size, we suggest to use the neural network based approach for more accurate prediction of the enhancement reports when large labeled dataset is available for training.

4.6.2 RQ2: Influence of re-sampling

Different ways of re-sampling include over-sampling the underrepresented class, under-sampling the overrepresented class, or sometimes changing the classifier threshold for one of the classes to give more weight to under-represented class or less weight to over-represented class. We performed re-sampling of the dataset by *under-sampling* the reject class since a large number of reports are rejected in our dataset. The resulting balanced dataset with proportionate approved and rejected reports is compared in evaluation with the original imbalanced dataset.

To determine the influence of re-sampling, we evaluate the Bayes based approach on the balanced dataset. Same reports drawing method is used as with the original dataset from all the subject applications. Since our dataset is imbalanced with more rejected reports, we performed under-sampling of rejected class to equal the number of rejected reports to approved reports in training set. In the tenfold cross validation, only the training fold was subjected to under-sampling with balanced number of reports of both classes in each fold.

To under-sample the dataset of 40,000 reports, 36,000 reports were drawn as training set in each fold, and under-sampled to have equal number of approve and reject reports. The under-sampling was performed by randomly eliminating the rejected reports. Total number of rejected reports eliminated depends on the number of approved reports in the training set. The resulting training set has half approved and half rejected reports, where the total number of reports in each fold varies slightly, since the number of approved reports in a fold varies and the total number of reports become double of this number. The testing fold dataset of 4000 (10%) reports was used unchanged with more rejected reports. The trained classifiers on re-sampled datasets were evaluated on the original test set (not re-sampled). This technique is applied to mimic the real test situation of more rejected reports in the issue tracking system.

The tenfold cross validation results of multinomial naive Bayes based approach with re-sampled reports dataset are presented in Table 10. Each row shows result of the corresponding fold number in tenfold cross validation. The output of a fold are total true positive, true negative, false positive and false negative predictions of the fold, which are used to calculate the accuracy, precision, recall and f1-score of the fold. The average accuracy of the approach is 81.66%, with precision, recall and f1-score being 58.94, 89.68 and 70.51% respectively.

We further evaluated the classifiers on balanced training dataset with less number of reports. Table 11 summarizes the average accuracy, precision, recall and f1-score of the classifiers with under-sampled dataset. Each cell in the table shows performance on balanced dataset along-with corresponding value in parenthesis for the full

Table 10 Multinomial naive Bayes tenfold cross validation performance with re-sampling

Iteration	TP	FP	TN	FN	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	961	346	2449	243	85.27	73.52	79.81	76.54
2	626	339	2903	131	88.24	64.87	82.69	72.70
3	489	322	3043	145	88.32	60.29	77.12	67.68
4	510	364	3023	102	88.34	58.35	83.33	68.64
5	513	575	2851	60	84.12	47.15	89.52	61.77
6	871	785	2292	51	79.09	52.59	94.46	67.57
7	1021	931	2011	36	75.81	52.30	96.59	67.86
8	1157	939	1875	28	75.81	55.20	97.63	70.52
9	1320	1160	1491	28	70.29	53.22	97.92	68.96
10	1813	708	1435	43	81.22	71.91	97.68	82.84
Average	928.1	646.9	2337.3	86.7	81.66	58.94	89.68	70.51

Table 11 Average tenfold cross validation performance with and without re-sampling

Metrics	MNB	SVM	RF	LR
Accuracy (%)	81.66 (89.25)	76.70 (76.73)	55.43 (71.84)	50.31 (71.36)
Precision (%)	58.94 (84.99)	57.52 (56.76)	29.34 (46.47)	28.13 (41.38)
Recall (%)	89.68 (63.26)	21.51 (21.97)	56.37 (33.18)	63.23 (27.74)
F1-Score (%)	70.51 (72.53)	31.31 (31.50)	37.83 (38.71)	38.03 (31.94)

imbalanced dataset. The results suggest that re-sampling the enhancements corpus does not improve accuracy of the approval prediction. In many cases, re-sampling is a useful technique deal with the over-fitting problem due to skewed data. However, in our case, it does not work well. We have not yet figured out exactly the reason. One possible reason is that the chosen re-sampling technique is not suitable in our case and in future replacing it with other re-sampling techniques may improve the accuracy.

4.6.3 RQ3: Negative words

Lamkanfi et al. (2010) investigated the problem of predicting the bug reported as *non-severe* or *severe* using naive Bayes classifier. The authors calculated probability of the words in both severe and non-severe bug reports and found that the most appearing significant words appear across different applications. Considering this finding, we try to identify some words that are more likely to appear in each of the enhancement reports classes.

We calculate the significance of each word feature affecting the rejection likelihood. The words likelihood of rejection is calculated in general for all applications without calculating the likelihood for each individual application. The TF/IDF is a useful technique to measure the significance of the words in a document. We sorted the

Table 12 Some words with highest reject probability P_{reject}

Word	Likelihood of getting rejected (%)
Meaningless	87.50
Stupid	92.30
Awesome	90.37
Offensive	90.00
Receive	89.90
Reproduce	89.24
Crap	87.50
Funny	81.81
Monster	80.00
Guess	78.43
Suspect	76.19

words with highest occurrences in the enhancement reports according to TF/IDF. We however, take all the reports belonging to the same class as a document to calculate term frequency (TF) of the word and the entire set of reports in which the words appear to calculate inverse document frequency (IDF). Sorting the words based on the probability with this technique suggests that if the likelihood of a class is high, the word has occurred in most of the reports of that class and thus affects the likelihood of that particular class. Equation 9 depicts the formula to calculate the reject likelihood of word.

$$P(w_i, reject) = \frac{count(d_{reject}, w_i)}{count(d_{total}, w_i)} \quad (9)$$

where $P(w_i, reject)$ is the reject probability of word w_i , $count(d_{reject}, w_i)$ is the number of rejected reports in which the word appears, and $count(d_{total}, w_i)$ is the total number of reports in which the word appears.

Table 12 lists the words that are frequently associated with rejected enhancement reports. Most of such words are negative revealing the anger and satire of the reporters. It is often difficult for reports with such feelings to specify the enhancement report clearly, let alone providing constructive suggestions. Consequently, such reports are more likely to be rejected. It is not to say that all the reports containing such words are doomed to rejection, but that they are more likely to be rejected than those described in a more constructive way.

4.6.4 RQ4: Time complexity of the approach

We calculate the time complexity of the approach by measuring enhancement reports lemmatization time, and the training and testing times of the classifier. For lemmatization, we measured and averaged the time of five lemmatization requests of the same report. The time is measured by initializing the timer just before the call to the lemmatization REST API server and stopping it just after the server response is received. The

Table 13 Average training and testing time of the classifiers in seconds

Classifier	Training time	Testing time
Multinomial naive Bayes	0.3649	0.0798
Support vector machine	0.2483	0.0364
Random forest	1.6922	0.0143
Logistic regression	0.9072	0.0009
MLP	9.66	0.617
DBN	187.8	1.021

total time is calculated as the difference between the start and stop time interval. The average time of the lemmatization over five iterations on our system is 2.18 seconds which is a rough estimate of the actual lemmatization time since the lemmatization API is an external program hosted online. The average time for lemmatization is the combination of communication time (lemmatization request to the server and the response time) and the time of text lemmatization process. The communication time depends on the underlying network system, and is an important factor as it affects the total execution time.

To measure the execution time of the classifier training and prediction, we calculated the average execution time of five trials on one fold for training and prediction of the classifiers. The average time is calculated in seconds scale on our system with training dataset of 36,000 reports and test dataset of 4000 reports. The results are depicted in the Table 13.

The runtime time measures for the neural network based classifiers were performed with the fixed parameters. The training time is higher for neural networks, and significantly higher in case of DBN as compared to MLP.

For training to generate the classifier, multinomial naive Bayes algorithm is most time efficient compared to the other classifiers. While the logistic regression implementation is relatively fast in case of classification time.

The time and space complexity of naive Bayes classifier is linear to the number of training reports and the number of features. Theoretically, the training time complexity for the naive Bayes classifier is given by

$$O(|D|L_d + |C||V|) \quad (10)$$

where L_d is the average length of report vector, $|D|$ is the number of reports used in training, $|C|$ is the number of distinct classes of the reports and $|V|$ is the feature vectors size. The testing time for the naive Bayes classifier is given by

$$O(|C|L_t) \quad (11)$$

where L_t is the average length of a test report vector and $|C|$ is the total number of distinct classes to classify the reports.

5 Threats and limitations

5.1 Threats to validity

5.1.1 Construct validity

First threat to construct validity is the accuracy of the labeled reports. We assume the reports are correctly classified by the developers which may not be correct for all the reports. Thus our training and testing may have slight inaccuracy. We only used the reports that are resolved and closed to minimize this inaccuracy.

Second threat to construct validity results from the extrapolation of the deep learning based approach, which may not be accurate. Since deep learning algorithms usually require a large training dataset for better performance, but we have limited data, so we extrapolate the results to observe expected results on more data. However, it is possible that the actual performance may not be as good as shown in extrapolation.

5.1.2 Internal validity

First threat to the internal validity is that the lemmatization process of the enhancement reports may be inaccurate. This inaccuracy is possible because a natural language tends to be ambiguous and the words change over time which are possible problems for lemmatization program. To counter this threat, we used one of the well known and standard lemmatization API and cross-checked sample lemmatization outcomes of the API with Stanford NLP library to validate the results.

Second threat to internal validity is that the deep learning algorithms have a number of parameters to be adjusted and performance is influenced by the settings of such parameters. These parameters include the initial weights, learning rate, activation, number of layers in the network, and so on. Thus it is possible that the neural network algorithms underperformed due to non-optimal parameters.

The third threat to internal validity comes from the approach being limited to only two outcomes of resolution status. We treat a report as approved if its resolution is fixed, and reject otherwise.

5.1.3 External validity

First threat to validity is the language of the enhancement reports. We evaluated the approach on English based text of the reports. Since our approach is based on textual features of an enhancement request on which a classification model is trained, the performance may not be as effective in case of the reports written in other languages, for instance in Chinese.

Second threat to external valid is that only one re-sampling technique was used (Sect. 4.6.2). Since there are different data balancing techniques available, the performance may be affected by the technique used. We chose under-sampling as it is a standard re-sampling technique that limits itself within existing corpus and does not require creation of new data.

The third threat to external validity is that the evaluation is performed on the applications from Bugzilla issue tracking system. Bugzilla allows to access a limited number of open-source applications data. We therefore examined the performance of our approach with a limited number of reports from the open-source applications. The domain of these applications is mostly the Internet and desktop applications. Therefore the approach may not perform well on other domains like smart-phone applications. The evaluation on more applications may be conducted to reinforce the conclusions.

5.2 Limitations

The proposed approach is limited to text description of the reports. However, other factors like available resources, business concerns, budget and reporters may also influence the approval decision. Such factors are difficult to obtain and quantify, however, in future, these factor may be incorporated that may improve performance of the approach.

The second limitation of the proposed approach is that it classifies the enhancement reports into only two groups, the approved and non-approved. An enhancement report can have a number of possible status like approve, reject, duplicate and invalid. We treat all the reports that are not approved as rejected and make no further classifications. We make such a binary classification instead of multi-class classification because of the following reasons:

1. First, the main purpose of the approach is to recommend those reports that are likely to be approved and ignore others. Consequently, classifying the reports into approved and non-approved is enough for the task.
2. Second, reducing the number of classes helps improve classification accuracy.

6 Conclusion

A majority of enhancement reports of the software applications on Bugzilla are not approved which results in wastage of time in proposing and the manual evaluation of these reports. The approach in this paper shows that it is possible to automatically predict whether an enhancement report should be approved or rejected. It helps enhancement reporter see beforehand if the report is likely to be approved or not. This would translate to lower number of reports for developers to evaluate thus saving their time.

Although, the reports need to be finally decided and implemented by the developers, the proposed approach would help the reporters to forward better reports. This will limit the number of proposed reports to the developer for evaluation. From developers perspective, the automated predication approach is helpful in assisting the developers rank more likely reports and resolve them first.

Our supervised machine learning based approach learns from the history data to classify approved reports from rejected. We evaluated performance of the approach on enhancement reports of open-source applications acquired from Bugzilla. We conclude that the prediction accuracy of the Bayes based approach is optimal given sufficient

training data of two enhancement reports classes, with average precision and recall of 84.99 and 63.26% respectively.

We observed relatively low performance in certain test cases and observed that some words affect the results accuracy. A weak point in the multinomial naive Bayes classifier is that when one or more words appear in only one class in the training dataset but occur in the other class in test dataset, it reduces the overall likelihood of the actual class by the classifier. A smoothing factor is applied to minimize the effect of such words. Furthermore, the imbalanced nature of the dataset may also affect the classifier with biased results towards more occurring class.

An important resolution for enhancements is duplicate report. Although a requested feature may be good enough but duplicate of another enhancement, thus being rejected. Duplicated bug report detection tools could be employed to remove duplicate reports and thus avoid duplicate (and unnecessary) approval prediction. However, it is not required for our approach to be filtered through duplicate detection first. The approach works even if duplicate reports exist.

Some of the rejected enhancement requests may have used the words that did not express the enhancement requirement clearly and constructively. Some of these words are mentioned in Sect. 4.6.3 (Table 12). However, not all the requests with these terms are rejected. The approval is not decided by a single word but usually with a group of words. The reporter may change the keywords or use such words if necessary, in a more clear and constructive way to make it more likely to be approved. Our prediction approach therefore alerts the reporter before the request is actually submitted, to improve chances of approval. Furthermore, it can save time of the software maintainer by filtering more conceivable enhancement reports.

The evaluation results further show that the increase in dataset size reduces the error rate in deep learning technique. We therefore conclude that the deep learning algorithms may outperform the Bayes based approach when the dataset is sufficiently large.

The proposed approach is merely based on the text description of an enhancement report. However, other factors like available resources, business concerns, budget and reporters may also influence the approval decision. In future, an enhanced prediction model may incorporate such factors to improve performance of the approach.

Acknowledgements The work is supported by the National Key Research and Development Program of China (2016YFB1000801) and the National Natural Science Foundation of China (61472034, 61772071, 61690205).

References

- Uysal, A.K., Gunal, S.: The impact of preprocessing on text classification. *Inf. Process. Manag.* **50**(1), 104–112 (2014)
- Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.G.: 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, p. 23 (2008)
- Anvik, J.: Automating bug report assignment. In: *Proceedings of the 28th international Conference on Software engineering*, ACM, pp. 937–940 (2006)

- Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering, ACM, pp. 361–370 (2006)
- Banerjee, S., Cukic, B., Adjeroh, D.: Automated duplicate bug report classification using subsequence matching. In: 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE), IEEE, pp. 74–81 (2012)
- Bhattacharya, P., Neamtiu, I., Shelton, C.R.: Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *J. Syst. Softw.* **85**(10), 2275–2292 (2012)
- Chen, J., Huang, H., Tian, S., Qu, Y.: Feature selection for text classification with naive bayes. *Expert Syst. Appl.* **15**, 2160–2164 (2011)
- Chen, Z., Lü, K.: A preprocess algorithm of filtering irrelevant information based on the minimum class difference. *Knowl.-Based Syst.* **19**(6), 422–429 (2006)
- Delany, S., Buckley, M., Greene, D.: SMS spam filtering: methods and data. *Expert Syst. Appl.* **39**(10), 9899–9908 (2012)
- Domingos, P., Pazzani, M.: On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn.* **29**(2), 103–130 (1997)
- Eberhardt, J.: Bayesian spam detection. *Sch. Horiz. Univ. Minn. Morris Undergrad. J.* **2**(1), 2 (2015)
- Feng, L., Song, L., Sha, C., Gong, X.: Practical duplicate bug reports detection in a large web-based development community. In: *Web Technologies and Applications*, Springer, pp. 709–720 (2013)
- Gad, W., Rady, S.: Email filtering based on supervised learning and mutual information feature selection. In: 2015 Tenth International Conference on Computer Engineering & Systems (ICCES), IEEE, pp. 147–152 (2015)
- Gopalan, R., Krishna, A.: Duplicate bug report detection using clustering. In: *Software Engineering Conference (ASWEC), 2014 23rd Australian*, IEEE, pp. 104–109 (2014)
- Hellerstein, J., Thathachar, J., Rish, I.: *Recognizing End-User Transactions in Performance Management*, vol. 19. IBM Thomas J. Watson Research Division, New York (2000)
- Herzig, K., Just, S., Zeller, A.: 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, pp. 392–401 (2013)
- Hindle, A., Alipour, A., Stroulia, E.: A contextual approach towards more accurate duplicate bug report detection and ranking. *Empir. Softw. Eng.* **21**(2), 368–410 (2016)
- Hu, H., Zhang, H., Xuan, J., Sun, W.: Effective bug triage based on historical bug-fix information. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp. 122–132 (2014)
- Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ACM, pp. 111–120 (2009)
- Jiang, L., Cai, Z., Zhang, H., Wang, D.: Naive bayes text classifiers: a locally weighted learning approach. *J. Exp. Theor. Artif. Intell.* **25**, 273–286 (2013)
- Jin, Z., Li, Q., Zeng, D., Wang, L.: Filtering spam in Weibo using ensemble imbalanced classification and knowledge expansion. In: 2015 IEEE International Conference on Intelligence and Security Informatics (ISI), IEEE, pp. 132–134 (2015)
- Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, pp. 1–10 (2010)
- Lamkanfi, A., Demeyer, S., Soetens, Q., Verdonck, T.: Comparing mining algorithms for predicting the severity of a reported bug. In: 2011 15th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, vol. 322, pp. 249–258 (2011)
- Lazar, A., Ritchey, S., Sharif, B.: Improving the accuracy of duplicate bug report detection using textual similarity measures. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, pp. 308–311 (2014)
- Le, Q., Mikołov, T.: Distributed representations of sentences and documents. In: *international Conference on Machine Learning, 2014, ICML*, vol. 14, pp. 1188–1196 (2014)
- Lin, M., Yang, C., Lee, C., Chen, C.: Enhancements for duplication detection in bug reports with manifold correlation features. *J. Syst. Softw.* **121**, 223–233 (2016)
- Liu, Y., Liu, Z., Chua, T., Sun, M.: Topical word embeddings. In: *The 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, AAAI, pp. 2418–2424 (2015)
- Murphy, G., Cubranic, D.: Automatic bug triage using text categorization. In: *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, Citeseer (2004)

- Menzies, T., Marcus, A.: Automated severity assessment of software defect reports. In: IEEE International Conference on Software Maintenance, ICSM, pp. 346–355 (2008)
- Naguib, H., Narayan, N., Brügge, B., Helal, D.: Bug report assignee recommendation using activity profiles. In: 2013 10th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, pp. 22–30 (2013)
- Pingclasai, N., Hata, H., Matsumoto, K.: Classifying bug reports to bugs and other requests using topic modeling. In: Software Engineering Conference (APSEC), 2013 20th Asia-Pacific, IEEE, vol. 2, pp. 13–18 (2013)
- Rajlich, V.: Software evolution and maintenance. In: Proceedings of the on Future of Software Engineering, ACM, pp. 133–144 (2014)
- Rish, I.: An empirical study of the naive bayes classifier. In: IJCAI 2001 workshop on empirical methods in artificial intelligence, IBM New York, vol. 3, pp. 41–46 (2001)
- Rish, I., Hellerstein, J., Jayram, T.: An analysis of data characteristics that affect naive bayes performance. IBM TJ Watson Research Center 30 (2001)
- Roy, N.K.S., Rossi, B.: Towards an improvement of bug severity classification. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE, pp. 269–276 (2014a)
- Roy, N.S., Rossi, B.: Towards an improvement of bug severity classification. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE, pp. 269–276 (2014b)
- Santos, I., Laorden, C., Sanz, B., Bringas, P.: Enhanced topic-based vector space model for semantics-aware spam filtering. *Expert Syst. Appl.* **39**(1), 437–444 (2012)
- Saric, F., Glavas, G., Karan, M., Snajder, J., Basic, B.: 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, pp. 441–448 (2012)
- Schölkopf, B., Burges, C.: *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge (1999)
- Sohrawardi, S.J., Azam, I., Hosain, S.: A comparative study of text classification algorithms on user submitted bug reports. In: 2014 Ninth International Conference on Digital Information Management (ICDIM), IEEE, pp. 242–247 (2014)
- Su, J., Shirab, J., Matwin, S.: Large scale text classification using semi-supervised multinomial naive bayes. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 97–104 (2011)
- Sun, C., Lo, D., Khoo, S., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, pp. 253–262 (2011)
- Tan, S., Wang, Y., Wu, G.: Adapting centroid classifier for document categorization. *Expert Syst. Appl.* **38**(8), 10,264–10,273 (2011)
- Thung, F., Kochhar, P.S., Lo, D.: Dupfinder: integrated tool support for duplicate bug report detection. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ACM, pp. 871–874 (2014)
- Tian, Y., Sun, C., Lo, D.: Improved duplicate bug report identification. In: 2012 16th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, pp. 385–390 (2012)
- Valdivia Garcia, H., Shihab, E.: Characterizing and predicting blocking bugs in open source projects. In: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, pp. 72–81 (2014)
- Wang, S., Jiang, L., Li, C.: Adapting naive bayes tree for text classification. *Knowl. Inf. Syst.* **44**(1), 77–89 (2014)
- Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering, ACM, p. 461470 (2008)
- Wei, Z., Feng, G.: An improvement to naive bayes for text classification. *Proc. Eng.* **15**, 2160–2164 (2011)
- Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y., et al.: Top 10 algorithms in data mining. *Knowl. Inf. Syst.* **14**(1), 1–37 (2008)
- Xia, X., Lo, D., Shihab, E., Wang, X., Yang, X.: Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Inf. Softw. Technol.* **61**, 93–106 (2015)

- Xuan, J., Jiang, H., Ren, Z., Yan, J., Luo, Z.: Automatic bug triage using semi-supervised text classification (2017). arXiv preprint [arXiv:1704.04769](https://arxiv.org/abs/1704.04769)
- Xuan, H., Ming, L.: Enhancing the Unified Features to Locate Buggy Files by Exploiting the Sequential Nature of Source Code. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, pp. 1909–1915 (2017)
- Yang, J., Liu, Y., Zhu, X., Liu, Z., Zhang, X.: A new feature selection based on comprehensive measurement both in inter-category and intra-category for text categorization. *Inf. Process. Manag.* **48**(4), 741–754 (2012)
- Zaghloul, W., Lee, S.M., Trimi, S.: Text classification: neural networks vs support vector machines. *Ind. Manag. Data Syst.* **109**(5), 708–717 (2009)
- Zhang, H.: The optimality of naive bayes. *AA* **1**(2), 3 (2004)
- Zhang, H., Li, D.: Naïve bayes text classifier. In: IEEE International Conference on Granular Computing, 2007. GRC 2007, IEEE, pp. 708–708 (2007)
- Zhang, W., Tang, X., Yoshida, T.: TESC: An approach to TEXT classification using semi-supervised clustering. *Knowl.-Based Syst.* **75**, 152–160 (2015)
- Zhang, Y., Wang, S., Phillips, P., Ji, G.: Binary PSO with mutation operator for feature selection using decision tree applied to spam detection. *Knowl.-Based Syst.* **64**, 22–31 (2014)
- Zhou, J., Zhang, H., Lo, D.: Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. In: Proceedings of the 34th International Conference on Software Engineering, IEEE Press, pp. 14–24 (2012)
- Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., Weiss, C.: What makes a good bug report? *IEEE Trans. Softw. Eng.* **36**(5), 618–643 (2010)