AUTOMATIC APPROVAL PREDICTION FOR SOFTWARE ENHANCEMENT REQUESTS

ZEESHAN AHMED NIZAMANI

安木德

2017 年 6 月

中图分类号	:	TP311.11
UDC 分类号	:	004.8

软件增强性需求的自动化预测方法

作者姓名 (中文)	:	安木德
指导教师	:	刘辉
学院名称	:	计算机学院
答辩委员会主席	:	牛振东
申请学位	:	硕士
学科专业	:	计算机科学与技术
学位授予单位	:	北京理工大学
论文答辩日期	:	2017/06/05

Automatic Approval Prediction for Software Enhancement Requests

Candidate Name	:	Zeeshan Ahmed Nizamani
Research Mentor	:	Prof. Liu Hui
School or Department	:	School of Computer Science and Technology
Chair, Thesis Committee	:	Prof. Niu Zhendong
Degree Applied	:	Masters
Major	:	Computer Science
Degree Awarding Institute	:	Beijing Institute of Technology
Date of Defense	:	2017/06/05

AUTOMATIC APPROVAL PREDICTION FOR SOFTWARE ENHANCEMENT REQUESTS

摘要

随着时间的推移，软件应用会出现大量新的需求。这些需求经常以增强性需求（enhancement report）的形式随其他缺陷报告一起提交到缺陷跟踪系统。这些需求报告通常需要由开发人员手动检查和确认，需要耗费巨大的时间和精力。为此，本文提出的一种基于朴素贝叶斯的自动化的预测技术以预测给定的增强性需求是否会被批准。自动化预测技术的价值包括两个方面。首先，通过该算法，开发人员可根据时间的充裕程度对改进提案进行分级，从而剔除大量低质量的不太可能被批准的需求；其次，需求报告的申请人也可以根据预测结果提前修改提案，从而提高获批的可能性。我们从 Bugzilla 获取开源软件应用程序的增强性需求报告并进行评估。在评估过程中，对每个报告进行预处理，并表示为一个向量。将该向量和增强性需求报告的批准状态作为训练集，训练一个基于贝叶斯的分类器。最后，用该分类器来预测新的增强性需求报告是否能够通过批准。本文对不同的机器学习算法的性能进行比较，包括多项式朴素贝叶斯、支持向量机、随机森林和逻辑回归分类器。也将这些算法与神经网络（多层神经网络、深层置信网络）和深度学习算法进行比较。结果表明，多项式朴素贝叶斯分类器对给定数据集下具有较高的置信度。我们在 35 个开源应用程序的 4 万份增强性去修报告上进行 10 折交叉验证，结果表明其平均准确度高到 89.25%。

关键词: 软件需求、预测、机器学习、文档分类

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 a time consuming and tedious task. Consequently, an approach that can automatically predict whether a new enhancement will be approved is beneficial for both the developers and the enhancement suggesters. The benefit of such approach is two-fold. First, 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. Second, reporters of the enhancements may revise the reports in advance to improve the chance of approval. 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 applied different feature vector modeling approaches for textual data modeling to evaluate the performance of machine learning algorithms, including multinomial naive Bayes, support vector machine, random forests and logistic regression classifiers. These algorithms are compared with neural networks and deep learning algorithms including multi-layer neural networks and deep belief network, and it turns out that the multinomial naive Bayes classifier yields the highest accuracy with the given dataset. The proposed approach is evaluated with 40,000 enhancement reports from 35 open source applications. The results of ten-fold cross validation suggest that the average accuracy is up to 89.25%.

Keywords: Software Enhancements, Approval Prediction, Machine Learning, Document Classification

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

This chapter introduces the enhancement type software issue reports and the problem of predicting the resolution status for the new reports. Section 1.1 briefly presents the software enhancement reports and their classification. Section 1.2 presents the proposed machine learning based automated classification approach for the reports. Section 1.3 briefs the evaluation and contributions.

1.1 Software Feature Enhancements

New feature enhancements to software applications are inevitable. A software needs to adapt to an ever changing environment and fulfill user requirements, and these requirements change over time [9, 38]. Hence new feature enhancements become necessary for the success of the software due to evolving user requirements and changing technologies [40]. A non-trivial software, therefore receives a number of suggestions about adding new feature enhancements and improving the existing ones.

Non-trivial software applications planned for use in different domains can usually not be completely specified in advance in terms of their functional and non-functional requirements [31]. Hence an application cannot be implemented once and for all. The subsequent maintenance and enhancement work of a software is a continual process driven by its deployment outcomes, the advances and growth in the application domain, adaptation to changes in the external world and the users and stakeholders feedback. The new enhancements are incorporated in a software project maintenance patches and new version releases. Such enhancements help keep the software relevant and may encompass different purposes such as:

- Introduce new feature extensions in the application.
- Improve the application systems' performance, processing efficiency, and maintainability.
- Adapt the software system to changes for new data and processing environment.

Large software projects generally receive a significant number of issue reports [37]. Such software applications generally use an issue tracking system or bug repository for filing and managing different types of issues or bug reports. The feature requests are special kind of issue reports called enhancements. Such reports for a project suggest the developers and maintainers, a change or upgrade that raises a software's capabilities beyond the original specifications. To request an enhancement in a software project, an enhancement suggester writes a report to describe the enhancement. In issue tracking systems, 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 account for 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, 1,857 reports were enhancements, which account for 18.57% of all types of issue reports. This data 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. Rest of the types of bug severities for the issues filed for Thunderbird during the same period are represented by these number of reports: blocker:127, critical:840, major:1202, normal:4757, minor:888, trivial:323.

1.2 Background

After releasing a software application, the developers have to deal with a myriad of bug and enhancement requests [6]. Managing such reports among the developers, and making sure that each of these bug reports or feature enhancement requests is properly handled is a effort taking activity. Issue or bug reporting and management using an issue tracking system is a standard practice in software development. However, many bugs reported for an application are low quality, leading to rejection of such reports.

The problem of the low quality reports which may lead to rejection of these reports, often arises because the enhancement reporters of the software are not fully aware of the history, environment, functionality and technical limitations associated with the software. This leads to the mismatch between what the developers expect and what the reporters provide [72]. Consequently, the reporters end up proposing poorly written enhancement reports and the software maintainer ends up evaluating many such reports that are not fixable thus wasting his time and efforts. Anvik et al. [3] analyzed general bug reports of Firefox application between May 2003 and August 2005 and found that 56% reports of Firefox are not fixed, while out of remaining 44%, only 11% are fixed leaving 33% open bug reports. This suggests that a majority of the issue reports are not fixed. Not all of the enhancement reports will be finally approved. The enhancement reports dataset in our dataset also has a similar statistics with most of the reports not approved. By analyzing the enhancement reports from the 35 applications we find that around 75% of the reports are not approved.

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 response 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 much more quickly.

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.

Implementation of a new software enhancement may potentially bring in new bugs. Some modules in the software are more risky for bugs than the others depending on the enhancement. Relationship between the functional enhancement activity and the resulting distribution of software defects has been exploited to develop a model to identify high-risk program modules [28].

Machine learning based approaches have been successfully applied in classifying different kinds of software documents. Quite a few approaches have been developed for software bugs classification [18, 20, 44]. Approaches have been applied to solve the problem of predicting issue reports as bug or non-bug [48] and duplicate bug report classification [7, 32, 50] using the machine learning algorithms. The technique for duplicate issue detection using the similarity measures of word frequency has shown effective performance [29]. These solutions save the developer time and improve the development efficiency. The approaches to solve these problems have shown high accuracy in predicting bugs severity, bug or non-bug report classification and duplicate bug report identification.

The existing applications of machine learning algorithms on software documents suggest that it is potentially practical to apply such algorithms to predict the approval of software enhancement reports. However, to the best of our knowledge, there is no existing approach specifically designed to predict the approval of the enhancement reports. To this end, in this work, we propose an automatic approach to predict the software enhancement reports' approval.

A number of important studies and approaches to triage and automate tasks specifically for the 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 [48] support practicality of these algorithms to predict the approval of software enhancement reports. Wang et al. [57] 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 [29]. 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.

1.3 Approval Prediction Approach

Since a large of enhancement reports filed in large software applications consume a lot of valuable time of developers, we propose a supervised machine learning based approach to automatically predict the approval of software enhancement requests. The enhancement reports are acquired from an issue tracking system. Such reports are preprocessed to lower-case the reports' description text, remove non-dictionary words and punctuations, 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.

The approach is evaluated with 40,000 enhancement reports from 35 open-source software applications. The results of ten-fold cross validation suggest that the approach is accurate. The accuracy of the classifier evaluation is up to 91.15%, and the average accuracy is up to 89.25%. 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 work include the following:

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

Rest of the thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 presents the proposed approach. Chapter 4 covers the experimental setup, evaluation and results. Finally, chapter 5 concludes the thesis and discusses future work in this direction.

1.4 Summary

This chapter presents the motivation for the problem of software enhancement reports approval and some background on software issue reports. The chapter further outlines the contributions of the this work towards the research, an automated approach predicting the approval or rejection of software enhancement reports and evaluation of the approach with open-source applications data.

Chapter 2
SOFTWARE ISSUE REPORTS CLASSIFICATION

This chapter covers some of the important related work in application of machine learning in software bug reports handling. Some of the important related applications of machine learning classifiers are covered in the first part. Application of machine learning in automation of software issue reports handling are discussed in section 2.2. The following subsection covers the detection of duplicate issue reports detection.

2.1 Machine Learning Based Classifiers

For text based document classification problem where manually categorized history data is available, a range of supervised machine learning classification models can be used [35, 39]. Some of the popular classifiers include decision tree, support vector machine, naive Bayes and neural networks [26, 35]. 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 top ten text classification algorithms [22, 60], which is a probabilistic learning model. The classifier assumes all the features are fully independent in a given class [56]. 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 [22]. 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 [47].

Decision tree classifier is a simple non-parametric supervised learning algorithm which generates a model (decision tree) to predict the class of the new document by learning simple decision rules from the data features. Each non-leaf node in the decision tree is a condition over a feature and each branch represents the outcome of the condition. A leaf node is a class label which is the final class predicted by the classification model.

A binary text classifier assigns one of the two classes to each text document. Some of the applications of binary classifiers include the Spam content filtering for emails [17,45,69], SMS [13] and social media [25]. Moraes et al. [34] applied the binary classifiers in sentiment detection to classify product reviews as positive or negative. The classifiers have also been successfully applied to detect the inappropriate contents by search engines [12]. 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.

2.2 Automated Handling of Software Documents

Bug reports are an important part of software development and maintenance life-cycle. The reported bugs are triaged to classify their severity, approval status, priority, developer assignment and other such tasks. A misclassification of the reports in these tasks incurs both development time and costs. The reports are usually manually examined for such classification, which is often time consuming and tedious for the developers.

Machine learning based classifiers have been successfully applied to different kinds of software documents, e.g., bug reports. The problem of bug report misclassification was identified by Antoniol et al. [2] to distinguish 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%. The machine learning techniques have been applied to predict the severity of issue reports on open-source projects [27,43]. The goal was to classify the severe and non-severe bugs. The authors 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 for classifying the bug reports. Pingclasai et al. [39] 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. Their results suggested that the naive Bayes classifier produced the most accurate and efficient classification model.

Zhou et al. [71] proposed a machine learning based approach that combines text mining and data mining techniques for the bug report classification. In their two stage technique, text mining is used first to analyze the summary of bug reports. Secondly, the extracted features of bug reports are used to train the machine learning algorithm and finally data grafting techniques are used to get final classification results of the two stages. Their experiments show a better overall performance than the previous studies on the same data.

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. [70]. The authors proposed an information retrieval based method BugLocator, to rank the files using the textual similarity between the initial bug report and the source code that uses the information about similar bugs fixed before.

The bug assignment is the problem in which the issues reported requiring resolution, need to be assigned to a developer with the responsibility of resolving the issue. Automatic approaches for assignment or tossing of the bugs reports to developers have been extensively studied [4,8,23]. Anvik et al. [5] proposed a semi-automated machine learning based algorithm for the assignment of reports to relevant developer. The algorithm learns the kinds of reports 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 developers relevant 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 have been proposed by Naguib et al. [36] to assign a bug to appropriate developer. In their formulation, user activity profile for every developer is formed based on his activities including reviews, assignment and resolution that suggest the user's involvement in the project and his expertise.

The work by Lamkanfi et al. [27] aims to help developers distinguish sever bugs from non-severe to resolve them first. Their automated approach scales down multiple levels of severity into two classes; sever and non-severe to classify new bug reports into one of these two categories. In our problem similarly, we have multiple resolution classes for enhancement reports which we scale down to two; approved and reject. From the enhancement reports, we only label fixed reports as approved and others reject. Our goal is to separate and list the approved reports from rejected and hence assist the developer and enhancement suggester in predicting the likelihood of approval.

Due to limited resources availability, bug reports resolution work is often affected by their priorities. To automatically recommend priority level for a bug report, a machine learning based approach called DRONE has been proposed [54]. The system recommends a priority level based on the information including text, reporter, severity, related reports, and product. These factors treated as features are used to train a new discriminative classification model. The experimental results of this approach on Eclipse project show an improvement in F-measure of 58.61% over baseline approaches.

2.2.1 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. These similar bug reports are classified as duplicates of each other. Duplicate enhancement reports contribute noticeably in the total reports filed. Duplicate bugs consume valuable time of developers while being difficult to pinpoint when the subject application is sufficiently large sized with a huge number of issue reports. Manually going through a pile of existing reports to detect duplication is a tedious effort. 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 reported 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 [7, 21, 52]. These approaches have shown reliable performance and hence can be leveraged in the domain of the enhance reports.

Yang et al. [63] evaluated the influence of three feature selection schemes with the multinomial naive Bayes classifier to predict the severity of bug reports. The experimental results conducted on four open-source components from Eclipse and Mozilla show that feature selection schemes can affect and improve the predication performance. *TakeLab* system [46] automates semantic similarity measure of short text documents using supervised machine learning. Using the TakeLab system, Lazar et al. [29] presented an improved method to detect duplicate bug reports that uses textual similarity features.

Sun et al. [50] 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. [32] for duplicate bug detection. The approach achieves improvements between 2.79% to 28.97% in evaluation. Tian et al. [53] 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. [16] proposed profile information of the bug reporter to improve the accuracy of the existing approaches in detecting duplicate bugs.

DupFinder et al. [52] is an integrated duplicate bug report detection tool that is 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.

2.3 Summary

This chapter covers some of the prominent research studies on automated handling of software bug reports. The major works include application of machine learning in bug reports classification, automated triaging of the bugs reported and the duplicate issue reports detection. These machine learning based approaches have shown reliable results and thus are helpful in software development process.

Chapter 3
APPROACH

This chapter proposes the approach devised for automated software enhancement reports classification into either approved or rejected class. Section 3.1 presents the problem definition in terms of enhancement report and classification function that assigns corresponding class to the report. Section 3.2 shows the overall approach from reports data extraction and preprocessing to classification of the reports. Section 3.3 shows the enhancements data collection, organization and preprocessing steps. Section 3.4 shows the way the preprocessed data is converted into feature vector form for machine learning algorithms to use. Finally Section 3.5 shows building a naive Bayes classification model on the training data and use it to classify new reports.

3.1 Problem Definition

The proposed approach in this paper predicts whether a new enhancement report will be approved or rejected. We define the problem of 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 new enhancement report r into a classification category c, is given by

$$c = f(r); \quad c \in \{approve, reject\}, \quad r \in R \tag{3.1}$$

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 the ones that 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.

3.2 Overview

Overview of the approach is presented in Figure 3.1. The proposed approach to predict the approval of enhancement report 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 corpus is preprocessed to partition the reports into two classes (approved and rejected). Each report is converted into a feature vector form to input in a classifier.
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.

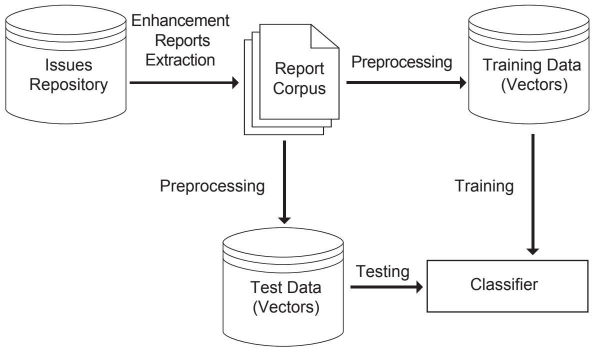
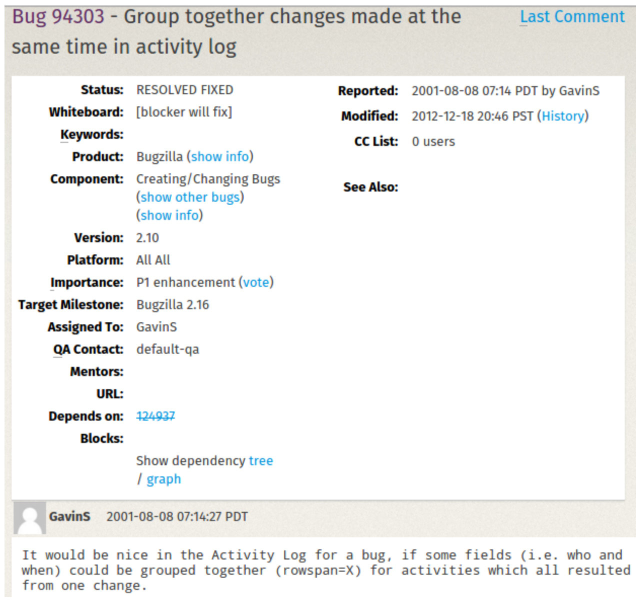


Figure 3.1: Supervised machine learning based prediction approach



Bug 94303 - Group together changes made at the same time in activity log
Last Comment

Status:	RESOLVED FIXED
Whiteboard:	[blocker will fix]
Keywords:	
Product:	Bugzilla (show info)
Component:	Creating/Changing Bugs (show other bugs) (show info)
Version:	2.10
Platform:	All All
Importance:	P1 enhancement (vote)
Target Milestone:	Bugzilla 2.16
Assigned To:	GavinS
QA Contact:	default-qa
Mentors:	
URL:	
Depends on:	~~124937~~
Blocks:	
	Show dependency tree / graph
Reported:	2001-08-08 07:14 PDT by GavinS
Modified:	2012-12-18 20:46 PST (History)
CC List:	0 users
See Also:	

GavinS 2001-08-08 07:14:27 PDT

```
It would be nice in the Activity Log for a bug, if some fields (i.e. who and
when) could be grouped together (rowspan=X) for activities which all resulted
from one change.
```

Figure 3.2: Enhancement report on Bugzilla

3.3 Data Collection and Preprocessing

The bug reports data is usually stored and managed on issue tracking systems. Issue tracking system keeps track of software application issues, allowing the users to submit issue reports for software, while facilitate the developers and maintainers to collaborate on those issue reports by making comments.

3.3.1 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 3.2 shows a graphical view of an enhancement report of an application submitted to Bugzilla. Some of the attributes of 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

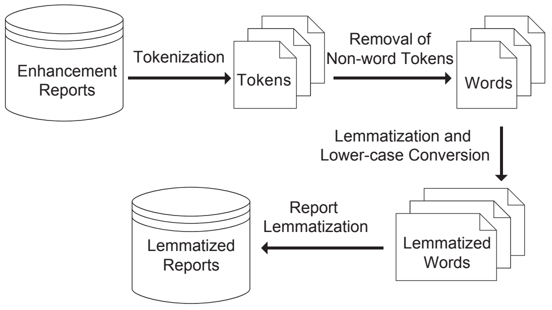


Figure 3.3: Enhancement reports preprocessing

description of the enhancement requested.

In our approach, the text features are selected from the enhancement reports' description for the approval prediction. The textual description defines what feature enhancement is requested and thus is an essential parameter for approval decision.Approaches on different bug reports classification tasks using textual features as input have shown reliable performance suggesting that these text features are effective in the issue reports classification [27, 43, 62]. Other factors, e.g., available resources, budget, and business concerns are also essential parameters of approval decision. However, it is challenging to obtain and quantify such factors from a large number of applications, which makes it difficult to figure out the quantitative relationship between the approval decision and these factors.

3.3.2 Preprocessing

Preprocessing is an important step for text classification since it improves performance of the classification approach [1]. The steps of preprocessing are summarized in Figure 3.3.

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 into lower-case 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 features 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 the classifier. A text feature selection scheme 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 [10, 59].

For preprocessing and feature selection, we used the lemmatization API to get the features from the reports' text. A report is lemmatized and returned 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 that has occurred in the reports dataset.

Text classification often involves high dimensional and sparse data features [49]. Lemmatization reduces the number of the features and thus the size of the feature vectors. The reduction of the feature vectors size improves the efficiency of the approach. Using the lemma form of words reduces the chances of a word occurring in the training set in one form, but occurring in test report in a different form, thus being treated as two different features.

3.4 Vector Space Model

The textual reports are converted into feature vector model [64], which is a compact representation that consists of all the unique features extracted from the enhancement reports. Each report in the lemmatized reports set 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,

$$v = \{c,\ f_1,\ f_2,\ \dots f_n\} \tag{3.2}$$

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

$$f_i = \begin{cases} 0, & \text{if i}^{\text{th}} \text{ feature is absent} \\ N, & \text{if i}^{\text{th}} \text{ feature appears; N>0} \end{cases} \tag{3.3}$$

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 classifier are report feature matrix where the columns are the features and rows are the report vectors [11]. We model all the reports according to the equation 3.2.

3.5 Multinomial Naive Bayes Classifier

The classification method has a significant impact on the accuracy of the text classification approach [51]. Given a set of report $(d_1, \dots, d_n)$, 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 [68].

The multinomial model is considered better and efficient classification model than the multivariate Bernoulli model [56]. Multinomial naive Bayes keeps track of the frequency of words in the feature vectors representing the reports [15, 24]. For a test report d, represented by feature vector $< w_1, w_2, \dots, w_n >$, multinomial naive Bayes uses the equation below to classify the report.

$$c_{MNB}(d) = P(c) \prod_{i=1}^{n} P(w_i|c)^{f_i} \tag{3.4}$$

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,

$$P(c) = \frac{N_{doc}}{T_{doc}} \tag{3.5}$$

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

$$P(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in v} count(w, c)} \tag{3.6}$$

Murphy and Cubranic [35] used the set of text words that from the bug report's summary and description fields. Authors used term frequency of words in the text to model the text documents 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 the empirical results suggest the classifier performs optimally considering the assumption not entirely unreasonable [14].

Despite the simplified assumption of features independence, the naive Bayes classifiers are shown to have sound theoretical reasons for their competitive performance [67]. 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 [41]. Such characteristics of naive Bayes model apply to our dataset and the binary classification problem, make it optimal for our approach.

3.6 Summary

In this chapter, the enhancement prediction problem and the approach is formally defined. The enhancement reports data is acquired from issue tracking system. The reports' text is preprocessed and modeled as feature vectors. A part of feature vectors are used to train Bayes based supervised classification model to generate the classifier. The classifier thus formed is evaluated with a different part of the dataset to test the performance.

Chapter 4
EVALUATION

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 generated from the classifiers' training are applied to the test dataset to evaluate the 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 enough, the algorithm can outperform the multinomial naive Bayes classifier.

The original enhancements dataset acquired from Bugzilla is imbalanced which may affect the 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 some rejected reports. For instance, Bugzilla application had 2197 approved reports in the original dataset, and 3537 rejected. After under-sampling the reject class reports, we retained 2197 rejected reports while discarding the remaining 1340 reports for the project. The full dataset after re-sampling has total 21317 reports with 11072 rejected reports making it roughly balanced. We compare the original dataset results against re-sampled data in the Bayes based approach and other classifiers models.

4.1 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 takes to train the classifier and classify new reports, respectively?

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 because they are popular and accurate in binary classification of text documents [22, 26, 60].

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 original 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 calculate the likelihood of different words affecting the reject probability of the enhancement reports.

The processing time is an important factor in prediction since if the approach takes longer to predict report approval than the developer, it will not be very useful. The research question RQ4 measures the time factor of the approach for training and approval prediction.

4.2 Dataset

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. 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 reports' text, choosing one issue tracking system over the other hence, does not noticeably affect the results. We chose Bugzilla for data for the evaluation since it is one of the most popular issue tracking systems for open-source applications. Furthermore, a number of large open-source projects including Firefox, use Bugzilla for their issues tracking. Bugzilla also provides a REST API to easily access to bug reports of open-source applications. We hence used Bugzilla issue tracking system for the data in our approach.

4.2.1 Data Retrieval

To acquire the data, we extracted the enhancement reports of open-source software applications from Bugzilla[1]. Bugzilla is a general-purpose bug and issue tracking system. Using the Bugzilla REST API[2], we wrote a nodejs script (available online[3]) to access the enhancement reports by specifying severity level as *enhancement* in the Bugzilla REST API. We specify severity type of enhancement in the URL to distinguish the feature enhancement requests from the bug reports and thus retrieve only enhancement reports. The text of the first comment of an enhancement report has a detailed description of the report which is separately extracted using the enhancement report ID[4]. We treat and refer this detailed description as the enhancement report in our approach since this field represents the requested enhancement in detail by the suggester. In evaluation, we only used the enhancement reports that have been resolved as either fixed or rejected, discarding the open reports that are not yet decided. The approach is generalizable to the other issue tracking systems by using the appropriate *resolution* and *report description* fields, specific to the issue tracking system being used.

The *resolution* field returned in an enhancement reports from the Bugzilla API specifies whether the enhancement report is approved or rejected. In Figure 3.2, 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. As there are multiple resolutions of the reports, we reduce this multi-class resolution problem to binary classification problem by treating the report as approved if its *resolution* is fixed and categorize rejected otherwise. Note that we do not include the newly submitted reports in the dataset whose resolution is not defined since we do not know whether they will be approved or rejected.

[1]https://bugzilla.mozilla.org/, verified 26/02/2016
[2]https://bugzilla.mozilla.org/rest/bug?severity=enhancement, verified 26/02/2016
[3]https://github.com/shanniz/Bugzilla
[4]https://bugzilla.mozilla.org/rest/bug/426904/comment, verified 26/02/2016

Table 4.1: Open-source applications for evaluation

Application	Total Reports	Domain	Approved	Rejected
Bugzilla	4697	Issue tracking system	2197	2500
Calendar	1505	Desktop calendar	439	1066
Camino Graveyard	1168	Mac OS X browser	344	824
Core	7223	Web browser components	2754	4469
Core Graveyard	1026	Core components	259	767
Firefox	6793	Web browser	896	5897
MailNews Core	2050	Mail and news components	376	1674
SeaMonkey	7922	Internet application suite	883	7039
Thunderbird	3934	Email client	398	3536
Toolkit	1678	APIs	380	1298

We collected the enhancements data for the open-source applications from Bugzilla. The significance of an issue report on Bugzilla is a composite of its priority and severity levels. The priority for the issue is decided and set by the maintainers or developers who plan to work on the bug, not by the one filing the issue report. General to issue tracking systems and specific to Bugzilla, the defined 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 a 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 for a current 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 the open-source applications obtained from Bugzilla is desktop applications for Internet. The top applications with most number of enhancement reports in our dataset are listed in Table 4.1. The number of approved and rejected reports for each of the top products is summarized in the table.

4.2.2 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 fixed 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 include DUPLICATE, EXPIRED, FIXED, INCOMPLETE, INVALID, MOVED, WONTFIX, WORKSFORME. The FIXED type is the approved enhancement report, while others are not approved. 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 4.2.

Table 4.2: Number of reports for each resolution type

Resolution	Number of Reports
FIXED	10020
INVALID	2621
DUPLICATE	15784
WONTFIX	7170
WORKSFORME	2501
INCOMPLETE	610
EXPIRED	1291
MOVED	4

Table 4.3: System configuration for evaluation

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

4.3 Metrics

We use accuracy, precision and recall 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 4.3.

4.4 Process

The evaluation is carried out as follows. First, the enhancement reports (notated as ER) of the open-source projects are retrieve from Bugzilla issue tracking system and converted into feature vector form. Second, based on ER, we carry out a ten-fold cross validation. We randomly partition ER dataset into ten equally sized groups denoted as G_i ($i = 1 \ldots 10$). For the ith cross-validation, we consider all reports except for those in G_i as the corpus of training data, and use the reports in G_i as the testing data.

For the ith cross-validation, the evaluation is performed as follows:

1. First, we extract all reports $trainingData_i$ from training dataset that is the union of all groups but G_i.
$$trainingData_i = \bigcup_{j \in [1,10] \wedge j \neq i} G_j \tag{4.1}$$

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 reports 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 the actual (correct) status.

7. Finally, we compute accuracy, precision, recall and f1-score for each of the classifiers.

We do not train individual classifiers for different applications, but train a single classifier with the reports from all the applications. This is 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 precision. In other words, we make predictions based on the same resulting classifier for different applications.

4.5 Subject Applications

We have used a total of 40,000 enhancement reports out of which 10,020 are approved. Some of the typical applications of which the reports were obtained include Firefox, Thunderbird, SeaMonkey and Bugzilla itself. The issue reports of these open-source applications are publicly accessible on Bugzilla. In the rest API call to access the data, the application is specified as the *product* and enhancement as the *severity* type to access enhancement reports of each of the applications.

- *SeaMonkey:* The Internet application suite for advanced users and the developers. It contains web browser, email and newsgroup client, HTML editor, IRC chat and web development tools. SeaMonkey project has 7922 enhancement reports out of which 883 are approved and 7039 are rejected.

Table 4.4: 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%**

- *Core:* Accessibility in web development to enable increase human accessibility to websites, even people with limited abilities. Provides information on developing more accessible content. Total reports from this application are 7223 with 2754 approved reports and 4469 are.
- *Firefox:* The widely used free and open-source web browser. The browser uses the Gecko layout engine to render web pages, which implements current web standards. This project received 6793 enhancement reports, where 896 reports were approved leaving remaining 5897 reports rejected.
- *Bugzilla:* The issue tracking server software that helps manage software bugs management. From the Bugzilla project itself, we retrieved 4696 enhancement issue reports where 2197 reports were approved while 2500 reports were rejected.
- *Thunderbird:* Mozilla Thunderbird is an email and chat client application. We retrieved 3934 enhancement reports from NSS, out of which 398 reports are approved and 3536 are rejected.

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 [48,60]. The results of applying these classifiers on the enhancements dataset revealed that the multinomial naive Bayes classifier yields most accurate results on our dataset.

The naive Bayes classifier algorithm is proven effective in many applications including the binary text classification [19]. Naive Bayes algorithm achieves competitive performance, even though its basis of conditional independence assumption on which it is based, is not often true. Zhang [67] 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

Table 4.5: 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%**

evaluated on Spam email detection [66] show that the algorithm is effective in binary classification problem. These works support our application of naive Bayes for the classification and its optimal performance on our binary classification dataset. In our dataset, the classifier achieves optimal performance.

Despite it's simplicity, the C++ implementation of multinomial naive Bayes[5] classifier produced accurate prediction results on ten-fold validation test. The implementation uses multinomial event model and the maximum likelihood estimate with a Laplace smoothing technique for learning parameters. The classifier mis-classifies few reports in the ten-fold cross validation. The mis-classifications are mostly false negatives due to the imbalance in dataset.

Support Vector Machine: Support vector machine is another classifier shown effective in text classification [65]. We trained the support vector machine classifier implementation called SVM^{light}[6]. 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 4.5 presents the performance of support vector machine classifier on ten-fold cross validation. Total number of the reports in one set is 4,000. Thus each training dataset has 36,000 reports while a single test set has 4,000 reports.

The decision tree[7] 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 ten-fold 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 [2, 55, 61]. For this reason, we assessed the performance of algorithms with same evaluation metrics and ten-fold cross validation. We used the same aggregate dataset of all the 35 subject applications reports text modeled as feature

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

[6]http://svmlight.joachims.org, verified 27/05/2016

[7]https://github.com/yandongliu/learningjs, verified 20/05/2016

Table 4.6: 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%**

Table 4.7: 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%**

vectors. The results of applying these algorithms on our dataset were however, not as encouraging compared to Bayes classification model. Random forest classifier exploits ensemble learning technique. The standard implementation from *sklearn.ensemble* module for *RandomForestClassifier* was evaluated for cross validation scores. Table 4.6 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 4.7 summarizes the performance of logistic regression classifier from sklearn library. The algorithm outputs an average accuracy of 71.36%. However, it does not performs 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 on the corresponding dataset, we perform ANOVA analysis on their resulting accuracy in ten-fold evaluation (ten different accuracy

Table 4.8: ANOVA analysis on accuracy

MNB vs SVM

Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F critical**
Between Groups	0.070590962	1	0.070590962	22.22338	0.000172997	4.413873419
Within Groups	0.057175686	18	0.003176427			
Total	0.127766648	19				

MNB vs RF

Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F critical**
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				

MNB vs LR

Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F critical**
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				

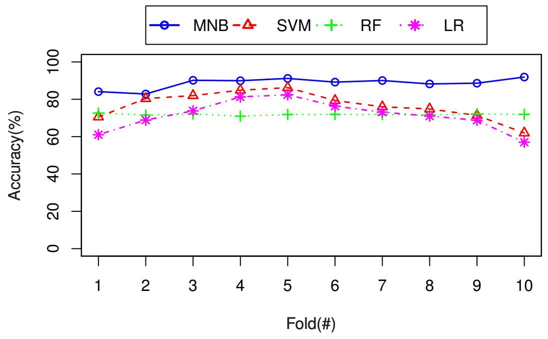


Figure 4.1: Accuracy of the approach

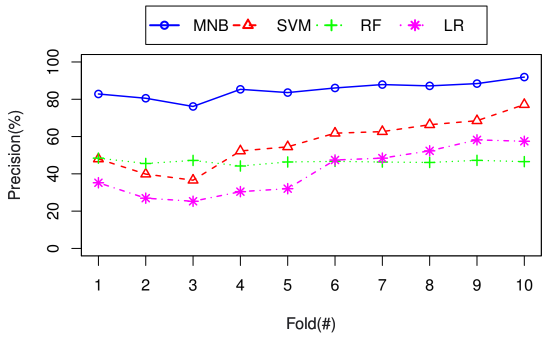


Figure 4.2: Precision of the approach

for each of the techniques). The results of ANOVA analysis are present in table 4.8. From this table, we observe that for each of the comparison present in the table, $F > F_{cric}$ and $P - value < (alpha = 0.05)$. These results suggest that the factor (different classification techniques) did cause significant difference in resulting accuracy, and the multinomial naive Bayes based approach results in best accuracy.

Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4 visually compare the performance of support vector machine, random forests, logistic regression and multinomial naive Bayes classifiers in terms of accuracy, precision, recall and f1-score performance metrics respectively. The graphs reveal that the multinomial naive Bayes implementation outperforms the compared algorithms in all performance metrics.

The distribution of accuracy over ten-fold cross validation for multinomial naive Bayes, support vector machine, random forests and logistic regression is presented in Figure 4.5. A bean-plot 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. Shape of the bean is the density, and the longer bold line represents the average accuracy of each classifier over ten-fold cross validation.

As observable in Figure 4.5, the multinomial naive Bayes classifier exhibits a high accuracy and a small deviation in the values through the different folds of ten-fold validation. Random

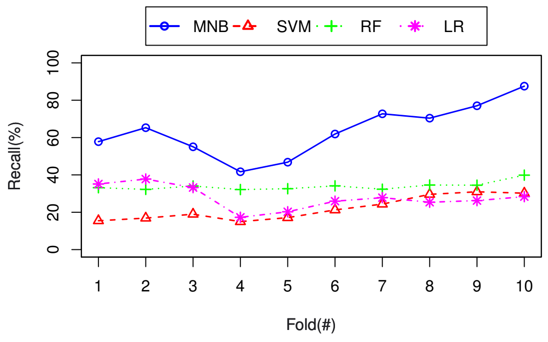


Figure 4.3: Recall of the approach

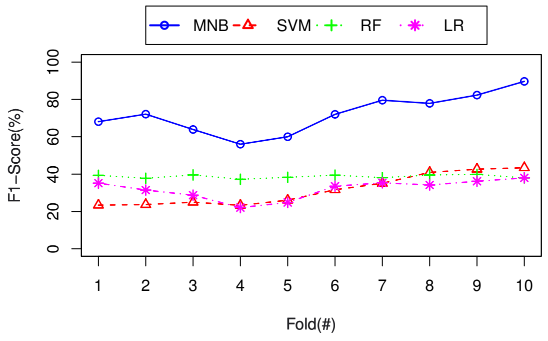


Figure 4.4: F1-Score of the approach

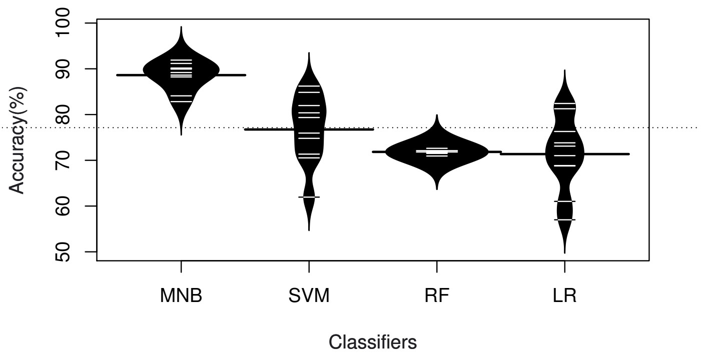


Figure 4.5: Accuracy distribution of the approach

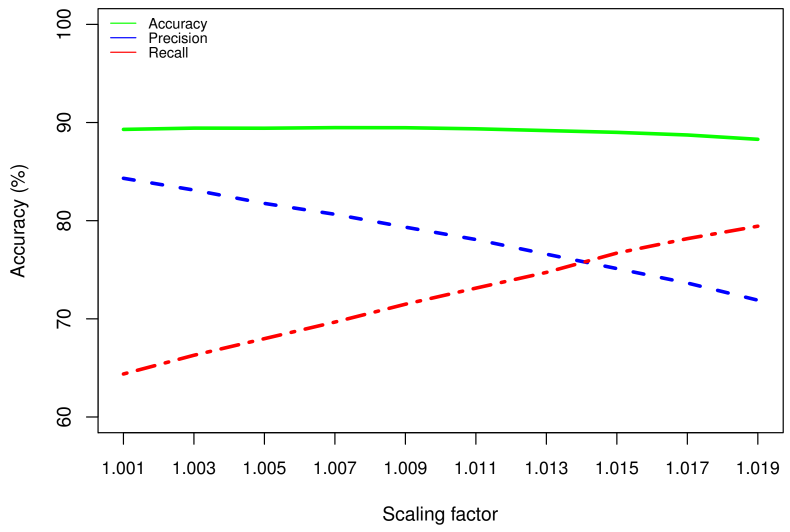


Figure 4.6: Effect of scaling the class probability on the performance

forest classifier shows consistent but lower accuracy as compared to multinomial naive Bayes. Support vector machine classifier and logistic regression does not show consistency in accuracy across different folds of ten-fold 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 ten-fold cross validation.

We conclude the preceding analysis that multinomial naive Bayes classifier outperforms other classification models in the approval prediction of enhancement reports. One of the possible reasons is that the Bayes is an effective model for text classification and the proposed approach is completely based on the text. Another possible reason is that the naive Bayes classifier is more effective for binary classification than multi-class classification [42], and the proposed approach classifies the reports into two classes only.

Neural network and deep learning based classification approaches have shown a superior performance in many applications, but they usually require a large number of training dataset. Apart from that, there are a large number of parameters that need to be optimally adjusted for such algorithms to result in higher performance. Neural network may not have outperformed the Bayes based approach on dataset due to these reasons.

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 the him, by scaling the class weight accordingly. With the scaling factor, posterior probability becomes

$$P(approve) = s * P(approve) \tag{4.2}$$

Figure 4.6 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 ten-fold 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 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)[8], a neural network algorithm. The deep learning algorithms require compact representation of the data for better performance. Le and Mikolov [30] proposed an unsupervised algorithm called *paragraph vector* that generates fixed length feature representation from text reports of varying lengths. We applied the *paragraph2vec* implementation [33] of the algorithm to convert the enhancement reports into feature vectors. Since the algorithm allows to generate the 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.

The deep belief network (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 1,000 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, we tested different sized subsets of the our original corpus to observe if the results improve with increasing dataset size. The MLP algorithm has shown consistent improvement when we applied increasing size of the dataset.

Table 4.9 shows that the validation accuracy of MLP gradually increases with the increasing dataset size. Deep Belief Networks' 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 5,000 records.

The result of extrapolation in Figure 4.7 (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. With the dataset size doubled to 280,000 reports, the extrapolation suggests that the results of the neural networks based approach will be 94.19% accurate. This trend of performance improvement shows that if the dataset size is sufficiently large,

[8]http://deeplearning.net/, verified 10/08/2016

Table 4.9: Multi-layer neural network performance improvement with respect to dataset size

Dataset Size	Error Rate
10000	28.50%
15000	25.46%
20000	25.45%
25000	22.76%
30000	20.56%
35000	20.45%
40000	18.92%

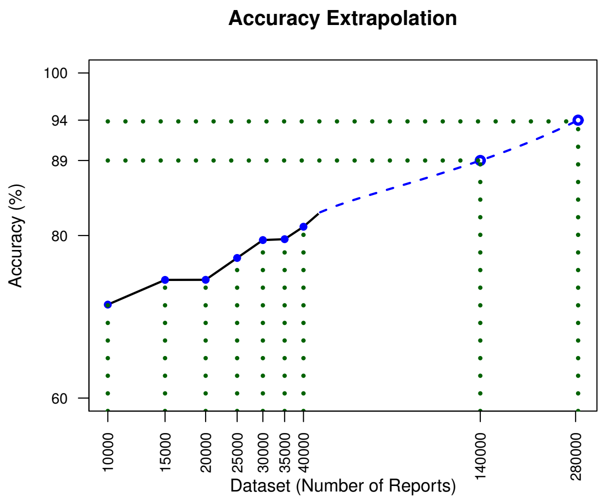


Figure 4.7: Validation accuracy extrapolation for neural network based approach

Table 4.10: Multinomial naive Bayes ten-fold 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%**

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

There are different ways of re-sampling including 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 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. Similar 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 to approved reports. In ten-fold cross validation of the classifiers, the training fold was subjected to under-sampling with balanced number of reports in each fold, while the testing fold was used unchanged. For instance, in our 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. Thus each training fold contained around double the number of approve reports, half approved, half reject. The testing fold dataset of 4,000 reports was used unchanged to mimic the real testing condition of more reject reports in the issue tracking system.

The ten-fold cross validation results of multinomial naive Bayes based approach with re-sampled reports dataset are presented in Table 4.10. Each row shows result of the corresponding fold number in ten-fold 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

Table 4.11: Average 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)

precision, recall and f1-score being 58.94%, 89.68% and 70.51% respectively.

Table 4.11 summarizes the average ten-fold cross validation performance for 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 imbalanced dataset. The results suggest that under-sampling the enhancements corpus does not improve accuracy of the approval prediction. In many cases, re-sampling is a useful technique to deal with the over-fitting problem due to skewed data. However, in our case, it does not show improvement. One possible reason is that the chosen re-sampling technique is not suitable in our case. Replacing it with other re-sampling techniques may improve the accuracy.

4.6.3 RQ3: Positive and negative words

Lamkanfi et al. [26] investigated the problem of predicting the bug reports as *non–severe* or *severe* using naive Bayes classifier. The authors calculated the 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.

Table 4.12 presents the probabilities of some of the words in the approved reports. Words *accordance*, *exponent* and *patch* for instance, have a positive or constructive sense and are mostly associated with the feature enhancements to software applications. So such keywords support the argument of the enhancement and improve the quality of the report. The positive words should be used to clearly specify the enhancement with constructive suggestions.

Words *arena*, *slot* and *patch* can clearly specify which part of the software should be improved. The enhancement reporter has clear idea of the software and the enhancement which makes it highly possible that he suggests high quality enhancement. So it can be asserted that such words improve the quality of the enhancement report, giving the developer a better understanding of the proposed enhancement, thus increasing the likelihood of enhancement approval.

We calculated 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. TF/IDF is a useful technique to measure the significance of the words in a document. We sorted the 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.

Table 4.12: Some words with highest approve probability $P_{approve}$

Word	Likelihood of approval
accordance	94.06%
bone	93.75%
exponent	88.23%
computation	83.33%
arena	80.00%
sweep	80.00%
slot	79.41%
deprecate	75.00%
patch	69.82%
module	54.87%

Table 4.13: Some words with highest reject probability P_{reject}

Word	Likelihood of rejection
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%
suspect	76.19%

Equation 4.3 depicts the formula to calculate the reject likelihood of word.

$$P(w_i, reject) = \frac{count(d_{reject}, w_i)}{count(d_{total}, w_i)} \tag{4.3}$$

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. The approve probability is similarly calculated as:

$$P(w_i, approve) = \frac{count(d_{approve}, w_i)}{count(d_{total}, w_i)} \tag{4.4}$$

Where $P(w_i, approve)$ is the approve 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 4.13 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 enhance report clearly, let alone providing constructive

Table 4.14: 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

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.

Although such words may have negative effect on approval of the report, they do not however always lead to rejection of the enhancement. Many of the reports containing such words are not approved, but sometime reports containing these words are approved. If needed, these words should be used carefully in more constructive ways to reduce the chances of rejection. Thus a report is not always rejected by just having one or two of such words, but frequent use of many of these words may lead to rejection.

Generally the words contributing towards rejection have higher likelihood probability than the words of approval likelihood. This is due to the fact that most of the enhancement reports of the applications obtained from Bugzilla, were rejected.

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 classifiers. 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 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, so we only estimate the process time. It is important to note that the average time for lemmatization is the combination of communication time (lemmatization request to the server and the response time) and the time of the 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. In each trial, we used the same sized dataset of the reports. The average time is calculated in seconds scale on our system with training dataset of 36,000 reports and test dataset of 4,000 reports. The results are depicted in the Table 4.14.

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

In case of the training to generate the classifier, multinomial naive Bayes algorithm is most 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

$$O(|D|L_d + |C||V|)) \tag{4.5}$$

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

$$O(|C|L_t) \tag{4.6}$$

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. The naive Bayes classifier is time efficient that learns with one pass of counting over the data and tests linearly in the number of attributes.

4.7 Threats to Validity

4.7.1 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. This is possible due to human error. Hence our training and testing data may have a few inaccuracies. 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 performance on more data and it is possible that the actual performance may not be as good.

4.7.2 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. This may lead to limitation and the problem for lemmatization programs. 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, the learning rate, activation function, 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. As we treat a report as approved if its resolution is fixed, and reject otherwise while different types of reject classes are not considered. An important resolution for enhancements is duplicate report. Although a requested feature may be good enough but be duplicate of another enhancement, thus be rejected. We do not specifically include this problem in our approach, since a number of duplicate detection techniques have been proposed, and one of such approaches can first be used to rule out the duplicate issue.

4.7.3 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 classifier 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 the re-sampling technique used. Since there are different data balancing techniques available, the performance can 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 generating or fabricating new data. The dataset size is however, reduced with under-sampling that can affect the performance. A different re-sampling method may have better results.

The third threat is that the evaluation is performed on the applications from Bugzilla issue tracking system. Unlike Jira, Bugzilla is a well known open-source system that provides API to access the issue reports data. We thus used Bugzilla to extract the enhancement reports. 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 open-source applications. The domain of these applications is mostly the Internet and desktop applications. Therefore the approach may possibly not perform well on other domains like smart-phone applications. The evaluation on more applications in different domains should be conducted to reinforce the conclusions.

4.8 Summary

In this chapter, the evaluation process for the proposed approach is presented. The approach was evaluated on 35 open-source projects from Bugzilla based on accuracy, precision, recall and f1-score. Bayes based approach is further compared with the other classifiers with and without re-sampling of the dataset. The negative words are discussed that may not be convey constructive suggestions and thus affect the approval of an enhancement reports. The training and testing time for the difference classification methods used in the evaluation show that the Bayes based approach is both effective and efficient. Finally some of the threats to validity are discussed that may affect the performance of the approach.

Chapter 5
CONCLUSION AND FUTURE WORK

In this study, we propose a supervised machine learning based approach to predict whether a software enhancement request would be approved or rejected. The approach builds a classifier from the history data of open-source projects from Bugzilla. The approach preprocesses the enhancement requests description, converts into feature vectors and uses the resolution status of the corresponding enhancement report feature vectors to train a Bayes based classifier. A test dataset without resolution status or class is fed in the classifier to test it's approval prediction performance. The valuation of the approach resulted a prediction accuracy of 89.25% over ten-fold cross validation.

A majority of enhancement reports of the software applications on Bugzilla are not approved and the manual evaluation of these reports by the developer wastes much productive time. The approach in this paper shows that it is possible to automatically classify the enhancement reports as approved or rejected.

To train a supervised classifier for issue handling tasks like approval prediction or triage, a number of labeled issue reports need to be collected. The history data may not always be accurate that may affect the classification model. The classified bug reports data can be insufficient or even incorrect as some of bug reports may not be clearly described. Moreover, a developer may mistakenly assign incorrect labels to the bug reports. If there are a significant number of such incorrectly labeled data, it would affect the quality of the training data and hence the resulting classification model. We chose to extract the confirmed issue reports which have already been resolved and closed after confirmation.

A feature enhancement may be suggested which is reasonable and likely to be approved that uses constructive suggestions, but a similar enhancement has been approved just in time of the suggestion or the suggester missed similar enhancement request previously filed, leading this report as duplicate. To this end, one of the numerous duplicate issue report detection technique can be applied to rule out a similar request before filing a new enhancement report. Extensive studies have been done to retrieve similar or duplicate issue reports.

We evaluated the performance of the approach on enhancement reports of open-source applications acquired from Bugzilla. We conclude that the prediction accuracy of the approach is optimal given sufficient training data of two classes of enhancement reports, with average precision and recall 84.99% and 63.26% respectively.

5.1 Study Limitations

A limitation of the proposed approach is that it 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, incorporating such factor may improve the 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. Actually, the enhancement reports have a number of possible status like approved, rejected, duplicate and invalid. We take all reports that are not approved as non-approved and make no further classifications. We make such a binary classification instead of multi-class classification because of the following reasons:
First, the major purpose of the approach is to recommend those reports that are likely to be ap-

proved and ignore others. Consequently, classifying the reports into approved and non-approved is enough for the task.
Second, reducing the number of the classes helps improve the classification accuracy.

5.2 Conclusion

We observed relatively low performance in certain test cases and observe that some words affect the performance. Particularly important to the performance is a weak point in the multinomial naive Bayes classifier that one or more words appearing in only one class in the training dataset but occurring in the other class in test dataset reduce the overall likelihood of the actual class. A smoothing factor is applied to minimize the effect of such words. Furthermore, the imbalanced nature of the dataset also affects the classifier which produces biased results towards more occurring reject class.

This study enables to implement an efficient automated enhancement classification method. The approach contributes to the current research trends of automating the software issue reports classification. If prediction is rejected, the report may have used the words which that did not express the enhancement requirement clearly, reducing the chances of approval. The reporter may change the keywords and improve the report with constructive suggestions to make it more likely to be approved. Our prediction approach can therefore help the reporter before the report is actually submitted so there will be more chances of approval. Furthermore, it can save the time of the developer or maintainer by getting more likely to be approved enhancement reports.

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

Apart from the application in Bugzilla, the study can be applied on the other issue tracking systems that use a similar process of issues cycle to track the issues. There may be minor differences in the names of different attributes of an enhancement report, which can be taken into account while using the approach with other issue tracking systems.

The proposed approach limited to classifying the reports into two classes. The multinomial naive Bayes classifier used in the approach has shown significantly accurate results. Since the approach is only optimal in classifying reports into two classes, the problem of predicting the exact resolution status of the report rejection is still unsolved. The rejected reports on a typical issue tracking system have many values of resolution for instance *invalid* or *expired*. A more advanced approach in the enhancement report prediction can predict the resolution to cover other resolution types. The approach may be extended to predict duplicate enhancement report by measuring the similarity to the history reports, using additional field of the product name. An advanced approach may include the features or words suggestions for improving the enhancement report so as to increase chances of the report approval.

Appendix A SUBJECT APPLICATIONS

A.1 Open-source applications

The 35 subject application products for which the enhancement reports were extracted from Bugzilla are shown in Table A.1.

Table A.1: The open-source projects data used in evaluation

Application	Total Reports	Approved	Rejected
SeaMonkey	7922	883	7039
Core	7223	2754	4469
Firefox	6793	896	5897
Bugzilla	4696	2197	2500
Thunderbird	3934	398	3536
MailNews Core	2050	376	1674
Toolkit	1678	380	1298
Calendar	1505	439	1066
Camino Graveyard	1168	344	824
Core Graveyard	1026	259	767
NSS	625	502	123
bugzilla.mozilla.org	500	264	236
Other Applications	472	284	188
mozilla.org	359	187	172
www.mozilla.org	276	116	160
Webtools	217	142	75
Webtools Graveyard	209	85	124
mozilla.org Graveyard	159	89	70
Mozilla Localizations	151	71	80
NSPR	108	79	29
Developer Documentation	102	74	28
Rhino	98	85	13
Marketing	76	41	35
Tech Evangelism Graveyard	51	12	39
CCK	40	19	21
Documentation	38	13	25
Directory	37	26	11
JSS	25	22	3
Grendel	21	9	12
Mozilla Localizations Graveyard	19	17	2
MailNews Core Graveyard	17	3	14
Tech Evangelism	14	3	11
MozillaClassic	10	2	8
Derivatives	5	1	4
addons.mozilla.org	1	0	1

Appendix B IMPLEMENTATION

The source code for the approach and preprocessing is written in Python and NodeJS. The first Section B.1 details how the enhancement reports of the open source are acquired from Bugzilla issue tracking system using its REST API. The second Section B.2 lists implementation of the reports preprocessing steps. Section B.3 shows listing for modeling the reports into Term Frequency based feature vector model.

B.1 Enhancement Reports Data

The raw reports of the open source applications are acquired from Bugzilla issue tracking system. The reports are acquired in two phases; in the first phase, the enhancement reports are extracted from Bugzilla API. In the second phase, using the report id, the enhancements detailed description text is obtained using the API.

All the reports are processed the same way without considering the specifics of the corresponding software application of the report, for instance the source code language, the type of application, and other parameters are not put in the equation. The mean number of words in the enhancement reports of all the projects are 30. The average number of words for the reports summay is 6. Some of the important components returns by Bugzilla include the followings: $id, url, summary, platform, priority, depends\_on, dupe\_of, is\_open, resolution, severity,$ $product$

B.1.1 Reports Acquisition

```
var rest = require('restler');
var fs = require('fs');

var save_to_file = function(reports_data){
  fs.writeFile("./bugsData.txt", JSON.stringify(reports_data), function(err) {
    if(err) {
      console.log(err);
      return -1;
    }
  });
}

var get_product_enhancements = function(product = 'Firefox'){
  var URL = 'https://bugzilla.mozilla.org/rest/bug?severity=enhancement&
      product='+product+'&include_fields=id,severity,summary,status,resolution
      ,product';
  console.log("Retrieving reports from Bugzilla for " + product);
  rest.get(URL).on('complete', function(data) {
    console.log('Retrieved data from Bugzilla server for ' + product );
    save_to_file(data);
  });
}

var get_all_enhancements = function(){
  var URL = 'https://bugzilla.mozilla.org/rest/bug?severity=enhancement&
      include_fields=id,severity,summary,status,resolution,product';
  console.log("Retrieving reports from Bugzilla for all the projects");
  rest.get(URL).on('complete', function(data) {
    save_to_file(reports_data);
  });
}

var get_report_description = function(issue_id = 155482) {
  var URL = 'https://bugzilla.mozilla.org/rest/bug/'+issue_id+'/comment';
  rest.get(URL).on('complete', function(data) {
    console.log("Retrieved desciption for the enhancement with id = " +
        issue_id);
  });
}
```

This python code fetches the enhancement reports for the specified list of open-source products from Bugzilla. It uses the REST API from Bugzilla specifying the severity field as enhancement. The code then loops through all the subject applications and fetches enhancements for each of the application.

The next step is to acquire the description of each enhancement report. The listing below uses the id of each enhancement filed to fetch its detailed description.

```

URL = 'https://bugzilla.mozilla.org/rest/bug/'

def fetchEnhancementIds():
  select_query ="SELECT id FROM Enhancements.bug_details WHERE isnull(comment)
      AND resolution!=\"\""
  enhanses = db.query(select_query)
  for enh in enhanses:
    ids.append(   (enh['id']))

fetchEnhancementIds()

def insertComments(ids):
  for eid in ids:
    r = requests.get(URL+eid+"/comment")
    jObj = json.loads(r.text)
    comment =    ( re.sub(r'[^\x00-\x7F]+',' ',  jObj['bugs'][eid]['comments'
        ][0]['text'] )).replace('\'','').replace('\"','')

query = "UPDATE Enhancements.bug_details SET comment='"+json.dumps(comment)+"'
    WHERE id=" + eid
dbResult = db.insert(query)
if dbResult == -1:
  errorEnhancementsComments.append(eid)
```

B.2 Reports Data Preprocessing

The raw reports are preprocessed to remove non-word tokens from their text. The reports are further lemmatized which results in dimension reduction. The code below makes use of an online lemmatization API to lemmatize the enhancement reports text.

```
import nltk, re, unirest, json
from nltk.stem import WordNetLemmatizer
from database import *
from utilities import *

lemmatizer = WordNetLemmatizer()
errorLemmaComments = []

db = Database()
url="https://twinword-lemmatizer1.p.mashape.com/extract/"

def insertFeatures(words):
  print 'now inserting lemma words - ' +    (   (words))
  for word in words:
  query = "INSERT INTO features (word) VALUES(\'"+ word +"\')"
  dbResult = db.insert(query)

def lemmatizeREST(txt):
  args =   {
    'X-Mashape-Key': '58kqT228vBmshu9OhloHiw2YmiMXp1YSZryjsnLAlqOUAiVetU',
    'Content-Type': 'application/x-www-form-urlencoded',
    'Accept': 'application/json'
  }
  comment_lemma = ""
  try:
    response = unirest.post(url, headers=args, params={'text': txt})
    if         (response.body, "lemma"):
      jObj = json.dumps(response.body['lemma'], skipkeys=True)
      json_ob=json.loads( jObj)
      for k in json_ob:
        comment_lemma += k + " "
    else:
      comment_lemma = re.sub('[0-9!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~]',' ', txt.
         lower())
  except IOError as ioerr:
    print ioerr

  return comment_lemma

def lemmatize(   ):
  lemmaWords = ''
  words = nltk.word_tokenize(  re.sub('[0-9!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~]',
      ' ',     .lower() ) )
  for word in words:
    lemmaWords += lemmatizer.lemmatize(word, pos='v') + ' '
return lemmaWords

def fetchComments():
  select_query ="SELECT id, comment, resolution FROM Enhancements.bug_details;
      "
  enhanses = db.query(select_query)
  comment = ""
  lemmaComment = ""
  index = 1
  for enh in enhanses:
    comments =     (enh['comment']).split("Additional Details :")
    if    (comments) > 1:
      comment = comments[1].replace("\n",' ')
    else:
      comment = comments[0].replace("\n",' ')

    print '\n\nlematizing report '  +    (enh['id'])
    lemmaComment = lemmatizeREST(comment)
    index+=1;

fetchComments()
```

B.3 Feature Vector Modeling - TF Vector Generation

The feature vector models are used to represent the textual enhancement reports.

```

var calculateVector = function(resolution, status, wordsArr){
  var vector=status;
  var wordTF={};
  var found = false, isFirst = true, isLast = false;
  var featureCount = 0;
  for (var i = 0; i < allWords.length; i++) {
    for (var arrI = 0; arrI< wordsArr.length; arrI++) {
      allWords[wordsArr[arrI]].probability_difference);
      if(wordsArr[arrI] == allWords[i].word){
        featureCount++;
        if(!wordTF[(i+1)+""]){
          wordTF[(i+1)+""]=1;
        }
      else{
        wordTF[(i+1)+""]+=1;
      }

    }
  }

  for(var key in wordTF) {
    vector+=" " + key + ":"+wordTF[key]+".0";
  }

  return vector;
}
```

Appendix C PERFORMANCE EVALUATION

C.1 Performance Test Code

The Python code below uses the test file class labels and compares it with the output labels predicted by the classifiers to evaluate accuracy, precision and recall measures. The code assumes ten-fold cross validation for evaluation and thus ten equal sized test and prediction output files.

```
testfilePath = '/path/to/test/files/'
predictionfilePath = 'path/to/prediction/files/'
tTP=0; tTN=0; tFP=0; tFN=0;

def getPredDiff(test, predict):
  tp=0; tn=0; fp=0; fn=0;
  for i in      (   (test)):
    if test[i][0] == '1' and test[i][0] == predict[i][0]:
      tp+=1
    if test[i][0] == '2' and test[i][0] == predict[i][0]:
      tn+=1
    if test[i][0] == '1' and test[i][0] != predict[i][0]:
      fp+=1
    if test[i][0] == '2' and test[i][0] != predict[i][0]:
      fn+=1

  return (tp, tn, fp, fn)

def getPerfParams(tp, tn, fp, fn):
  accuracy=     (tp+tn)/(tp+fp+tn+fn)
  precision=     (tp)/(tp+fp)
  recall=     (tp)/(tp+fn)
  return (accuracy, precision, recall)

for fold in      (1, 10):
  files = [testfilePath+"/testset"+    (fold)+".samp", predictionfilePath+"/
     nbresult"+    (fold)+".out"];
  testf =     (files[0], 'r')
  predictf =     (files[1], 'r')
  test =     (files[0], 'r').readlines()
  predict =     (files[1], 'r').readlines()
  testf.close()
  predictf.close()

  print '\textbackslash\{\}n\textbackslash\{\}nfold = ' +     (fold) + '\
     textbackslash\{\}ntp, tn, fp, fn'
  tp, tn, fp, fn = getPredDiff(test, predict)
  print(tp, tn, fp, fn )
  print '\textbackslash\{\}nAccuracy, Precision, Recall'
  print (getPerfParams(tp, tn, fp, fn))
  tTP+=tp; tTN+=tn; tFP+=fp; tFN+=fn;

print ('\textbackslash\{\}n Average Accuracy, Precision, Recall')
print (getPerfParams(tTP, tTN, tFP, tFN))
```

Bibliography

[1] S Gunal AK Uysal. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1):104–112, 2014.

[2] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 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*, page 23. ACM, 2008.

[3] J Anvik, L Hiew, and GC Murphy. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 35–39. ACM, 2005.

[4] John Anvik. Automating bug report assignment. In *Proceedings of the 28th international conference on Software engineering*, pages 937–940. ACM, 2006.

[5] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[6] Anand Balasubramanian. Methods and systems for managing update requests for a deployed software application, August 26 2014. US Patent 8,819,658.

[7] S Banerjee, B Cukic, and D Adjeroh. Automated duplicate bug report classification using subsequence matching. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 74–81. IEEE, 2012.

[8] Pamela Bhattacharya, Iulian Neamtiu, and Christian R Shelton. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software*, 85(10):2275–2292, 2012.

[9] Ned Chapin, Joanne E Hale, Khaled Md Khan, Juan F Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.

[10] J Chen, H Huang, S Tian, and Y Qu. Feature selection for text classification with naive bayes. *Expert Systems with Applications*, 15:2160–2164, 2011.

[11] Z Chen and K Lü. A preprocess algorithm of filtering irrelevant information based on the minimum class difference. *Knowledge-Based Systems*, 19(6):422–429, 2006.

[12] D. Cheung, D. Wu, B. Laffoon, A. Lang, S. Snell, J. Zhang, P. Wang, J. Wu, P. Goodwine, W.Y. Wong, et al. Automated processing of appropriateness determination of content for search listings in wide area network searches, January 3 2006. US Patent 6,983,280.

[13] SJ Delany, M Buckley, and D Greene. SMS spam filtering: methods and data. *Expert Systems with Applications*, 39(10):9899–9908, 2012.

[14] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997.

[15] JJ Eberhardt. Bayesian spam detection. *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, 2(1):2, 2015.

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

[17] W Gad and S Rady. Email filtering based on supervised learning and mutual information feature selection. In *Computer Engineering & Systems (ICCES), 2015 Tenth International Conference on*, pages 147–152. IEEE, 2015.

[18] RP Gopalan and A Krishna. Duplicate bug report detection using clustering. In *Software Engineering Conference (ASWEC), 2014 23rd Australian*, pages 104–109. IEEE, 2014.

[19] J Hellerstein, J Thathachar, and I Rish. *Recognizing end-user transactions in performance management*, volume 19. IBM Thomas J. Watson Research Division, 2000.

[20] K Herzig, S Just, and A Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[21] Abram Hindle, Anahita Alipour, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering*, 21(2):368–410, 2016.

[22] H Hu, H Zhang, J Xuan, and W Sun. Effective bug triage based on historical bug-fix information. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 122–132. IEEE, 2014.

[23] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120. ACM, 2009.

[24] L Jiang, Z Cai, H Zhang, and D Wang. Naive bayes text classifiers: a locally weighted learning approach. *Journal of Experimental and Theoretical Artificial Intelligence*, 25:273–286, 2013.

[25] Z Jin, Q Li, D Zeng, and L Wang. Filtering spam in Weibo using ensemble imbalanced classification and knowledge expansion. In *Intelligence and Security Informatics (ISI), 2015 IEEE International Conference on*, pages 132–134. IEEE, 2015.

[26] A Lamkanfi, S Demeyer, E Giger, and B Goethals. Predicting the severity of a reported bug. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 1–10. IEEE, 2010.

[27] A Lamkanfi, S Demeyer, QD Soetens, and T Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, volume 322, pages 249–258. IEEE, 2011.

[28] David L Lanning and Taghi M Khoshgoftaar. The impact of software enhancement on software reliability. *IEEE Transactions on Reliability*, 44(4):677–682, 1995.

[29] A Lazar, S Ritchey, and B Sharif. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 308–311. ACM, 2014.

[30] QV Le and T Mikolov. Distributed representations of sentences and documents. In *international conference on machine learning, 2014*, volume 14, pages 1188–1196. ICML, 2014.

[31] Meir M Lehman and Juan F Ramil. Software evolution and software evolution processes. *Annals of software Engineering*, 14(1):275–309, 2002.

[32] MJ Lin, CZ Yang, CY Lee, and CC Chen. Enhancements for duplication detection in bug reports with manifold correlation features. *Journal of Systems and Software*, 2016.

[33] Y Liu, Z Liu, T Chua, and M Sun. Topical word embeddings. In *The 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 2418–2424. AAAI, 2015.

[34] R Moraes, JF Valiati, and WPGO Neto. Document-level sentiment classification: An empirical comparison between SVM and ANN. *Expert Systems with Applications*, 40(2):621–633, 2013.

[35] G Murphy and D Cubranic. Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 2004.

[36] Hoda Naguib, Nitesh Narayan, Bernd Brügge, and Dina Helal. Bug report assignee recommendation using activity profiles. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 22–30. IEEE, 2013.

[37] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[38] Thomas M Pigoski. *Practical software maintenance: best practices for managing your software investment*. Wiley Publishing, 1996.

[39] N Pingclasai, H Hata, and K Matsumoto. Classifying bug reports to bugs and other requests using topic modeling. In *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*, volume 2, pages 13–18. IEEE, 2013.

[40] V Rajlich. Software evolution and maintenance. In *Proceedings of the on Future of Software Engineering*, pages 133–144. ACM, 2014.

[41] I Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[42] I Rish, J Hellerstein, and T Jayram. An analysis of data characteristics that affect naive bayes performance. *IBM TJ Watson Research Center*, 30, 2001.

[43] Nivir Kanti Singha Roy and Bruno Rossi. Towards an improvement of bug severity classification. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 269–276. IEEE, 2014.

[44] NK Singha Roy and B Rossi. Towards an improvement of bug severity classification. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 269–276. IEEE, 2014.

[45] I Santos, C Laorden, B Sanz, and PG Bringas. Enhanced topic-based vector space model for semantics-aware spam filtering. *Expert Systems with applications*, 39(1):437–444, 2012.

[46] F Saric, G Glavas, M Karan, J Snajder, and B Basic. 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*, pages 441–448. Association for Computational Linguistics, 2012.

[47] B Schölkopf and CJC Burges. *Advances in kernel methods: support vector learning*. MIT press, 1999.

[48] Saniat Javid Sohrawardi, Iftekhar Azam, and Shazzad Hosain. A comparative study of text classification algorithms on user submitted bug reports. In *Digital Information Management (ICDIM), 2014 Ninth International Conference on*, pages 242–247. IEEE, 2014.

[49] J Su, JS Shirab, and S Matwin. Large scale text classification using semi-supervised multinomial naive bayes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 97–104, 2011.

[50] C Sun, D Lo, S Khoo, and J Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262. IEEE Computer Society, 2011.

[51] S Tan, Y Wang, and G Wu. Adapting centroid classifier for document categorization. *Expert Systems with Applications*, 38(8):10264–10273, 2011.

[52] Ferdian Thung, Pavneet Singh Kochhar, and David Lo. Dupfinder: integrated tool support for duplicate bug report detection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 871–874. ACM, 2014.

[53] Y Tian, C Sun, and D Lo. Improved duplicate bug report identification. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 385–390. IEEE, 2012.

[54] Yuan Tian, David Lo, and Chengnian Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *ICSM*, pages 200–209, 2013.

[55] Harold Valdivia Garcia and Emad Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th working conference on mining software repositories*, pages 72–81. ACM, 2014.

[56] S Wang, L Jiang, and C Li. Adapting naive bayes tree for text classification. *Knowledge and Information Systems*, 44(1):77–89, 2014.

[57] X Wang, L Zhang, T Xie, J Anvik, and J Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, page 461âĂŞ470. ACM, 2008.

[58] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 461–470. IEEE, 2008.

[59] Z Wei and G Feng. An improvement to naive bayes for text classification. *Procedia Engineering*, 15:2160–2164, 2011.

[60] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[61] Xin Xia, David Lo, Emad Shihab, Xinyu Wang, and Xiaohu Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology*, 61:93–106, 2015.

[62] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. Automatic bug triage using semi-supervised text classification. *arXiv preprint arXiv:1704.04769*, 2017.

[63] CZ Yang, Hou CC, Kao WC, and Chen X. An empirical study on improving severity prediction of defect reports using feature selection. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 240–249. IEEE, 2012.

[64] J Yang, Y Liu, X Zhu, Z Liu, and X Zhang. A new feature selection based on comprehensive measurement both in inter-category and intra-category for text categorization. *Information Processing & Management*, 48(4):741–754, 2012.

[65] Waleed Zaghloul, Sang M Lee, and Silvana Trimi. Text classification: neural networks vs support vector machines. *Industrial Management & Data Systems*, 109(5):708–717, 2009.

[66] Haiyi Zhang and Di Li. Naïve bayes text classifier. In *Granular Computing, 2007. GRC 2007. IEEE International Conference on*, pages 708–708. IEEE, 2007.

[67] Harry Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.

[68] W Zhang, X Tang, and T Yoshida. TESC: An approach to TExt classification using semi-supervised clustering. *Knowledge-Based Systems*, 75:152–160, 2015.

[69] Y Zhang, S Wang, P Phillips, and G Ji. Binary PSO with mutation operator for feature selection using decision tree applied to spam detection. *Knowledge-Based Systems*, 64:22–31, 2014.

[70] Jian Zhou, Hongyu Zhang, and David Lo. 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*, pages 14–24. IEEE Press, 2012.

[71] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, 2016.

[72] T Zimmermann, R Premraj, N Bettenburg, S Just, A Schroter, and C Weiss. What makes a good bug report? *Software Engineering, IEEE Transactions on*, 36(5):618–643, 2010.

Publication During MS Study

1. **Nizamani, Zeeshan Ahmed**, Hui Liu, David Matthew Chen, Zhendong Niu (2017). Automatic Approval Prediction for Software Enhancement Requests. Automated Software Engineering. *(Under Review after major revision). (SCI, CCF Rank B Journal)*

ACKNOWLEDGEMENTS

First and foremost, I am grateful to Almighty God for bestowing me the persistence to complete my research work.
It is a great honor for me to express my deepest gratitude towards my research supervisor Prof. Dr. Liu Hui for his excellent mentorship, guidance, suggestions and support in my research and studies. He was always available to discuss the problems and gave really valuable suggestions to solve them.
I am grateful to China Scholarship Council (CSC) for granting me the scholarship under Cultural Exchange Scholarship program that has supported me for the period of my Masters studies and research work at Beijing Institute of Technology.
My thanks and appreciations go to my family for the emotional and moral support.
Finally, I am grateful to all the friends for the being there for any support.