CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Identifying an Effective Set of Attributes for Machine Learning Based Bug Reports Classification

by

Muhammad Shafiq

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing

Department of Computer Science

2021

My dissertation work is devoted to My Family, My Teachers and My Friends. I have a special feeling of gratitude for My beloved parents, brothers. Special thanks to my supervisor whose uncountable confidence enabled me to reach this milestone.

CERTIFICATE OF APPROVAL

Identifying an Effective Set of Attributes for Machine Learning Based Bug Reports Classification

by

Muhammd Shafiq

(MCS171019)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Waseem Shahzad	FAST, Islamabad
(b)	Internal Examiner	Dr. M. Shahid Iqbal Malik	CUST, Islamabad
(c)	Supervisor	Dr. Aamir Nadeem	CUST, Islamabad

Dr. Aamir Nadeem
Thesis Supervisor
June, 2021

Dr. Nayyer Masood	Dr. Muhammad Abdul Qadir
Head	Dean
Dept. of Computer Science	Faculty of Computing
June, 2021	June, 2021

Author's Declaration

I, **Muhammad Shafiq** hereby state that my MS thesis titled "**dentifying an Effective Set of Attributes for Machine Learning Based Bug Reports Classification**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Muhammad Shafiq)

Registration No: MCS171019

Plagiarism Undertaking

(Muhammad Shafiq)
Registration No: MCS171019

Acknowledgement

“Say, He is Allah, [Who is] One. Allah, the Eternal Refuge. He neither begets nor is born. Nor is there to Him any equivalent.” Al-Quran [112:1-4]. I would like to say Alhamdulillah, for blessing me, with strength to finish this work. My heartfelt thanks to my parents for their love, prayers and everything what I need. A special thanks to my teachers, brothers for their support. Thanks to my friends, with whom this journey becomes easy to me. Last but not the least special thanks to my esteemed supervisor Dr. Aamir Nadeem for the support, for his assistance, inspiration and guidance in the field of research. I have learnt not only research under his supervision, but also his attitude towards others. May Allah shower his countless blessing upon all of you, JazakAllah.

(Muhammad Shafiq)

Abstract

When a requirement or specification is not correctly implemented it generate errors, these errors are referred as software bugs or software issues. In order to fix the issue or bug it is reported to the concerned developer or software Development Company. The process of reporting a software bug to its concerned developer is called bug classification and triaging, software bug can have significant impact on the overall functionality of the software system. Software bug fixing and resolution is an important activity in the software development process, and it consume huge amount of resources in terms of cost and time. Normally software bug passes from different stages (i.e. open, close, resolved and re-opened), once it is reported to the developer. In order to keep track of the bug fixing and resolution process, bug tracking repositories are used to classify and categorize software bug reports into different classes and categories. In order to effectively classify and triage software issues various approaches and techniques have been proposed by researchers. To classify software bug reports into different classes, textual and categorical attributes of the bug reports is used to extract features and then machine learning algorithms are applied on it. In the existing literature most of the approaches focused more on textual attributes of the bug as compared to the categorical attributes. However, categorical attributes are an important part of issue reports and can enhance the issue reports classification process more effective and fruitful. Moreover, an important issue is the selection of classes for classifying software bug reports.

In this work, we perform an analysis of software bug reports attributes, number of classes and classification algorithms for classifying software issue reports into different classes and categories. We use categorical and textual attributes of issue reports obtained from two different bug reports datasets and apply machine learning algorithms for classification. Textual attributes (i.e. bug title, bug summary and description) and categorical attributes (i.e. product, platform, severity, priority, and version) are extracted from the software issue reports. We perform experiments to evaluate the performance of textual and categorical attributes of bug reports. Furthermore, the performance of six machine learning algorithms is compared that has been used in this work. Moreover, we also assessed the performance of different issue report attributes by applying wrapper techniques to determine the subset of features having better classification performance and accuracy. Our obtained results depicts that by using textual and categorical attributes of issue reports combined significantly enhances and improves the classification results for binary and as well as for multi class classification. In case of machine learning algorithms performance support vector machine, naïve Bayes, Decision tree yields better results as compared to the rest of algorithms that has been employed in this research work.

Contents

List of Figures

List of Tables

Abbreviations

BTS	Bug Tracking System
BTR	Bug Tracking repository
DT	Decision Tree
FM	F-Measure
GB	Gaussian Bayes
KNN	K Nearest Neibour
NB	Naive Bayes
PR	Precsion
RF	Random Forest
RE	Recall
SVM	Support Vector Machine

Chapter 1

Introduction

In software development and engineering domain when a requirement or specification is implemented incorrectly, it generates errors. Normally these errors are referred as software bugs. In software development terminologies these errors are also known as issues, tickets, and defects. Bugs or issues can have high impact on the overall performance of the underline software system in which bug emerges [1]. Normally when a bug occurs in a program, it is reported to development teams of the software company which have developed that particular software application. Bugs tracking system is used to keep track of reported bugs filed against a particular software product [2]. Bugs tracking system (BTS) provides a thorough approach for managing the reported bugs which contains several information related to a particular bug. In short, bugs tracking system (BTS) keeps track of software bugs from the start to the end of the bug life cycle.

Normally bug or error occurs in software application due to various reasons such as developer mistakes, tools, and development or operational environment issues [3]. When a problem occurs due to the environment where particular software is running, normally it is not considered as a software bug. However, when a bug occurs within the software application then it is considered as potential bug. Once a software bug emerges (i.e. logical, design etc.), the first step in the resolution or fixing process of the bug is to report it to the appropriate developer or development team. Once the bug is reported the testing team or developer assess the reported bug using different information present in the issue report and assigns it to developer for fixing. This assignment and allocation process of software bugs to appropriate developers is called bug classification [4][5][6].

Currently most of the software development organizations use various types of bug tracking systems in order to facilitate the speedy and efficient resolution of reported software bugs. Software companies that develops closed source (i.e. software's which does not allow public access to its source code) software applications have their own Bug tracking systems (BTS) which is designed and implemented according to their needs, on the other hand Open-source software development projects maintain their own bug tracking systems and bug reports repositories. For instance, Bugzilla [7], mantis BT [7], [8] and Google chromium [9] are open source bug tracking repositories used by different open source projects like Mozilla and eclipse.

Bug classification process consumes huge amount of resources (i.e. time and cost) of development teams. Furthermore, manual classification of bug reports is not feasible due to high number of reported bugs on daily basis. Hence, manual classification process of reported software bugs issues is neither a standard nor a feasible practice to be used on regular basis in large and complex development environments [10].

arious techniques have been proposed in the literature that uses bug report title description and summary for classification purposes[11][12][13][14][15][16][17]. However, bug report contains both the textual and categorical fields that provide variety of information about a particular bug. Fewer techniques in the existing literature incorporate both the textual and categorical attributes for bug's classification. Furthermore, bugs can be classified or triaged using three main techniques, i.e. (a) machine learning, (b) dictionary and information retrieval based and (c) metadata based approaches. however, machine learning techniques tends to be more productive and effective in term of better classification results as compared to the dictionary and metadata based approaches. in machine learning approaches initially bug reports are retrieved from bugs tracking systems (BTS), then natural language processing techniques are applied on the bug reports for preprocessing, once the datasets becomes clean then feature extraction is performed on the basis of which the machine learning model is trained and tested.

Existing techniques for bug classification suffers due to several limitations such as varying performance on different bug datasets, less use of the categorical attributes of the bug reports, low performance in terms of classification accuracy. Fewer numbers of classes for classifying software bugs into different categories, which result in better accuracy, but their results are less fruitful and cannot be relied upon for accurately classifying software bugs. Furthermore, to the best of our knowledge no existing technique have performed any evaluation of different issue report attributes such as title, summary, severity and priority for determining their effectiveness in classification process.

In this work we perform an analysis and identification of effective software bug reports attributes, number of classes for classification and machine learning algorithms that can be used for assigning software bugs reports to relevant bug classes. The basic intuition behind our approach is to analyze bug classification process based on machine learning algorithm that can classify software bugs into different classes and categories with effectiveness. We use both categorical and textual attribute of bug reports, i.e. bug title description, long summary is selected from the textual part of the bug report, while attributes like product name, component, version, severity and priority selected from the categorical part of the bug reports. Furthermore, we use varying number of classes for classification with an aim to determine suitable number of classes that can yield better performance and results. Software bugs are classified into five different classes; these classes are actual bugs, non-bugs, logical bugs, enhancement bugs and graphical user interface bugs. We also applies wrapping technique for determining the subset of features from both categorical and textual attributes that yields better classification results as compared to the set of all features that we use in this work.

We use six machine learning algorithms as classifiers for experimentation; these algorithms are (a) support vector machine (b) naïve Bayes, (c) KNN, (d) Gaussian Bayes , (e) Decision tree, and (f) Random forest . The obtained results of these machine learning algorithms performance is compared against each other as well existing state of the art approaches in the literatures. Furthermore, to effectively evaluate the performance of our proposed technique we use four different performance evaluation measures which are accuracy, precision, F-measure, and Recall. Moreover, we have compared our proposed technique with existing state of the art bug classification frameworks.

1.1 Background Knowledge and Motivation

In software systems and engineering domain when computer program or software based system produce/generates an unexpected result, tis unexpected result is referred to as software bug. A software bug can be the result of wrong implementation of the specified requirements of software or due to the environment where that particular software is used. Software bugs can be detected during the testing phase of the development process; however, bugs can also occur in software systems during operations due to various reasons. Once a bug occurs in a computer program that bug is reported to the concerned development organization. normally, developers, software test engineers, end users reports that particular bug to concerned people in order to resolve and fix it [1][2].

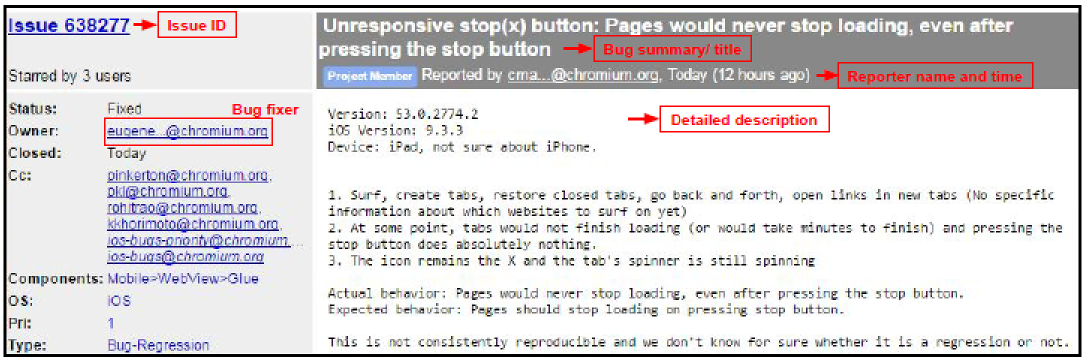
A software bug report contains different types of information related to a certain bug. Bug report consists of textual and categorical attributes that depicts different information about the nature bug. Bug report consists of textual and categorical attributes that depicts different information about a bug. The textual part of the software bugs contains the title description and detailed summary in the form of free text. While the categorical part consists of numerical and ordinal information, such as big ID, product, version, component name , reporter name, bug type, priority and many more details. Bug reports of specialized software applications such as telecom software's, embedded system and industrial automation system contains alarms and crash dumps, due to the involvement of hardware components as well, however, only 5.2% bug reports contains alarms and crash dumps data. Figure 1.1 shows an overview diagram of a bug reports in the open source bugs repository [18]. Software bugs reports are quite similar in nature in terms of their reporting techniques. For example, most of the bugs tracking system contain both textual and categorical information related to a particular software bug.

Normally software bugs/issues can be categorized or differentiated from each other using the predefine software bugs taxonomies. The following are the taxonomic categorization of the bugs [19].

1. A reported software bug/issue can be either actual bug or non-bug. Actual bug means that an abnormal activity or behavior in the program that contradicts specification, while non-bugs refer to a reported software bugs issue that is the result of the host environment where the software is used or the reporter have wrongly reported the issue.

2. Actual bugs can be further classified into different types depending on their occurrences in the software product. Normally actual bugs can be classified into two categories, that logical bugs or backend bugs, logical bugs refers to bugs which represent the programming errors in the software, while the backend bugs refers to the environmental errors or bugs where the software is used or developed.

3. Logical bugs can be further classified into graphical user interface (GUI) bugs and non GUI bugs. GUI bugs refers to errors and issues in the user interfaces of software system, while non GUI bugs refers to issues in other than GUI component or modules of software system.



Issue 638277

Starred by 3 users

Unresponsive stop(x) button: Pages would never stop loading, even after pressing the stop button

Reported by cma...@chromium.org, Today (12 hours ago)

Status:	Fixed
Owner:	eugene...@chromium.org
Closed:	Today
Cc:	pinkerton@chromium.org, pkl@chromium.org, rohitrao@chromium.org, kkhorimoto@chromium.org, ios-bugs-priority@chromium..., ios-bugs@chromium.org
Components:	Mobile>WebView>Glue
OS:	iOS
Pri:	1
Type:	Bug-Regression

```
Version: 53.0.2774.2
iOS Version: 9.3.3
Device: iPad, not sure about iPhone.


1. Surf, create tabs, restore closed tabs, go back and forth, open links in new tabs (No specific
information about which websites to surf on yet)
2. At some point, tabs would not finish loading (or would take minutes to finish) and pressing the
stop button does absolutely nothing.
3. The icon remains the X and the tab's spinner is still spinning

Actual behavior: Pages would never stop loading, even after pressing the stop button.
Expected behavior: Pages should stop loading on pressing stop button.

This is not consistently reproducible and we don't know for sure whether it is a regression or not.
```

Figure 1.1: Bug report in the Mentis BT bug tracking system

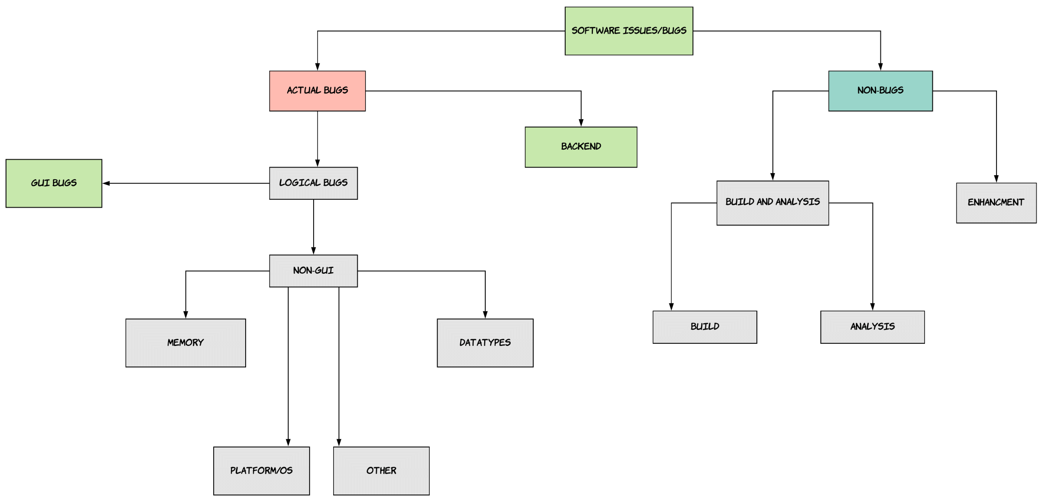


FIGURE 1.2: Software bug taxonomy

4. Non GUI bugs can further be divided into different types of bugs, such as data types, platform, Operating system and product related bugs.

5. Non bug issues refer to bugs or other than actual bugs, such as the implementation problems, integration and improvement or enhancement related problems.

Enhancement related issues may or may not be resolved logically. Actual bugs and problems are resolved programmatically and logically. In the software development paradigm enhancement related issues are not considered as bugs or defects, but it is classified as a further improvement issues. Normally enhancement related software issues have less priority in the bug's resolution process as compared to other bugs and issues. Figure 1.2 depicts software bugs taxonomy diagram of diverse bugs and their related classes.

1.1.1 Bugs Tracking System (BTS)

Bugs tracking systems commonly referred as BTS. Bug tracking system is a repository that maintains records of different software bugs reported by users or developers/test engineers against a particular software system. Bug tracking systems is considered an essential part of any software development environment. Both the closed source and large industrial software projects as well open-source software projects maintains their own respective bug tracking systems in order to make the process of bug assignment, classification and resolution easy, automated and effective. BTS is used to supervise management and tracking of reported bugs in the software development environment. Bugs tracking system offer variety of features to software test engineers and end users in order to reports bugs. For example, BTS offers facility for reporting new bug, retrieving and accessing information related to bugs, updating a specific bug report. Moreover, Bug tracking system plays an essential and important role in software issues management process. Bugzilla [7], Jira [20], Mentis BT [8] and Google Chromium [9] are some of mostly widely used and established bugs tracking systems available for open-source software projects.

1.1.2 Open Source Software Bugs Repositories

In recent years open source software development has grown with rapid face. Currently different open source projects are running in software development arena. Open source software development offer facility to their respective communities and members to contribute in the development of these software projects. Most of the open source projects maintain their own bugs and issue tracking repositories in order to facilitate software issues reporting process more convenient for users and software developers who contributes to these projects. Open source software development offer facility to their respective communities and members to contribute in the development of these software projects. Most of the open source projects maintain their own bugs and issue tracking repositories in order to facilitate software issues reporting process more convenient for users and software developers who contributes to these projects.The following are some of the most widely used and well established software bugs tracking database.

1. Bugzilla

2. Mentis BT

3. Jboss

1.1.3 Management of Software Bugs

The process of handling and maintaining software issues is called software bug management process. Issue or bug management process consists of different activities such as bug prevention, bug identification, classification of software bug/issue into different classes and categories, storing and maintaining record of reported bugs. Bugs can occur in any software system, and resolving these bugs requires huge amount of resource, hence, bug management is considered an important and fundamental activity in software development process.

1.1.4 Software Bugs Classification

Software bugs classification is one of the most important activities in bug management process. Bugs classification deals with grouping or assigning software bugs into different classes. Classification has several benefits, variety of patterns and useful data can be obtained by classifying software bugs into different classes. For instance, identifying frequently occurring bugs, category or module of the software system that have high number of reported bugs and nature of a reported software bug. The classification data of software bugs reports can be used to enhance and improve the software development process and to facilitate the development of new software development techniques that can reduce the number of software bugs.

Furthermore, Classification of software bug reports becomes necessary due to the huge number of new bugs reported on daily basis. According to statistics round about 100-150 new bugs are filed on daily basis in Bugzilla bug tracking system. Processing and assigning newly reported bugs to appropriate software development teams and developers require sufficient efforts and resources. Hence, due to the exponential number of bug reports in software development process the classification of bugs becomes a necessary task. Moreover, classification of software bug reports has several advantages, such as efficient management of development teams and resources, economical and cost effectiveness and ease in the identification of erroneous modules and components of software. Furthermore, classification of bugs significantly contributes in the quick and timely fixing of reported issues.

Bugs classification significantly reduces the efforts and cost required for bug fixing and resolution.

There are different applications of software bug's classification in the software development domain, such as (a) classification provides an easy way to identify and determine developers for a particular class of bugs, for example, using bug's classification we can determine developer that are good and have the technical background for resolving a GUI or Platform related bugs. (b) The second most clear and obvious benefit of classification is that newly reported can be easily assigned to the respective development teams or single developer based on the category of the reported bug. This will lead to quick and efficient resolution of software bug. (c) Classification of bugs helps software team leaders and mangers to perform complex and extensive data analysis task on the classified bugs in order to extract meaningful information about different aspect related to software development process.

Due to the above discussed importance of software bugs classification, various classification techniques and approaches have been proposed by researchers for classifying software bugs into different categories and classes. Most of the software bugs classification techniques uses machine learning algorithms for classification of the bug's reports into different classes and categories. However, the existing approaches suffers due to various limitations such as less use of the categorical attributes of bug reports, furthermore, the existing approaches have several limitations in terms of performance and accuracy and limited number of classes for classification. Moreover, very limited techniques have discussed the use of suitable number of classes to classify software bugs.

1.1.5 Motivation Behind Software Bugs Classification

The basic intuition behind the classification of reported software bugs is three folded, which is discussed in the following paragraph.

1. Bugs can occur in any software system either in the development process or in operations, timely fixing and resolution of bugs is a necessary task in any software development environment. Normally software bugs resolution process consumes huge amount of resources in terms of development cost and developer's time, classifying software bugs into relevant and appropriate classes can help in timely and effective resolution of software tickets.

2. Another motivation behind the automated and accurate software bugs classification is that automatic and accurate classification of bugs can significantly reduce developer's engagement; hence, it will help to allocate the free developer resources to other task instead of doing manual classification.

3. Software bugs reports contains both textual and categorical attributes, these attributes plays an important role in classification, hence, finding the most effective and suitable attributes from both the textual and categorical categories is very important and primary task in the classification process.

1.2 Problem Statement

Various approaches and techniques have been proposed in the literature to automate software bug's report classification process; however, most of the existing techniques suffer due to various limitations.

- Very few existing techniques have used both the textual (i.e. title, summary and Bug description) and categorical attributes (i.e. product, version, platform, component, priority and severity etc.) of bugs reports.

- In the literature very few approaches have focused on the selection of different Bug reports attributes in order to determine their effectiveness. Another issue is the selection of classes for classification. Various approaches have used the number of classes between two to five. For example in [21][22][23] the authors classified Bug reports into two main classes (i.e. corrective and perfective) and Bugs and non- Bugs classes. The use of less number of classes for classification can yield better performance; however, these results are less fruitful.

- The above discussed issues make the regular use of existing bugs triaging and classification techniques more challenging and complex in large industrial and open source software development environments.

1.3 Research Questions

We have formulated the following research questions which will be answered in this research work.

- **RQ1:**what is impact of using only textual attributes and using the combination of both textual and categorical attributes on bug's classification?

  To answer research question 1 we have analyzed the impact and behavior of both the textual and categorical attributes of the bug reports for classification. We will experiment with both textual and categorical attributes and will analyze their impact and significance in bug's classification process.

- **RQ2:** what is the impact on classification performance by increasing the number of classes from binary to multiclass classification?

  The proper and suitable selection of classes for classification of software bugs is a challenging and critical task; we will use varying number of classes in order to determine the accurate number of classes that can be used for classification.

- **RQ3:** what is the impact of machine learning algorithms in the classification process? In the existing literature we found that most approaches uses similar algorithms for classification, in order to assess the performance of Machine learning algorithms we will used different machine learning algorithm for classification.

- **RQ4:** what are the most effective textual and categorical attributes of software issue reports that can yield better classification results? In order to answer research question five, we use different subsets of textual and categorical attributes of reports in order to determine the best subset from our selected attributes that yields better classification results.

1.4 Objectives

The following are the main objectives of this work

1. The first objective of this research work is determine the impact of textual and categorical bug reports attributes in the classification process
2. The second objective is to investigate the impact of selecting varying number of classes for classification

1.5 Scope

The scope of this research work is only limited to the classification of open-source and publicly available software bug reports, furthermore, we do not consider bug reports from closed source software development projects. Moreover, this research considers only machine learning algorithms and classification techniques for classifying software bug reports.

1.6 Research Methodology

In the following steps we explain the research methodology for carrying out this research work.

1. In the initial step, we conducted literature review of most relevant approaches proposed for software bug classificatio process. After detailed analysis of

published literature we conclude that most of the existing classification techniques have several limitations. For example, few approaches have considered both the categorical and textual attributes of bug reports, categorical attributes are essential part of bug reports and can have an impact on the classification of bug reports, furthermore, we observed from the literature review that selection of suitable number of classes for classification is an important and crucial task, however, very few approaches have investigated this direction. Another limitation that we concluded from the literature review is that most of the existing approaches uses similar machine learning algorithm for classification purposes.

2. In order to cover the gap in the existing literature we have performed two main tasks, first to evaluate the effectiveness of textual and categorical attributes for classification. Secondly we perform experiments using different combination of textual and categorical attributes of software issues reports.

3. The implementation of our proposed work is discussed in the following steps;

   - In the first step we will obtain software bug reports datasets from widely used bug tracking systems, such as Bugzilla, Google chromium and Eclipse, and mentis BT.
   - In the second step preprocessing of bug reports dataset will perform. In the preprocessing of bug reports consists of different steps, it will start from sentence level segmentation of bug reports, then stop word removal will be applied, followed by lemmatization and feature extraction and feature selection processes.
   - In the next step we will extract both the categorical and textual attributes of the software bug reports, for instance, bug title and summary of the bug from bug report as well as categorical attributes like product, component, version, bug type, priority and severity of reported bug.
   - Once these attributes are extracted from the bug reports we will extract a feature set that will be used for mapping bugs reports into different classes and categories using TF-IDF and information gain.

- Once these attributes are extracted from the bug reports we will extract a feature set that will be used for mapping bugs reports into different classes and categories using TF-IDF.
- In fifth step we apply machine learning algorithm for classifying software bug reports into different classes and categories.

4. We use varying number of classes for the classification of software bugs, for instance, the label of classes are actual bugs and non-bugs, logical bugs, GUI bugs, and enhancement issues.
5. In the last step obtained classification results using machine learning algorithms are analyzed and evaluated using four evaluation measures, i.e. accuracy, Precision, Recall and F-measure.

1.7 Organization of This Thesis Work

The rest of this thesis document is structured as follows, chapter 2 surveys existing software bug reports classification approaches as well as summary and gap analysis in the existing literature. Chapter 3 presents our methodology for bug reports classification using machine learning classification algorithms. Chapter 4 discusses the implementation of our proposed approach followed by results and their details analysis and discussion. Chapter 5 concludes this thesis document with conclusion and future work.

Chapter 2

Literature Review

Software bugs can have high impact on the overall performance of software applications. Most of the software development uses variety of bug tracking system (BTS) for bug records. In order to locate and remove the bugs from software that is either in development process or already in operations, a process known as bugs triaging and classification is used. Bug triaging is process used to identify and classify software bugs into different categories. Normally software bugs are classified using their severity, summary and description part of the bug reports. In order to effectively monitor software bugs repositories both in open source and closed source software projects. Variety of bugs tracking and classification techniques are used. In the literature several techniques have been proposed for bug triaging and classification, some approaches use machine learning algorithms, natural language processing while other use information retrieval based techniques.

In this chapter we present related work that has been published in the area of software bugs classification and triaging. We conduct a survey of the existing approaches to find gap and limitations in the proposed techniques. Different challenges and problems have been extensively discussed in the existing literature by many researchers. One of the most complex and challenging task is the selection of a suitable number of classes for classification, as well the selection of textual and categorical attributes from the bug reports. In this Literature review we examine and investigate different techniques proposed for software bugs triaging and classifications.

2.1 Software Bugs Classification Approaches

Davor and Murphy 2004 presented a novel approach for triaging software bugs into different classes. The technique is based on multiclass classification that uses naïve Bayes classifiers to classify bugs into their respective classes [24]. In order to correctly classify the bug nature and then to assign it to the appropriate software developer, it requires bug description and developer history. To make the bug assignment process more reliable and effective verity of features are extracted from the description or summary of the bug reports. Naïve Bayes algorithm achieves better results on the Mozilla bug reports dataset for assigning the reported bugs to correct developers.

Anvik 2006 a semi-automated approach for bug triaging has been proposed by Anvik [25]. The basic intuition behind the approach is to develop a recommendation system that assists bug triagers to assign the appropriate bugs to respective developers. The recommendation system takes the textual description of the bug report in order to develop a model for different software developers that works in a project. The model assigns the bugs to different fixers based on the history of the previous resolved bugs by a single developer. The system takes different information related to bugs presented in bug report, such as bug component, operating system, hardware, version of the software and many more details. Machine learning algorithms have been used to evaluate the proposed approach.

Menzies and Marcus 2008 applied an automatic approach for predicting defects and bugs severity classification in the NASA dataset of bug's reports [26]. The prediction mechanism was based on the notion of multiclass classification containing four different severity classes that are critical, major, minor and trivial class. Experiments were performed on the NASA PITS dataset. However, the dataset is very small and was unable to exploit completely the approach. Due to the small number of training set, the approach achieved varying results in terms of precision and recall.

Anvik et al., 2009 a novel approach for automatic bug triaging and fixer recommendation was proposed by Anvik et al [27]. The technique use machine learning algorithms in order to develop a recommendation model that can be used to automatically assign the bug report to accurate developer. Verity of features are extracted from the summery and description sections of the bug reports, then a model is developed from the historical data of bugs reports, on the basis of historical data prediction is made about the new bug to be assign to a specific developer. The proposed technique is evaluated on Mozilla, GCC and Eclipse projects bug's reports, and achieves a moderate score in terms of precision and recall (i.e. 57% and 64% for precision and recall respectively).

Hindle el al., 2009 proposed a classification framework for software maintenance issue reports using machine learning algorithms [28]. The techniques classify different issue reports into five major classes that are corrective, adaptive, perfective, feature addition and non-functional classes. initially issue reports are extracted from version control system, and only 1% of the reports are manually labeled, then machine J48 machine learning is applied on the datasets to predict the actual class of the issue reports, J48 yields an overall accuracy of 81% on three different datasets, however, the most obvious limitation of the approach is the small number of dataset and manual labeling.

Lamkanfi et al., 2010 proposed a novel approach that predicts the severity of software bugs using the textual or summary description present in bug reports [29]. The proposed approach applies data mining and machine learning algorithms in order to predict the severity of software bugs using the textual description. Bugs are categorized on the bases of the two main severity levels that are major and critical to blocker bugs. The proposed technique applies Naïve Bayes algorithm in order to classify bugs according to predefine severity levels. One of the main limitations of the proposed approach is its domain specific nature. Hence it requires further enhancement in order to make it applicable for domain independent bug reports.

Chaturvidi and Singh 2012 Bugs classification using machine learning algorithms on the NASA bugs dataset has been performed by Chaturvidi and Sing[30]. This is the first attempt that has been made to classify the software bugs using machine learning algorithms using the summary section of the bug reports. The proposed technique classifies software bugs according to five severity levels identified by NASA. For the classification of bug's five widely used machine learning algorithm is used such as Naive Bayes, K-nearest Neighbour, Support vector machine, J48 and Naïve Bayes multinomial. Most of the machine learning algorithms achieved best performances when the top terms in the bug report description were selected.

Younus et al., 2012 the authors in [31] presented another classification approach to assist software bugs triagers in assigning bugs to relevant developer. The proposed approach classifies the bugs based on the summary of the bug report. First of all, features are extracted from the summary reports using the chi-Square and TFIDF techniques. After the features selection process multinomial Naïve Bayes algorithm is applied to the dataset. The dataset has been obtained from Bugzilla and Eclipse containing more than 25 thousands bug's reports. The multinomial Naïve Bayes classifier achieved an overall accuracy of 86% on both the Bugzilla and Eclipse bug reports datasets. However, one of the main limitation of this approach is it cannot handle the synonyms that frequently occurs in bug reports.

Kanwal and Maqbool 2012 Recommendation system to assign priorities to new software bugs using classification was presented by Kanwal and Maqbool [32]. The proposed approach applies support vector machine and Naïve Bayes machine learning algorithms in order to determine the priority of software bugs in bugs reporting repositories. Two new evaluation measures are also proposed that can be fruit full in determining the accurate priority of bugs. The approach considers two main features such as categorical and textual features. the obtained results depicts that SVM performs better then Naïve Bayes in terms of textual features, while Naïve Bayes yields better results in term categorical features. However, the approach achieved better results when both the textual features and categorical features were combined.

Alenezi and Bantiaah 2013 another similar approach for the prioritization of software bug reports using machine learning algorithm and texting has been presented by Alenezi and Bantiaah [33]. Support vector machine, naïve Bayes and decision tree algorithms is used to classify bug reports according to their priorities. Initially cooperative or related features are extracted from the bugs reports descriptions and summaries, and then SVM, Decision trees and Naïve Bayes algorithm is applied to classify bug reports according to their priorities. The obtained results depict that SVM and Decision trees completely outperformed Naïve Bayes algorithm.

Zhang et al., 2015 presented an approach for predicting the severity of software bugs reported in open-source software repositories like Mozilla Firefox and Eclipse [34]. The proposed techniques predict the severity of software bugs using concept profiles by mining software bug's repositories. Historical bug's data are mined from the bug repositories using the different. Machine learning algorithms (i.e. KNN, Naïve Bayes and multinomial Naïve Bayes) are applied on the Mozilla Firefox and Eclipse dataset in order to severity of the reports bugs. The proposed technique achieves better results in terms of precision and Recall using the concept profiles. However, extracting concept profiles from software bug's repositories is a trivial task.

Goyal et al., 2015 presented an approach for classifying software bug reports using data mining and machine learning algorithms [35]. The basic intuition behind the approach is to develop a mechanism that can automatically classify bug reports based on their severity levels. In order to accomplish this authors initially clustered the reports based on the title of bug reports. After the clustering process classification algorithms are applied to predict the severity of specific reports. The proposed technique predicts the priority of bugs in two phases. In the initial step similar bugs are combined into clusters based on different attributes such as report title. In the second phase, classification algorithms are applied to clusters in order to assign a priority of the different bug reports. All the classification algorithms (i.e. Random forest, Bayes nets and SMO) yield better results on the clustered bug reports as compared to the results on un-cluster data.

Gujral et al., 2015 presented an approach for predicting and classifying software bugs reports using text mining and machine learning algorithms [36]. The proposed technique combines different dictionary terms that highlight the severity of specific bugs present in bug reporting repositories. The technique develops a dictionary of most common words that occur frequently in bug reports. The basic Intuition behind the approach is to combine those terms that often describe the severity and non-severity of terms to form a dictionary that will be used to assign priorities to new incoming bug's reports. TFIDF is used to extract features or terms form bug reports that will be used for predicting bug severity and non-severity.

Pushaplatha and Murnalini 2016 presented an ensemble based classification technique for predicting software bugs reports severity [37]. Begging ensemble method is used to predict the severity of software bugs in open source bugs repositories. Bugzilla dataset has been used in the evaluation and experimentation process. Furthermore, the begging ensemble method is compared against the C4.5 classifier. The technique predicts the severity of bugs on seven predefined severity levels. The technique achieves better results on the underline dataset; however, the technique is domain dependent and scales very poorly on other datasets.

Jin et al., 2016 another research work presented by Jin et al [38] tried to enhance the prediction process of bug's severity using machine learning algorithms. The technique takes only those bugs reports that are reported with normal severity level. Various features like product name, component, reporter and severity of the reported bugs are included. The technique only focuses on the bugs that have normal level of severity which most of the existing approaches does not consider in the prediction process. Only normal severity levels of bugs are obtained from Eclipse and Bugzilla bug repositories. The proposed approach achieves better results in terms of precision and recall (i.e. 70 % precision and 74 % recall).

Zhang et al., 2016 presented an automatic bug's severity prediction and fixer recommendation system using machine learning algorithms. Historical bug's data is extracted and process using REF and K-NN algorithm to find patterns that similarity to new reported bugs [39]. In the next phase verity of features like reporter and similarity among the existing reports and new reports is identified using similarity measures. The approach is evaluated on five different datasets obtained from GNU compiler collections (GCC), open office, Eclipse, Mozilla and Net Beans. Experimental results on these datasets reveals that the approach out performs the existing state of the art bugs severity prediction and recommendation system. However, the approach is a semi-automatic and requires human interference and expertise.

Xuan et al., 2017 presented a semi supervised based classification technique for the triaging new bugs reports using a machine learning algorithms and weighted recommendation list of candidate developers for the resolving a specific issue [40]. The proposed technique makes use of both label and unlabeled bug reports for classification. Initially the naïve Bayes classifier is trained with label bugs reports and then unlabeled bugs are used for training the classifier. The proposed approach achieves an overall accuracy of 75% on the Bugzilla and eclipse datasets.

Mani et al., 2018 presented a novel approach for bugs triaging and classification using a DBRNN-A [23]. The techniques takes bugs reports from three open source bugs tracking system and applies the neural network algorithm in order to extract the syntactic and semantic information from the bug reports. Machine learning algorithm is applied to classify and triage the bug reports which also contain the unfixed bug's reports. The proposed technique only considers the bug title and bug description of the bug report and achieves an overall average accuracy of 78% on the three open source datasets.

Kukkar and Mohna 2018 A hybrid approach that combines text mining (TM), Natural language processing (NLP) and machine learning for classifying software bug reports into actual bugs and non-bugs has been presented by Kukkar and Mohna [41]. The technique combines TM, NLP and machine learning algorithm and four incorporated fields related to software bugs. These fields are reporter, severity, priority and component in the bug's reports. TFIDF and bigram method is used to select and extract features from the textual description of bug's reports summary. After the extraction process K-NN algorithm is applied to five different bug reports datasets. The dataset are obtained from open source bugs repositories like Bugzilla, Eclipse, Jboss, Firefox and open FOAM. K-NN algorithm achieves better results in terms of precision, recall, and accuracy when TF-IDF is used.

Pushaplatha and Murnalini 2019 Bugs reports classification and severity prediction technique for closed source software bugs has been presented in [42]. Four widely used machine learning algorithms are applied on the PITS dataset of NASA obtained from PROMISE bugs reporting repository. The proposed technique uses begging, voting, Adaboost and ensemble methods to predict the severity of bug classification. To reduce and remove redundancies in the dataset two preprocessing techniques (i.e. information gain and Chi-Square) are used. The PITS dataset is divided into five different categories alphabetically starting from A to F. Information gain method significantly contributes to overall accuracy. All of the employed algorithms achieve the same results on all the PITS when information gain is used for data dimensionality reduction.

Ramay et al., 2019 an approach for classifying and predicting bug severity using deep neural network has been presented by Ramay et al [43]. Initially natural language processing techniques are applied to preprocess the bug's reports, and then a new score is calculated that is based on the emotion of bug reporters. Finally a vector is created from is the preprocessed bug reports which is passed into a deep neural network in order to predict the severity classes (i.e. critical, miner and blocker). The proposed technique achieves an overall F-measure score of 75%.

Otoom et al., 2019 presented an automated approach for bug's reports classification. The proposed approach extracts different features form the description of bug reports [21]. The feature set is developed on the basis of the common occurrences of the different key words in the summary and description part of the reported bugs. The developed feature set is then passed to machine learning algorithms for classifying the reports in two main classes that are corrective and perfective classes. the proposed approach achieved an overall accuracy of the 93% on three different datasets obtained from aspectj, Tomcat and SWT. One of the main limitations of the proposed approach is the small number of the dataset and the very limited number of feature set.

Sarkar et al., 2019 proposed a classification based approach for bug triaging at Ericsson which is major telecom services and technology provider through the world [22]. The proposed technique combines textual attributes with categorical attributes of the bug's reports; it also incorporates additional information such as alarms and crash dumps collected from the bug reports. A bug reports dataset is developed using the internal bugs tracking system used by Ericsson. a simple logistic regression machine learning algorithm for classifying and triaging bugs reports to different development teams. The most important aspect of this approach is the use of the confidence score for assigning a particular issue to a certain developer. The proposed technique achieves accuracy of 90%.

Otoom et al., 2019 presented an automated approach for bug's reports classification. The proposed approach extracts different features form the description of bug reports [21]. The feature set is developed on the basis of the common occurrences of the different key words in the summary and description part of the reported bugs. The developed feature set is then passed to machine learning algorithms for classifying the reports in two main classes that are corrective and perfective classes. the proposed approach achieved an overall accuracy of the 93% on three different datasets obtained from aspectj, Tomcat and SWT. One of the main limitations of the proposed approach is the small number of the dataset and the very limited number of feature set.

Sarkar et al., 2019 proposed a classification based approach for bug triaging at Ericsson which is major telecom services and technology provider through the world [22]. The proposed technique combines textual attributes with categorical attributes of the bug's reports; it also incorporates additional information such as alarms and crash dumps collected from the bug reports. A bug reports dataset is developed using the internal bugs tracking system used by Ericsson. a simple logistic regression machine learning algorithm for classifying and triaging bugs reports to different development teams. The most important aspect of this approach

is the use of the confidence score for assigning a particular issue to a certain developer. Bug reports are triaged and classified based on the confidence score given by classifier to each developer. The proposed technique achieves accuracy of 90%.

Table 2.1: Summary of different approaches investigated in the literature review

Ref.	Description	Attributes used		Performance				Algo.	Dataset	Labels	Analysis
		Tex.	Cat.	Acc	PR	RE	FM				
D. Cubranic and G. C. Murphy 2004	Assign bugs to correct software developer to be resolved in time using machine learning algorithm	bug description and summary	Nil	30%	Nil	Nil	Nil	Naïve Bayes Classifier	Mozilla Firefox dataset	2	Naïve Bayes achieve better results on small training sets
John Anvik 2006	Recommendation system for bugs assignment to expert developers	bug description and summary	product, component, version, platform, priority	67%	Nil	Nil	Nil	SVM and Clustering algorithm	Eclipse and Bugzilla datasets	2	The approach is still manual at large and requires human expertise
Menzies and Marcus 2008	Predicting severity and non-severity of bugs reports using machine learning algorithm	bugs description, summary	priority	Nil	91%	59%	71%	Naïve Bayes	NASA promise Dataset	5	The approach performs well on small datasets, however, it performs poorly on large dataset
Murphy et al 2009	Automatic approach for new bugs prediction and assignment to appropriate developers	bug description and summary	Product, component, version	69%	71%	73%	Nil	Support vector machine	Mozilla, GCC and Eclipse	2	SVM yields better results on both Mozilla and Eclipse datasets, however, it poorly performs on GCC dataset
Hindl et al 2009	machine learning based classification technique for classifying issue reports in to different classes	issue title, description	author, commit information, type	81%	Nil	Nil	Nil	J48 and SMO	MySQL, boost. spring framework		the classification accuracy is better, however, the manual labeling of dataset is quite challenging task

Lamkanfi et al 2010	Predicts the severity of bugs using machine learning algorithm and data mining	bugs description, summary	product, component, version, platform	Nil	82%	78%	Nil	Naïve Bayes	Mozilla Firefox, Eclipse and GNOME datasets	2	the technique achieves efficient results, however, the main limitation is the domain specific nature of the technique
K. K. Chaturvedi and V. B. Singh 2012	Software bugs severity using machine learning	Bug title, description and bug type	priority	65%	Nil	Nil	74%	SVM, Naive Bayes, K-Neighbour	NASA software bugs Dataset	Nill	The algorithms achieves varying results in terms of precision and Recall
M. Younus Javed Hufsa Mohsin 2012	Software bugs classification using Naïve Bayes and Chi-Square and TFIDF	bug title and description	product, component, version, platform, priority	83%	Nil	Nil	Nil	Multinomial Naïve Bayes	Bugzilla and Eclipse Datasets	Nil	Naïve Bayes performs well in terms of accuracy on Bugzilla, however, the Eclipse results are not satisfactory
Jaweria Kanwal and Onaiza Maqbool, 2012	Prioritize new bugs to assist the bug triaging process using machine learning and text mining	Title, summary	product, component, version, platform , priority	Nil	85%	81%	Nil	SVM and Naïve Bayes	Eclipse dataset	5	SVM and Naïve Bayes generates efficient results when textual and categorical features are combined, however, the approach is still dependent on human expertise
Mamdouh Alenezi and Shadi Banitaan, 2013	Prediction and classification to prioritize new reported bugs using machine learning techniques and text mining approaches	bugs description, summary	priority	Nil	41%	39%	40%	SVM, Naïve Bayes and Decision Trees	Eclipse and Firefox	3	Decision trees and SVM performs better on both datasets, however, Naïve Bayes yields varying results in terms of precision, recall and accuracy

ShruuGujral et al 2015	Texting mining based dictionary and ML algorithm is used to predict the severity and non-severity of bugs	bugs description, summary		91%	Nil	Nil	Nil	Naïve Bayes multinomial	Bugzilla	2	The approach is heavily dependent on the dictionary terms
Neetu Goyal et al 2015	Close source software bugs severity prediction using machine learning algorithms	Description title and bugs report summary	Priority	75%	67%	759	Nil	K-mean, Bayes net, RF and SMO	Closed source dataset	3	RF, Bayes net and SMO achieved better results on the clustered reports as compared to un-clustered reports
Tao Zhang et al 2015	Predicts the severity of bugs using concept profiles and machine leaning algorithms	bugs description and summary	priority	Nil	81%	89%	85%	KNN, Naïve Bayes and Multinomial Naïve Bayes	Mozilla Firefox and Eclipse	2	The applicability of the approach is only limited to open source software projects
Pushpalatha M N and Mrunalini M 2016	This approach predicts the bug severity using ensemble begging classifier	bugs description and summary	priority	79%	81%	77%	Nil	Ensemble begging classifier	Bugzilla dataset	3	The proposed technique achieves better results, however, it does not cover cross component bugs
Tao Zhang et al 2016	Automatic bug severity prediction and fixer recommendation system for open source bug repositories	bugs description, summary, product, version		75%	Nil	Nil	Nil	K-NN and R£F	GCC, open office, Eclipse, Net Beans and Mozilla		The approach out performs the existing approach for severity prediction and fiver recommendation, however, the process is still manual at large
Jin et al 2016	Predicts bugs severity of normal severity level of bugs using machine learning algorithms	bug description and summary	product, component, version, status, priority	Nil	78%	79%	-	Multinomial Naïve Bayes algorithm	Eclipse and Bugzilla	2	MNB achieves better results on Bugzilla dataset, however, its performance varies on Eclipse bugs reports

Xuan,et al 2017.	Classification based approach for bug triaging using machine learning algorithms	bugs description and summary		75%	Nil	Nil	Nil	Naive Bayes	Bugzilla and eclipse	2	naive Bayes achieves good results but the approach is Manuel at large
mani et al 2018	Novel approach for bugs classification and triaging using deep learning	bugs title and description		78%	Nil	Nil	Nil	Deep learning (DBRCNN-A)	core, Firefox and Google chromium		the approach generates better results however, it only considers the textual information of the bugs report
Ashima and Mohana 2018	Classification of bugs reports into bugs and non-bugs	bug description, title and summary	component, status, priority	86%	78%	80%	79%	K-NN	Bugzilla, Firefox, Jboss, Eclipse and open FOAM	2	The feature extraction process is very time consuming and heavily dependent on specific domain
Ramay et al 2019	Prediction of bug severity using deep neural network and emotion based features	bugs description, summary	product	87%	80%	80%	81%	Deep neural network	Eclipse and Mozilla		Deep neural network achieves good results, however, the score emotion calculation process can influence the results
Otoom et al 2019	Automated classification of software bugs reports using the feature set instead of complete reports	Bugs title description and summary	report status	93%	Nil	Nil	Nil	SVM, Decision tree	Tomcat, Aspectj and SWT	2	the approach achieves an overall accuracy of 93% on very small and manual labeled dataset
Sarkar et al 2019	bugs triaging and classification technique using ML and confidence score	bugs description and summary	product, site, priority, customer, configuration, generation	79%	78%	79%	Nil	logistic Regression	Ericsson dataset	2	the technique achieves better accuracy, however, the no of triage bugs is small

Pushaplatha 2019	Four machine learning algorithm is used for predicting bug severity of closed source software projects	bugs description, summary	priority	81%	Nil	Nil	Nil	Begging, Random forest, Adaboost and ensemble classifiers	PITS dataset obtained from NASA PROMISE repository	2	Information gain method gives better results in terms of accuracy as compared to Chi-Square method

2.2 Summary and Gap Analysis of the Literature Review

In this section we have performed a comprehensive analysis of the existing techniques for bugs classification and triaging shown table 3. There exists an extensive literature on bug's classification and triaging process. In the existing literature most of the approaches uses machine learning algorithms and some techniques are based on text mining and information retrieval as well as dictionary and metadata based. However, most of the existing techniques have some limitations and short-comings, and therefore, they are unable to fully address the problems and issues of automated bugs classification and triaging process. The following are the findings of our conducted literature review.

1. Most of the existing approaches use only the textual attributes of bug reports, although some approaches have considered the categorical attributes present in bug's reports, however, only a few a categorical attributes are considered for the classification and triaging process. For example, in [25][37][28]. [29][31][22] used different categorical attributes likes product, platform, developer, status and reporter and textual attributes like bug title and description for classifying software bug reports. However, all of these approaches classified software bug reports in two classes, that were actual bug's and non-bug's classes. These approaches achieved better results in terms of classification but their results are less fruitful to be used in real development and

bug classification environments. Furthermore, authors in [33][35][26]used 3 and five number of classes for classifying software bug reports, however, they mostly focused on the textual attributes of the bug reports. Although, categorical attributes can be beneficial for the accurate classification and triaging of the bugs reports.

2. Another interesting point we have observed from the detailed analysis of the literature review is that most techniques have used very few machine learning algorithms, for instance, algorithms like (i.e. naïve Bayes, Support vector machines, decision trees and KNN) are widely used for bugs classification and triaging process. Although few approaches have used deep learning algorithms such as deep neural network and DBRNN-A algorithms, but in the case of the deep learning algorithm only textual bug attributes are considered. Hence, other machine learning and deep learning algorithm can be applied in bug's classification process.

3. Another issue that has been observed from the detailed analysis of the literature review is the number of the classification classes that have been used, for instance, in [21][22][42] the authors only used two classes (i.e. perfective and corrective class) for the classification of software issue reports. The use of the less number of classes significantly increases the accuracy, but their results are less fruitful. Selecting the number of accurate and suitable classes for classification is a challenging and crucial task.

4. No existing approach in the literature has performed the comparison of different attributes of issue reports.

5. The last major issue that we have observed during the literature review is that only one proposed technique by Sarkar et al[22] have used the feature of confidence score, the high confidence score means the model have triaged or classified particular software bugs report with higher confidence score.

6. Furthermore, to the best of our knowledge no approach in the literature has evaluated the performance of different attributes of software issue reports.

Chapter 3

Proposed Approach

In this chapter we explain the working of our proposed approach for bugs classification. We describe different components and steps involved in the implementation of the propose work , for instance dataset selection, preprocessing, selection of classes and applying machine learning algorithms on the three open sources bug reports datasets. We explain the working of the proposed approach step by step in this chapter.

3.1 Selection of Bug Reports Dataset

In order to effectively categorize and classify software bugs reports into different classes using machine learning algorithms. We have selected and retrieved bug reports from two widely used bugs tracking repositories. These repositories are Bugzilla and Google Chromium. We extracted two bug datasets from Bugzilla [44] and Google chromium [45] bug tracking system. The dataset contains bugs reports filed against Mozilla core and eclipse platform

The composition of our datasets consists of forty thousand bug reports. From which twenty thousand bug reports belong to core category of Bugzilla product list, twenty thousand bugs from eclipse product category. We extracted all these bug reports in CSV format, currently; most of the bug tracking systems provides

Table 3.1: Statistical information for Eclipse Dataset

Eclipse Dataset	
Total No of Features	14
Total No of Instances	20000
Total No of Classes Data	4 (Enhancement bug, GUI bug,Logical bug, No bug)
Total No of Instances of Enhancement bug	4398
Total No of Instances of GUI bug	5025
Total No of Instances of Logical bug	7815
Total No of Instances of No bug	2757

Table 3.2: Statistical information for Firefox Dataset

Firefox Dataset	
Total No of Features	12
Total No of Instances	19937
Total No of Classes Data	4 (Enhancement bug, GUI bug, Logical bug, No bug)
Total No of Instances of Enhancement bug	3457
Total No of Instances of GUI bug	5298
Total No of Instances of Logical bug	8215
Total No of Instances of No bug	2967

the facility to download bug reports for different purposes. We extracted the bugs reports filed from 2010 to 2018 in these bugs tracking systems. Normally some bug reports contain alarms and crash dumps. However, only 5.2% of bug reports contain alarms and crash dumps [46], we ignore and discard those bug reports that contains alarms and crash dumps. Furthermore, processing of alarm and crash dumps reports required huge amount of resources and it rarely contributes in improving bugs classification.

Moreover, the selection of software bug reports dataset is also based on the literature review we conducted in chapter 2. As most of the approaches have used te Eclipse and Firefox dataset for experimentation. The Tables 3.1 and 3.2 we provide the statistical information about the selected datasets used in this research work

3.2 Dataset Preprocessing

One of the most fundamental activities in bug's reports classification process is the preprocessing activity. In order to clean the dataset from noise and irregularities for experiments we perform the preprocessing of bug reports. The preprocessing consists of the following steps.

1. Segmentation of bug reports: the first step after reading the dataset files in CSV format; we separated the bug reports in to sentences level segments.

2. In the next step we perform tokenization process; tokenization is applied for separating each term into tokens.

3. After the tokenization process natural language processing technique is applied for removing stop words. Stop words are the most common occurrence of frequent words in the text that have no impact of the overall semantic of the text. We remove stop words such as "is", "the", "are", "it" and many more word like this. However, we give great care with the removal of stop words, we applied negation in order to not remove world like "not", "don't" present in the bug reports. . We did this due to retain the actual status of the text in bug report. For example if a title of bug contains text like "this login option is not working" and if we remove the stop word "not", it will completely alter the semantics of the bug title which will means that the login option is working correct instead of wrong. Hence, keeping in mind the importance of certain terms in the bug's reports, we applied the negation function to retain those terms that are necessary for maintaining the actual meaning of the text.

4. IV. After the stop word removal process we apply lemmatization technique to stem all terms into their base form. In this work we use porter stemmer and snowball stemming algorithm [47] for reducing certain key words into base form.

5. Once the stop word removal and lemmatization process completes, we extract features from the datasets of the bug's reports. Feature set is extracted

for mapping bugs reports into different classes based on the occurrence of common words present in bug reports. We extract feature set against the most common occurrence of certain key words that represents bugs that belongs to different bug classes, such as actual bugs, non-bugs, logical bugs and enhancement bugs.

3.3 Feature Set

Normally in bug classification process a bug report is assigned to a particular class based on the feature set commonality. We extract and develop a feature set based on the most common occurrences of different terms that is normally used in bug reports to refer to a certain problem. For example, to distinguish a graphical user interface bug from data type bug, normally terms like buttons, boarders, page, pixel and other similar terms are present in the bug reports that refer to graphical user interface (GUI) bugs. We extract more than 200 terms from the bugs reports, which will be used as feature set for mapping software bug reports to different classes and categories.

3.4 Selection of Bug Attributes

Normally, software issue reports consist of different fields and attributes. These attributes or fields are classified into two categories, (a) textual attributes and (b) categorical attributes. Textual fields contains information like bug title, description and summary attributes of the bug attributes. Normally these attributes are in free text form. Textual attributes contains very useful information that describes the occurrence of issue in a software system. Furthermore, the textual information plays an important role in classification process of bug reports. However, processing and getting meaningful information from the textual attributes is a challenging and complex task [48].

On the other hand categorical field consists of different information related to software bugs. Majority of software bug reports contains several categorical information such as product name, version, component name, priority, reporter name, reporting time, and date etc. apart from this bugs reports contains several other categorical attributes.

In order to effectively classify bug reports in correct and appropriate classes, we select and choose different combination of both categorical and textual attributes for experiments. For example, we first use textual attributes like title and summary and evaluate its classification performance. Next we use both the bug title and summary attributes of issue reports and assess its classification performance. However, from the literature survey we conducted in chapter 2 of this thesis we observed that using only textual attributes of the bug reports does not provide fruitful results. Hence, we combine and use both categorical and textual attributes of bug's reports in order to improve the performance of our proposed approach. Moreover, the software bug reports attributes are selected keeping in mind the attributes that have been used in the literature. For instance, in the literature review we observed that most of the existing approaches have used title, summary and description attributes from the textual attribute category, while in case of categorical attributes like severity, priority, platform and versions are used extensively. Our proposed approach for software bugs classification employs 8 bug report attributes shown in Table 3.3.

3.5 Feature Selection Method

Normally there are different methods to select features from the datasets, one of the easiest but very time consuming and expensive method is to select all the possible combination of features present in the dataset and run experiments based on that. However, this feature selection method is not feasible when we have more number of features. The other method is to use a filtration method for the feature selection process. In our case we initially extract features using a filtration method and after that we apply wrapper technique to find the best and effective set of

Table 3.3: textual and categorical attributes of bug reports that has been used in this work

Attribute type	**Attribute name**	**Short description**
Textual	Title description	Brief description and details about the bug
Textual	Detail Summary	Explanation of the reported bug with possible reproduction mechanism
Textual	Expert suggestion	Answers or suggestions of the developers against a particular bug
Categorical	Priority	This shows the criticalness of the bugs on the scale of 1 to 5
Categorical	Product	The name of the product in which the bug appeared
Categorical	Version	the specific version of product
Categorical	Severity	Impact of the reported bug in the functionality of the reported bug, severity classes are minor, major, critical, blocker etc.
Categorical	Platform	Platform of the product in which the bug appeared

attributes. In order to select textual features we used the filtration method of TF-IDF, while for the selection of categorical features we used the information gain.

We have used the wrapper technique for the subset evaluation in order to determine the effectiveness of our selected 8 attributes of software bug reports. Wrapper technique offer different methods and ways to perform subset evaluation. In our case we have applied the classifier subset evaluation technique within the wrapper method for determining the effectiveness of the different attributes of software bug reports.

3.6 Feature Selection Method for Evaluating the Effectiveness of Bug Reports Attributes

In machine learning, Feature selection is the process of choosing variables that are useful in predicting the response (Y). It is considered a good practice to identify which features are important when building predictive models. In our thesis we have used one of the well-known feature selection techniques (i.e. wrapper method) which are used by various authors in literature. In these methods, the feature selection process is based on a specific machine learning algorithm that we are trying to fit on a given dataset. It follows a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion. The evaluation criterion is simply the performance measure which depends on the type of problem. For classification the evaluation criterion can be accuracy, precision, recall, f1-score, etc. Finally, it selects the combination of features that gives the optimal results for the specified machine learning algorithm. In the attributes wise comparison for finding the effectiveness of our selected attributes, we have used the wrapper (i.e. classifier subset evaluation) in order to find the effectiveness of the selected attributes that we have used in the this research work. We have applied first TF-IDF and information gain in order to limit the feature set obtained from the dataset. Because applying wrapper technique on large number of feature yields huge number of combinations. In order to avoid tis we applied the filtration method. The main reason for applying wrapper technique is to determine the most effective set of attributes from the list of all attributes that we have used in experimentation

3.7 Classes Selection for Classification

Traditionally, in the classification problems data instances are mapped into different labeled classes based on the similarity and commonality between them. In software bugs classification various approaches have used different number of classes for classifying software bugs into different categories. However, some techniques use only two classes while some techniques have used more number of classes. It is evident from literature that less number of class's yields better performance; however, their results are not satisfactory and cannot be relied upon. On the other hand more number of classes yields moderate results in terms of performance but it leads to better classification.

In order to counter number of classes issue for bug's classification, we use total five classes. These classes are actual bugs and non-bugs. Furthermore, these two classes are divided into sub-classes. Such as actual bug class is divided in two classes i.e. logical and GUI class. Non-bug class is divided into two subclasses, such as enhancement and analysis/build class. The basic intuition behind our selected number classes is that fully covers bugs taxonomic categorization. Furthermore, our feature set is comprehensive and covers all the selected number of classes. In the experimentation and evaluation phase we use varying number of classes in order to evaluate the performance of machine learning classification algorithm on all the classes. We believe that use of variable number of classes will leads us to the selection of accurate number of classes that can be used for classifying software bug reports. List of features of respected categories present in Table 3.4.

3.8 Conversion of Classes into Binary and Multiclass

We have classified software bug reports into different classes, these classes are actual bugs, non-bugs, logical bugs, enhancement bugs and graphical user interface bugs. As our database is consist of multi labels, so in order to convert the dataset

Table 3.4: List of features set extracted from Eclipse and Firefox Bug reports dataset

Class name	**Feature set**
Actual Bugs	˜Actual bugs are the software issues ˜which is included in the logical, GUI ˜and enhancement bugs
Non Bugs	Terms which are not included in the feature set for the rest of the classes
Logical bugs	assertion, annotation,˜ argument, application, attempting, break, broken, behavior, badly, call, cause, code, clustering, component, core, default, ˜does not, error, exception, ˜edit, found, fail, frame, handle, host, implement, incorrect, integrate, incomplete, ˜java, library, logic, miss, mention, ˜work, null, option, pointer, parameter, pluggable, problem, ˜portability, redirect, remove, ˜read, replica, run, repeat, server, session, submit, search, statement, status, service, start, throw, validate, wrong, not, proper, should, array, blob, binary, char, numeric, hard, real, string, text, variable, value, cache, crash, throw, fault, ˜heap, memory, segmentation, ˜segmentation-fault, smart, threads
GUI bugs	button, border, background, blank, bundle, CSS, container, captcha, display, event, font, HTML, item, image, list, label, ˜line, layout, locale, method, message, navigator, pixel, page, render, resource, space, selection, show, tag, toolbar, typo, click, mouse, key, resolution, table
Enhancement bugs	add, enhance, ignore, improve, optimization, performance, required, support, can, may, suggest,˜ ant, build, compile, ˜config, debug, log, make, module, patch, redeploy, syntax, warnings,˜ OS concurrent, path, RedHat, Unix, windows, case

from multiclass to binary class for binary classification. We have used a conditional function that is used for converting multi classes to binary classes. The conditional functional is made of the following values and parameters.

Actual bugs = logical, enhancement, graphical user interface bugs.

Non bugs = other ten the actual bug class

Here the actual bug class in the binary classification is consists of three different classes (i.e. Logical, enhancement and graphical user interface bugs) while the non-bug class will contain bug reports that does not fall into any of the actual bug class category.

3.9 Selection Method of ML Learning Algorithms

In classification problem the selection of classifier is very much important and a very critical decision. This selection depend on many factors such as 1) size of a dataset, 2) Speed or training time, 3) Number of features , 4) type of data 5) complexity of classifier and so many more. However, different classifier have different set of requirement for their selection such as 1) If you have a limited and small dataset and your data is supervised, then base on machine learning theory it is better to use a classifier with high bias like Naive Bayes, 2) KNN and SVM are more conventional, and is suitable for small datasets, 3) Naïve Bayes also perform well in case of categorical input variables compared to numerical variable and so on. So there is no suitable classifier which covers all of the above factors of our dataset. Due to this reason mostly researchers have recommend two ways, the first one is review research papers of your area in which you are working and picked that classifier which are mostly used in literature and second way is to evaluate all the classifiers and then, make comparison on their performance and select that one which have return highest results.

After comprehensive literature we have selected 6 different classifiers which are mostly used by researchers. The classifiers are 1) Decision Tree (J48), 2) Random Forest, 3) K-Nearest Neighbors, 4) Gaussian Naïve Bayes, 5) Naive Bayes and 6)

Table 3.5: Algorithm Properties

Algorithm	Parameters or Properties
Decision Tree	J48, Kernel =sigmoid
Random Forest	max_depth=2, random_state=0
K-nearest Neighbor	n_neighbors =5
Gaussian Naive Bayes	Use default setting
Naive Bayes	Use default setting
SVM	Kernel =sigmoid

SVM. The below table represent the classifiers name and there parameter which we have used The main reason behind the selection of these six particular machine learning algorithms is that SVM, naïve Bayes, Decision tree, K-NN and Random forest are regularly used in the literature. For example, in the literature [42] used SVM and Decision tree, [43] used random forest, while [37], [23] used K-NN and naïve Bayes The Table 3.5 represent the classifiers name and there parameter which we have used.

3.10 Machine Learning Algorithms

Machine learning algorithms are one of the most powerful and efficient family of algorithms. Our bug's reports classification and triaging approach employs machine learning algorithm for classification purposes. Traditionally machine learning algorithms such as support vector machines (SVM), naïve Bayes (NB), decision tree (DT) and multinomial naïve Bayes(MNB) are widely used in the existing literature for bugs classification and triaging process. In our approach we used several machine learning algorithms. Basic reason behind the use of more algorithms for classification is to investigate the performance of these algorithms for bug's reports classification purpose. Another important reason behind the selection of machine learning algorithm is to perform a comparative analysis among the selected machine learning algorithms. Hence this comparison will lead us to make decision for selecting the most proper and suitable machine learning algorithms for classifying software bugs reports. Apart from this we compare performance of machine learning algorithms using different evaluation measures such as F-measure, Precision, recall and Accuracy.

3.10.1 Support Vector Machine (SVM)

Support vector machine commonly referred as SVM is a supervised machine learning algorithm that takes an input and produces a mapping function output using a labeled data instances [49]. SVM is considered as one of the most effective and powerful machine learning algorithm and has been widely used in different domains, such as text mining, image processing and facial recognition systems. SVM algorithm starts working by transforming the training set data into high dimensions. After the transformation process the algorithm starts search for finding the most effective hyper planes that divided the training set. Hyper plane is used to distinguish classes from each other. Support vector machine can work with both linear and nonlinear data classification. SVM provides powerful features such as linear classification approaches can be applied to data which is nonlinear in nature. Furthermore, it is considered effective classification algorithm than other classification algorithms in terms of performance. LIB support vector machine (LIBSVM) is the publicly available java based implementation which can be integrated into data mining tools like Weka [50].

3.10.2 Naïve Bayes

Naïve Bayes is another supervised machine learning algorithm used for classification of text and several other problems. Naïve Bayes algorithm uses conditional probability of Bayesian rule for classification purposes [51]. NB treats all the instances and attributes of training is a single entity and undertakes that all the attributes or data instances have equal importance for accurate classification. Naïve Bayes is very simple and effective machine learning algorithm which is wieldy used in different application areas such as text classification and text mining [52]. The performance of naïve Bayes is not effective for text classification problems due to several reasons. For example, naïve Bayes algorithm treats every attribute independent of other class attributes. Hence, this assumption of naïve Bayes algorithm results in low performance in text classification. However, due to simplicity and effectiveness it is still widely used in text classification problems.

3.10.3 Gaussian Naïve Bayes

Gaussian naïve Bayes is a variant of the naïve Bayes algorithm which is an ensemble classifier widely used in the classification problems [53]. GB is normally used to remove or Hindle the linear attributes in the naïve Bayes classifier. Gaussian naïve Bayes algorithm uses Gaussian distribution mechanism in order to represent the outcomes of the different classes based on the distribution probability.

3.10.4 K-Nearest Neighbour

K-nearest Neighbour is another supervised machine learning algorithm widely used in different classification problems such as text classification [54]. K-nearest Neighbour is also referred as K-NN. K-NN starts working by finding the K nearest elements in the data and uses these K nearest categories to weight the different elements in the document. Normally the performance of the K Nearest Neighbour algorithm depends on two parameters; these parameters are the similarity function for K and its value.

3.10.5 Decision Tree

Another supervised machine learning algorithm we use for classifying software bugs reports is decision tree. Decision tree resembles like a simple tree like structure, which contains leafs and nodes [55]. Normally, decision tree is also referred to as classification tree; classification tree learns different models and their associated functions that contain dependent and independent variables. Decision is very simple but yet very powerful classification mechanism and is widely used in different classification problems.

3.10.6 Random Forest

Random forest commonly referred as RF is an ensemble classifier that works in bagging and boosting mechanism. Random forest classifier uses a combination of

Table 3.6: Confusion matrix for evaluation

Total number of bugs	**Accurately Predicted bugs**		
	˜	Positive	Negative
	Positive	TP	FN
˜	Negative	FP	TN

different models that help in assigning votes to the most accurate and suitable input class [56]. Random forest consists of many classifiers that help in voting process for selecting the most suitable value for input class. Random forest has several advantages as compared to bagging or boosting, such as random forest handles outliers and proximities more effectively and accuracy as compared to bagging and boosting.

3.11 Performance Evaluation Measure

Traditionally classification model accuracy and performance evaluated using various performance evaluation measures, such as accuracy and precision. Normally a confusion matrix is used for performance evaluation of machine learning classifiers, which gives results about the actual and predicted classification done by classifier model. In order to evaluate the performance and effectiveness of our proposed approach, we use four widely used evaluation measures, i.e. accuracy, precision, recall and F-measure. These measures are obtained using a confusion matrix, the confusion matrix contains four values that true positive (TP) which means the actual positives values the classifier predicted, true negative (TN) values means actual negative values which the classification model predicted accurately. The other two values are FP and FN, false positive means the actual values are negative but the classification model predicted it positives. FN is positive values which the system has predicted negatively. Given Table 3.6 shows confusion matrix.

3.11.1 Accuracy Measure

Accuracy is well-established and widely used evaluation measure for assessing machine learning algorithms performance. Accuracy can be defined the ratio of correctly predicted elements to the total number of the elements present in a training set. Mathematically, accuracy measure can be calculated using the given formula 3.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

In order to compute the classification accuracy of our proposed software bugs classification technique, we use the given accuracy measure formula 3.2.

$$Accuracy(\%) = \frac{number\ of\ accurate\ classified\ software\ bugs}{total\ numebr\ of\ bugs/issues} * 100 \tag{3.2}$$

3.11.2 Precision

Precision is another measure we use for evaluating the performance of our proposed approach. Precision can be defined as the percentage of correctly classified bugs and the total number of bugs/issues which was assigned to particular class or category. To calculate precision we use a confusion matrix that contains both positive and negative values.

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

3.11.3 Recall

Recall is one of the most widely used evaluation measure for classification purposes. Recall can be defined is the ratio of correctly classified software bugs and actual number of software bugs of particular type. Mathematically Recall can be calculated as using the given formula 3.4.

$$Precision = \frac{TP}{TP + FN} \tag{3.4}$$

3.11.4 F-Measure

F-measure is one of the most widely used evaluation parameter for assessing the performance and effectiveness of any machine learning algorithm. F-measure considers both precision and recall by taking their harmonic mean. F-measure is considered one of the most powerful and effective evaluation measure used for performance evaluation of machine learning algorithms. F-measure can be calculated using the given formula 3.5.

$$Precision = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.5}$$

3.12 Extraction of Textual Feature from Bug Reports

In order to extract feature set against textual attributes, i.e. title, summary. To extract these features from preprocessed bug's reports we use term frequency and inverse document frequency (TF-IDF) scheme. TF-IDF term weighting scheme is used in majority of the software bug reports classification, bugs prioritization techniques. Mathematically, TF-IDF can be represented using the given formula 3.6.

$$TF - IDF_{ti,bj} = TF_{ti,bj} * (1 + log(\frac{1+B}{1+b(ti)})) \tag{3.6}$$

In the above formula B represents the total number of bug reports that are present in the corpus or dataset, while ti is the total number of occurrence of certain term in the specific bug report bj. While on the other hand (bt) is total number of issue reports in which a specific term (ti) has occurred.

3.13 Extraction of Categorical Features from Bug Reports

Categorical attributes are integral part of any software bugs and can play a crucial role in bugs classification and assignment process. In the existing literature various approaches considered categorical features for bugs classification. In our proposed technique we use categorical attributes for software bugs classification. These categorical attributes are product, component, version, priority and severity. To extract from categorical attributes of the bug report we use one hot encoding scheme [58] which converts categorical features into binary vectors. This binary vector of categorical attributes is used to train classification algorithm. Furthermore, the length of the binary vector contains the number of all possible values present in the categorical attributes. In the feature extraction process of categorical terms we use term frequency and inverse document frequency (TF-IDF), however, we apply TF-IDF on line level instead of applying it whole document. This is called line-IDF and use two binary values, i.e. 0 and 1.

3.13.1 Challenges in using the Categorical Attributes of the Bug Reports

Working and dealing with categorical variables is a quite challenging and time consuming task. We would like to share some of the challenges we faced while dealing with categorical variables. One of the main challenges dealing with the categorical attributes is that not every classifier can be applied on the categorical attributes. For instance, applying neural network (NN) or Convolutional neural network (CNN) may not be feasible and its results will not be fruitful. Another important challenge with categorical attributes is that it conveys less information as compared to numerical values. For example, Most of the algorithms (or ML libraries) produce better result with numerical variable. In python, library like "sklearn" requires features in numerical arrays. Another major issue with categorical attributes is that it requires encoding schemes like Onehot encoding or hamming encoder in order to convert it into vectors. We have used Onehot encoding scheme To convert the categorical attributes into vectors.

3.14 Proposed Approach Diagram

Our proposed approach works in several steps. In initial step software bugs reports are retrieved from Bugzilla and Google chromium bugs tracking system, in the second step preprocessing is applied and textual and categorical features are extracted. In the fourth step machine learning algorithms are applied for classifying software bugs into different classes. in the last step classification results are evaluated using four evaluation measures, such as F-measure, accuracy, precision and Recall. Figure 3.1 depicts an overview diagram of our proposed approach for bug reports classification.

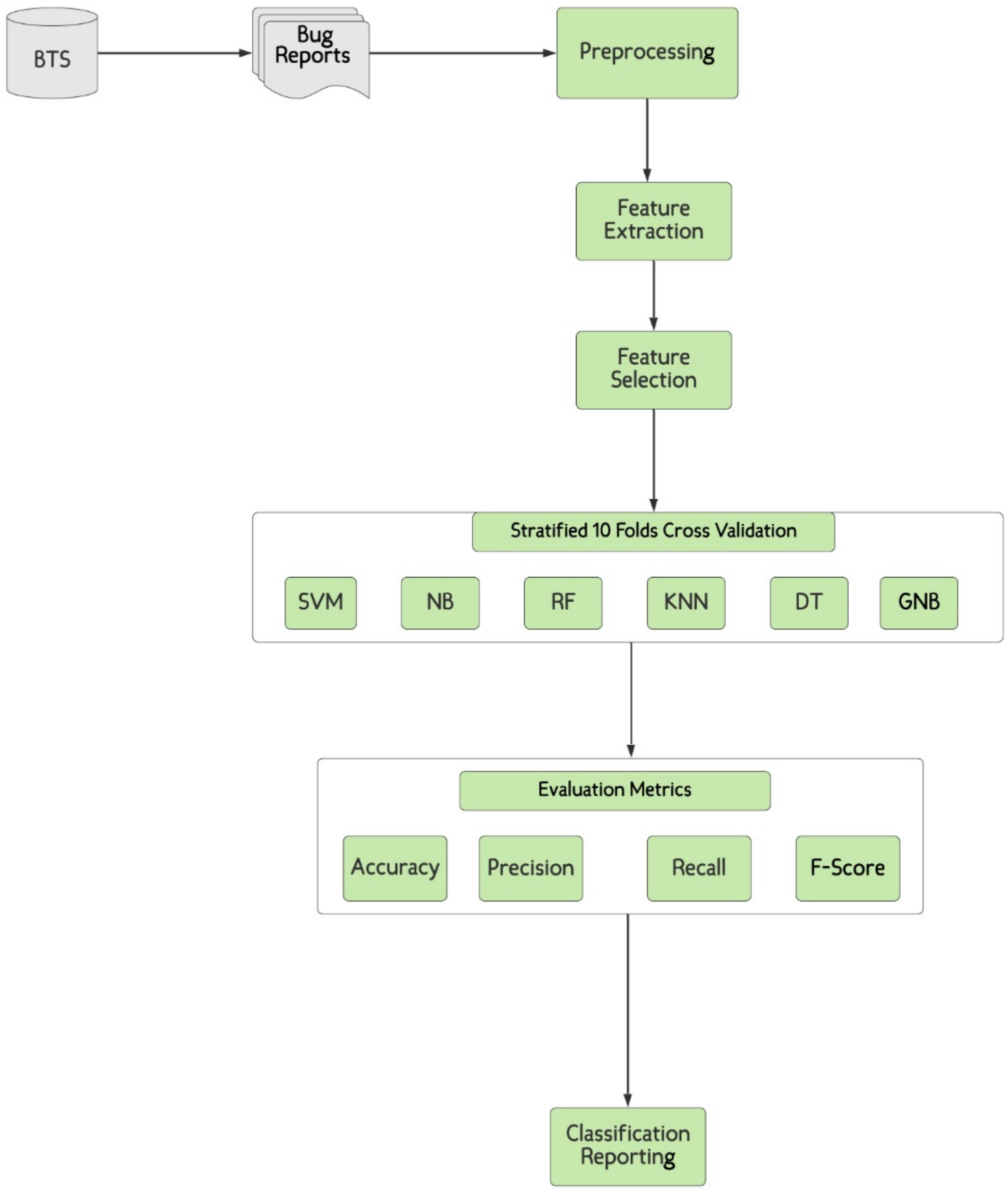


FIGURE 3.1: Methodology diagram

Chapter 4

Implementation and Results

4.1 Experimental Setup

Experiments were performed on bug's reports datasets using machine learning algorithm by implementing a comprehensive experimental setup. Experiments were performed on machine having Intel core i5 processor with 2.5 GHz, with installed main memory of 4 GB and Windows 7 operating system. All the implementation and pre-processing of software bug reports were done using python programming language and natural language processing toolkit (NLTK). We used the open source API of WEKA tool in Python programming language, which contains the implementation of all algorithms we used in this research work.
Our experimental analysis and evaluation consists of three main steps, i.e. datasets selection and processing, feature extraction and applying machine learning algorithms on pre-processed bug reports in order to classify software bugs in different classes. We extracted feature set from bug reports dataset; the feature extraction was done using Term Frequency-Inverse Domain Frequency (TF-IDF). TF-IDF is widely used technique for feature extraction in classification problems. Features are extracted on the basis of common occurrence of certain key words that appears in bug reports. The obtained results are evaluated using four evaluation measures that are F-measure, Precision, Recall and accuracy. We perform comparison of our proposed approach with other techniques in the literature closely related to

our approach. In the rest of this chapter we explain the obtained results of our proposed techniques.
The process which we have performed for the classification of bugs is given below:

4.2 Dataset

The first step is dataset selection. We have selected two different datasets of software bus reports that Firefox and Eclipse platform.

4.2.1 Firefox for Android Bugs

The Firefox for Android bugs dataset contains bug reports files against the different versions of Firefox browser for smart phones. The dataset contains 20 thousand reports filed in the Bugzilla bug tracking repository. We remove the duplicate bugs during the retrieval process from the Bugzilla bug tracking system.

4.2.2 Eclipse Platform Bugs

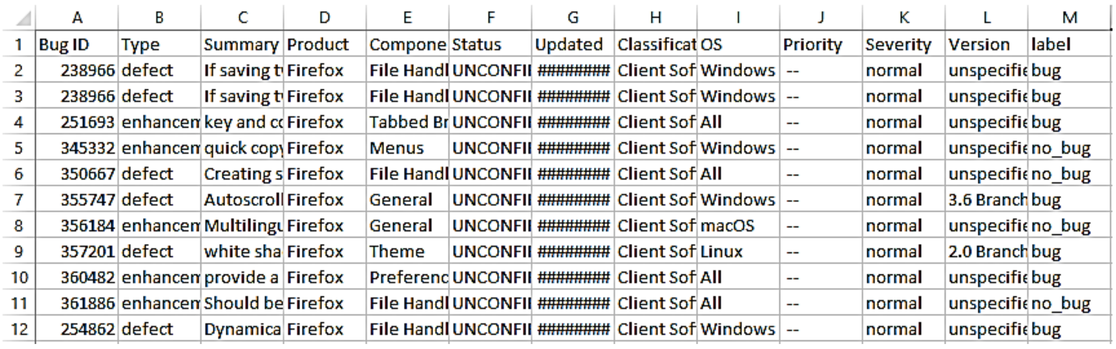
The Second Dataset we use for evaluating our proposed approach is Eclipse platform bug reports dataset, which also contains the same number of software bug reports.

4.3 Labeling

As the classification requires respective categories against bugs, in the software bug reports datasets we are using contains the respective categories. The dataset which we have used for experimentation purpose is already labeled. In the dataset some of the instances are unlabeled. In the pre-processing stage we have labeled them the unlabeled instances by comparing with the feature set which we have make from the bug reports. In order to make the datasets more consistent and uniform we are renaming the labels into five different labels. These labels are bugs, non-bugs, logical bugs, graphical user interface bugs and enhancement bugs.

4.3.1 Binary Classification

For binary Classification we have re-named labels of all the instances of the datasets (i.e. Firefox and Eclipse) into bugs and non-bugs. The dataset are somehow like this: 4.1:

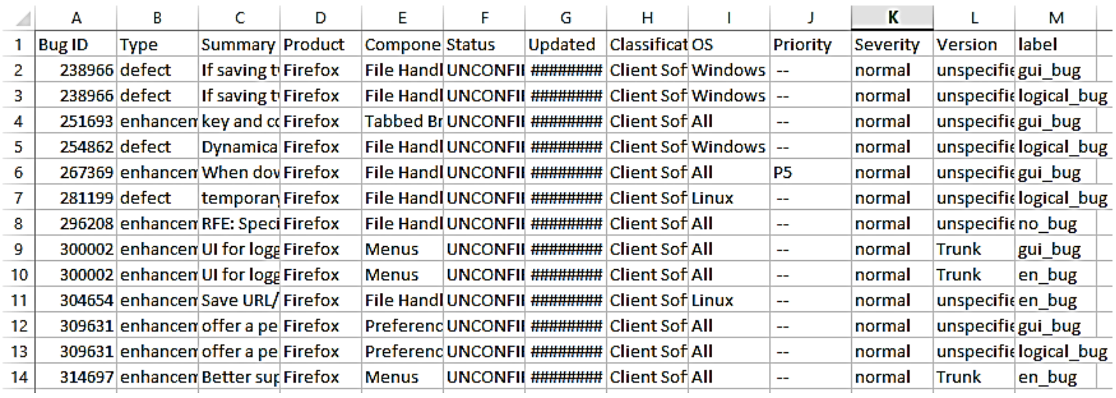


	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Bug ID	Type	Summary	Product	Compone	Status	Updated	Classificat	OS	Priority	Severity	Version	label
2	238966	defect	If saving t	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	bug
3	238966	defect	If saving t	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	bug
4	251693	enhancen	key and cc	Firefox	Tabbed Br	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	bug
5	345332	enhancen	quick copy	Firefox	Menus	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	no_bug
6	350667	defect	Creating s	Firefox	File Handl	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	no_bug
7	355747	defect	Autoscroll	Firefox	General	UNCONFII	#########	Client Sof	Windows	--	normal	3.6 Branch	bug
8	356184	enhancen	Multilingu	Firefox	General	UNCONFII	#########	Client Sof	macOS	--	normal	unspecifie	no_bug
9	357201	defect	white sha	Firefox	Theme	UNCONFII	#########	Client Sof	Linux	--	normal	2.0 Branch	bug
10	360482	enhancen	provide a	Firefox	Preferenc	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	bug
11	361886	enhancen	Should be	Firefox	File Handl	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	no_bug
12	254862	defect	Dynamica	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	bug

FIGURE 4.1: Binary classification labeling

4.3.2 Multi-Class Classification

For Multi-Class Classification we have re-named the label all the instances of a datasets into four different classes such as 1) Logical bugs, 2) GUI bugs, 3) Enhancement bugs and 4) Non bugs. The datasets are somehow like this: 4.2:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Bug ID	Type	Summary	Product	Compone	Status	Updated	Classificat	OS	Priority	Severity	Version	label
2	238966	defect	If saving t	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	gui_bug
3	238966	defect	If saving t	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	logical_bug
4	251693	enhancen	key and cc	Firefox	Tabbed Br	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	gui_bug
5	254862	defect	Dynamica	Firefox	File Handl	UNCONFII	#########	Client Sof	Windows	--	normal	unspecifie	logical_bug
6	267369	enhancen	When dov	Firefox	File Handl	UNCONFII	#########	Client Sof	All	P5	normal	unspecifie	gui_bug
7	281199	defect	temporan	Firefox	File Handl	UNCONFII	#########	Client Sof	Linux	--	normal	unspecifie	logical_bug
8	296208	enhancen	RFE: Speci	Firefox	File Handl	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	no_bug
9	300002	enhancen	UI for logg	Firefox	Menus	UNCONFII	#########	Client Sof	All	--	normal	Trunk	gui_bug
10	300002	enhancen	UI for logg	Firefox	Menus	UNCONFII	#########	Client Sof	All	--	normal	Trunk	en_bug
11	304654	enhancen	Save URL/	Firefox	File Handl	UNCONFII	#########	Client Sof	Linux	--	normal	unspecifie	en_bug
12	309631	enhancen	offer a pe	Firefox	Preferenc	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	gui_bug
13	309631	enhancen	offer a pe	Firefox	Preferenc	UNCONFII	#########	Client Sof	All	--	normal	unspecifie	logical_bug
14	314697	enhancen	Better sup	Firefox	Menus	UNCONFII	#########	Client Sof	All	--	normal	Trunk	en_bug

FIGURE 4.2: Multi class classification label

4.4 Preprocessing of Bug Reports

First step in the experimentation and analysis phase of our proposed technique is the preprocessing step. Initially we retrieved software bug reports from tracking systems (BTS). After the Firefox and Eclipse dataset retrieval process from their respective bug tracking repositories we applied standard natural language processing techniques for preprocessing and cleaning of software bug reports dataset. First we ordered all issue reports in ascending order using bug IDs. Then we segment all the instances in all bug's reports in sentence level. After the segmentation step tokenization process is applied to tokenize software bug reports into tokens. Once the tokenization process completes we remove stop words from the bug reports. Stop words are those terms that occur frequently in text and have no effect on the overall semantic of the text. However, we apply a stopping mechanism for removing stop words, for example we do not remove words like "does not", "not", "cannot" etc. because these terms have special meaning in a bug report. Stop word removal process is followed by feature set extraction process. Features are extracted from bug reports based on different keywords that occur in software bug reports. We use a taxonomic hierarchy for feature extraction. Term Frequency-Inverse Domain Frequency (TF-IDF) is used to extract feature set for five different classes. We use total of five classes for classifying software bug reports which includes actual bugs and non-actual bug classes, the other three classes are logical, GUI and enhancement bug reports classes. Main reason behind the use of varying number of classes is to evaluate and investigate the impact of the classifier performance and to suggest the number of classes which can be used to correctly classify software bugs into different classes.

4.5 Preprocessing Steps

Before performing the classification we have to preprocess the textual data. There are few parameters that needed to be cleaned form noise such as title of the bugs.

The stop words removal and stemming is done on different parameters. Let's discuss it step by step.

4.5.1 Stop Words Removal

In English language the words like, "is, the, a, which, at, in etc." is often found in multiple sentences. Therefore, their removal is necessary to get the unique terms from titles. To remove stop words from titles of bugs, we have to use the NLTK library because it contains list of stop words.

4.5.2 Lemmatization

The stemming and lemmatization both techniques are used to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. However, there is some difference in these two techniques. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. We have used the lemmatization technique for reducing the words into its base form without altering it semantic or structure. However, lemmatization requires a dictionary to be carried for deployment, while stemming does not require any dictionary to be carried for the deployment purpose.

4.5.3 Noise Removal

Removing of noise from data is important because it can affect the accuracy of classification. We have remove noise from different parameter (i.e. title, description and summary etc.) of a datasets. Therefore, we have removed all of these unnecessary punctuations by using NLTK library.

4.5.4 Vectorization

After performing preprocessing steps on te Firefox and Eclipse bug reports dataset, we have used Term Frequency and Inverse Domain Frequency (TF-IDF) technique to represent the textual attribute of our instances. In this technique we have calculated the TF-IDF score for every word of a parameter. As the number of words in parameter is vary in length so we have considered the average word per instance of the datasets.

4.6 Classification

Generally, we have performed two different types of classification, i.e. Binary classification and multi-class classification. For both type of classification we have evaluated different attributes of Firefox and Eclipse bug reports datasets such as: 1) Textual attributes, 2) Categorical attributes and combination of both categorical and textual attributes. For the classification we have used six different Classifiers which are given below.

1. Support Vector Machine (SVM)
2. K-Nearest Neighbors (KNN)
3. Naive Bayes (NB)
4. Gaussian Naive Bayes (GB)
5. Decision Tree (DT)
6. Random forest (RF)

4.7 Bug Reports Attribute Wise Experiments

We performed experiments using three different combinations of the bug reports attributes. These attribute are textual, categorical and combination of both textual and categorical attributes. We develop three different classification models for assessing the performance and effectiveness of our approach.

We have used stratified 10 fold cross validation, which is the extension of regular k-fold cross validation but specifically for classification problems where rather than the splits being completely random, the ratio between the target classes is the same in each fold as it is in the full dataset. We repeated and run our experiments for 5 times.

4.8 M1 Model: Classification Model using Bug Textual Attributes

The first classification model only uses textual attributes of software bug reports extracted from two datasets. The basic intuition behind the use of only textual attributes of bug reports is to assess their significance in the classification and prediction process. We use natural language processing technique to extract textual information related to bug title description and summary using TF-IDF. It is evident from the literature review that textual attribute is an essential component of any software bug reports. We report the performance of textual attributes in following figure 4.3. Furthermore, the performance of the textual attribute on our selected algorithms is evaluated using precision, F-measure, Recall and accuracy evaluation measures.

4.8.1 Firefox and Eclipse Datasets Results on Model M1:

First we have evaluated the Firefox bug reports dataset and performed two type of classification. I.e. binary classification which consists of actual software bugs class and non-bugs class. While multi classification contains four classes which include GUI bugs, non-bugs, enhancement and logical bugs.

In binary classification we selected first two classes for classification, which are actual bugs and non-bugs class. Furthermore, we experimented with three different combinations of bug's reports attributes. These combinations are categorical attributes, textual attributes and both textual and categorical combined. Figure 4.3 depicts the results of different machine learning algorithms on the Firefox dataset when textual attributes from the software bugs reports dataset are used for classification. Figure 4.3 depict the classification results of textual attributes of the software bugs, Support Vector Machine, K-NN and Decision trees algorithm achieves better results as compared to Naïve Bayes, GB and Random Forest. SVM attained 83% score in terms of F-measure. On the other hand, Random forest was the poorly performing algorithm that produced less the 65% score in terms of F-measure score. Gaussian Bayes is another algorithm that performed poorly when only textual attributes of Firefox software bug reports is used for classification. However, GB attained 67% F-measure score for classifying software issue reports into binary classes (i.e. Actual bugs and non- bugs). In case of multi class classification we have evaluated the performance of machine learning algorithms on only textual attributes of the software bug reports. The basic reason behind the use of more number of classes is to evaluate and suggest suitable number of classes that can be used for bug reports classification. Figure 4.3 describes the obtained results of our six selected machine learning algorithms when only textual attributes from the Firefox bugs reports dataset is used. From figure 4.3 it very clear and obvious that most of the algorithms yielded varying results in terms of different evaluation measures. For instance SVM achieved an F-score of 0.53, followed by K-NN and RF with an F-score of 0.52 and 0.50 respectively. On the other hand Naïve Bayes, GB and Decision Trees classifiers yielded very moderate results on the evaluation measure (i.e. F-score). One of the main reasons behind the moderate performance of our selected machine learning algorithms is the more number of classes as compared to that of binary classification.

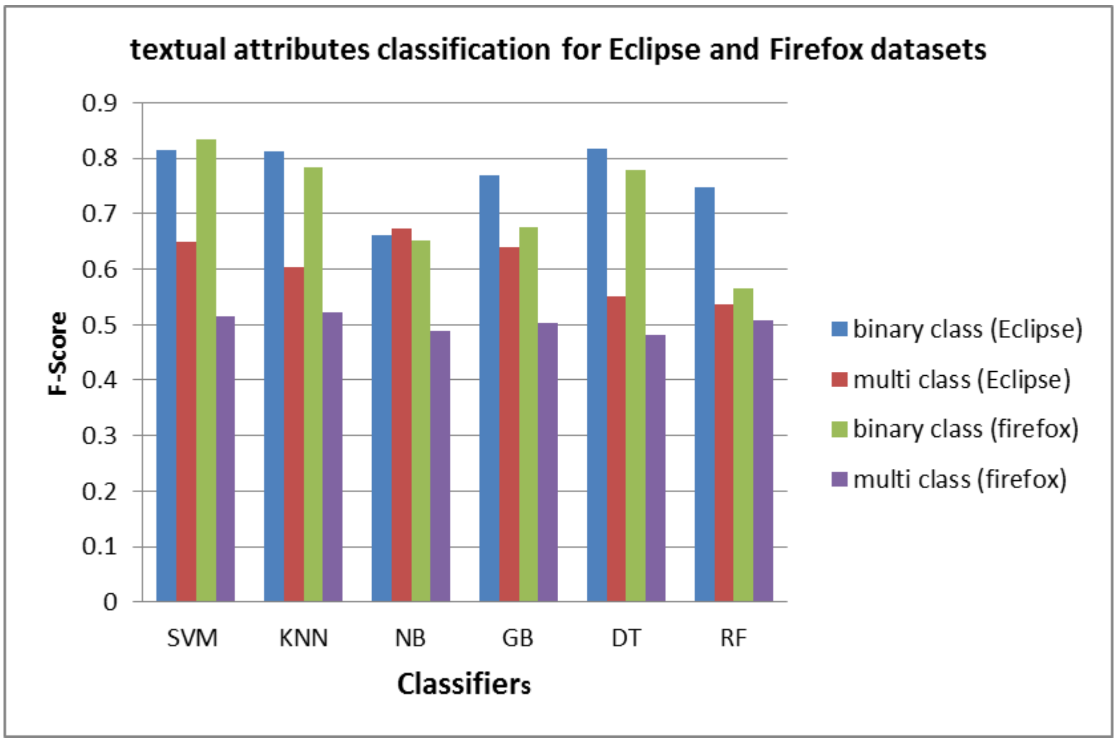


FIGURE 4.3: binary classification and multi Classification results of Firefox and Eclipse datasets using textual attributes

Eclipse platform is the second bug's dataset we use for demonstrating the performance of our proposed approach. We perform exactly the same experiments on the eclipse dataset as we performed on the Firefox software bugs dataset. We classify eclipse software bug reports using binary classification and multi classification. The binary classification contains two classes which are actual software bugs class and non-bugs class. While the multi classification contains four numbers of classes, i.e. actual bugs, graphical user interface bugs, logical bugs and enhancement or improvement related issues. Figure 4.3 portrays the performance of our selected machine learning algorithm performance on the eclipse bugs dataset when only textual attributes are used for binary and multi classification.

In the case of binary classification we classify software bugs into actual bug and non-bug classes. Figure 4.3 shows the obtained results of machine learning algorithm for binary classification. In the case of textual attributes of eclipse software bugs reports support vector machine, K-nearest Neighbour, GB and decision tree all obtains better results as compared Naïve Bayes and Random Forest. SVM and K-NN were the stand out performers yielding an F-score of 0.78, 0.79, and 0.77

respectively. In case of precision and recall SVM, KNN and decision tree obtained better results as compared to the Naïve Bayes and Random Forest classifiers. Furthermore, Naïve Bayes achieved relatively poor results on all the evaluation measures as compared to the rest of classifiers. Naïve Bayes achieved F-score of 0.71 and accuracy score of 0.67. Although most of our selected classifiers achieved better results except Naïve Bayes for binary classification on Eclipse dataset textual attributes.

Figure 4.3 also depicts the classification results of our six selected machine learning algorithms for classifying software bugs reports in to different categories based on the textual attributes of the bug reports. For multi classification we use four classes for classifying software issues. In case of multi classification most of the classifiers produced mixed results when textual attributes of the software bugs reports were used. For instance, SVM, Naïve Bayes and GB produced F-score of 0.66, 0.67 and 0.67 respectively. While in case of accuracy SVM, K-NN, and Gaussian Bayes performs better than random forest and decision tree that attained an accuracy of 0.56 and 0.58 respectively.

4.9 M2 Model: Classification Model using Categorical Attributes

The second classification model is related to categorical attributes of the software bug reports. We use several categorical attributes from software issue reports, these categorical attributes are “product”, “Platform”, “priority”, “severity” and “version” of reported software issues. Categorical attributes/features are extracted from the bug reports and converted into binary vectors using “one hot encoding” which is most widely and effective technique for encoding categorical features. The basic intuition behind the use of only categorical features of bug reports is to analyze the impact of categorical attributes in software issues reports classification process. Categorical features are considered an integral part of any software bug reports and it provides variety of information that can be beneficial in software issues classification process. We report the M2 model results in the following figures. Which is evaluated using four performance evaluated measures, i.e. F-measure, precision, Recall and accuracy.

In order to use the categorical attributes of software bug reports we first encode the categorical attributes to binary vectors. One hot encoding scheme is used to encode the categorical attributes of software bug reports into binary vectors.

4.9.1 Firefox and Eclipse Dataset Classification Results

From figure 4.4 it can be observed that support vector machine, KNN and decision tree algorithms performs better than naïve Bayes and Gaussian Bayes. SVM has an F-score of 0.78, and then followed by KNN with 0.75 and decision trees with 0.74. In case of accuracy SVM, KNN and DT performed better than the rest of algorithms (i.e. naïve Bayes, Gaussian Bayes and Random forest). On the other hand Naïve Bayes and Gaussian Bayes (GB) achieved similar F-score of 0.58 each. Random Forest performed better than the Naïve Bayes and Gaussian Bayes which produced an F-score of 0.61. The binary classification results of categorical attributes are quite better as compared to multi class classification. Hence, categorical features can be used in the classification process of software bug reports. However, selection of appropriate categorical attributes and their classification performance can vary depending upon the dataset. Figure 4.4 also shows the multi classification of only categorical attributes of firebox bug reports dataset. We use classify the software bugs reports into four classes (non-bugs, logical bugs, GUI bugs, and enhancement bugs) based on the categorical attributes. In case of multi classification only SVM classifier achieved better results as compared to the rest of the classifiers. SVM achieves an f-score 0.76, and then followed by KNN with 0.51 F-score. Furthermore, SVM and KNN produced results between 0.520 to 0.691 in terms of accuracy, recall and precision. Naïve Bayes, GB, DT and RF achieved poor results on all the evaluation parameters, the performance of RF; GB achieves an F score of 0.32, 0.35 and 0.318 respectively. The same was the case with other evaluation measures for GB, DT and RF classifier. Hence,

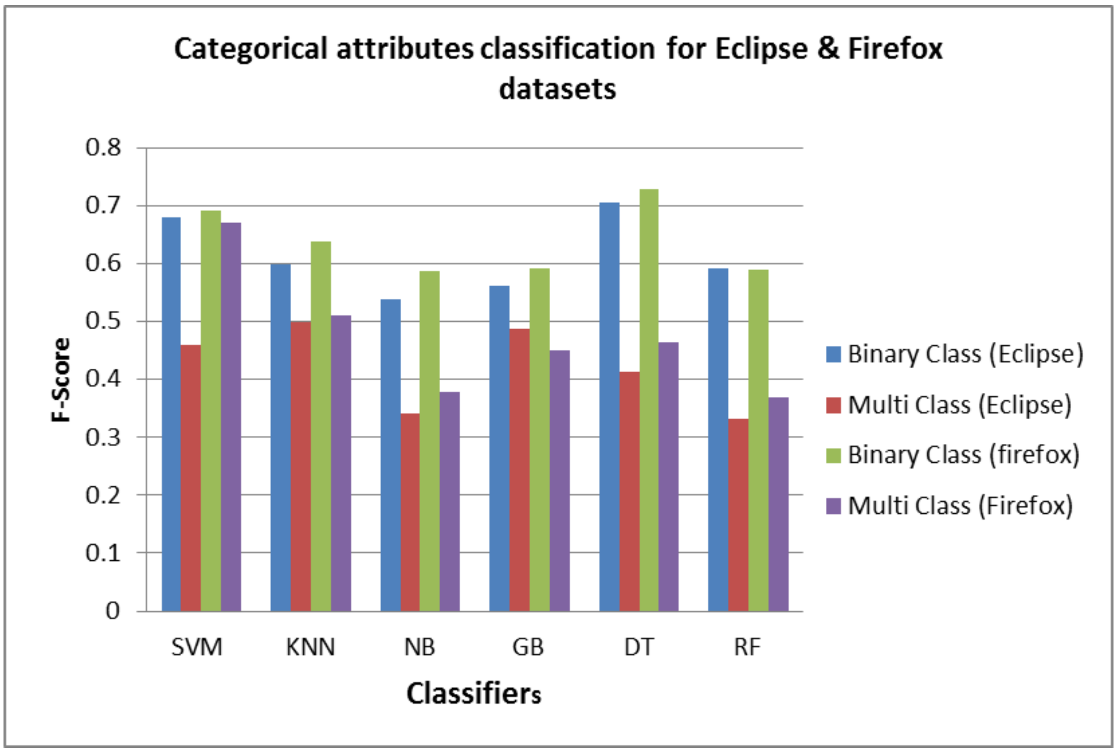


FIGURE 4.4: Classification results for Firefox and Eclipse dataset using categorical attributes of software issue reports

the classifiers results indicates that using solely using the categorical attributes for bugs reports classification is not good option and it does not yield effective results. Two reasons are behind this poor performance of machine learning classifiers such as more number of classes, less effectiveness of bug's reports categorical attributes. Hence, it will not be feasible to use the categorical features for classifying software bug reports into different categories.

We performed the same experiment as we did on the Firefox bug reports dataset; we extract categorical attributes of software bugs reports from eclipse platform datasets and encode them using one hot encoding method. Figure 4.4 shows the binary classification of categorical attributes of eclipse bugs reports dataset. Here all the classifiers achieved moderate results on the eclipse bugs dataset. For instance SVM achieved highest F-score of 0.60, followed by naïve and GB and DT with 0.53, 0.56 and 0.566 respectively. In case of accuracy all the algorithms achieved scores between 0.64 and 0.51. SVM produced 0.65 and 0.69 precision and recall score. From the experimental analysis it is very clear that most of the classifiers generated average results when only textual attributes of the bug's reports were used. Bugs classification using only textual attributes of software bugs reports is not much that effective , because software bug reports contains other important attributes other than textual. Hence, by combing both textual and categorical attributes can be much effective then using only textual or categorical attributes of the bug reports.

Figure 4.4 shows the results of our selected classifiers on categorical attributes for multi classification. Most of the algorithms perform poorly on categorical attributes for multi classification. For instance the highest performing classifiers were K-NN and Decision trees in terms of F-measure that was 0.34 and 0.33 respectively. The other most effective classifiers were Naïve Bayes and GB that yielded an F-score of 0.30 and 0.25 respectively. This experimental result indicates that using categorical attributes alone for bug's reports classification is not a feasible way and effective way due to different reasons. Categorical features do not contain that much information and rich features that can be used for classification. Although, categorical features are considered essential part of any bug report.

4.10 M3 Model: Classification Model using Textual and Categorical Attributes

Our third classification model makes use of both categorical and textual attributes of software bug reports. In this model we use bug title, bug description and summary from textual attributes and product name, priority, severity and other attributes from the categorical fields of software issue reports. The main reason behind the selection of both categorical and textual attributes is to analysis the performance by combining these different attributes.

4.10.1 Classification Results of Firefox and Eclipse Dataset on M3 Model

Model M3 in our proposed approach make use of the both the categorical and textual attributes of the software bugs reports. The main reason behind combining both the textual and categorical attributes of software bug reports is to evaluate their effectiveness in the classification problem. In binary classification we classified Firefox and Eclipse dataset bug reports into two main classes, which are actual bug class and non-bugs class. Figure 4.5 depicts the classification results of software bug reports using both categorical and textual attributes of the bug reports.

Figure 4.5 that shows the binary classification results of our employed machine learning algorithms, most of the algorithms achieved significantly higher results in terms of all the evaluation measures (i.e. precision, recall, accuracy and F- score) we used for performance evaluation. For instance, SVM, KNN and decision tree algorithms achieved an F-score of 0.84, 8.77 and 0.76. Then followed by random forest and naïve Bayes which produced an F-score of 0.74, 0.73 respectively. The only algorithm that performed slightly poor then the rest of classification algorithm was Gaussian Bayes with an F-score of 0.69. In case of Precision, Recall and Accuracy SVM, K-NN and Decision trees achieved the highest results. Among these three algorithms SVM achieved the highest precision and accuracy of 0.85 and 0.84. The third best performing algorithm was KNN with 0.77 and 0.824 precision and accuracy.

Form the experimental results of using both the categorical and textual attributes of bug reports, we observed that by combining the textual and categorical attributes significantly Improves the performance of classification. There are several reasons behind the effective results of classification in the use of both categorical and textual attributes. Both the categorical and textual attributes are very important for classifying software bug reports into different classes. Hence, using both categorical and textual attributes are the perfect candidates for binary classification of software bugs reports. In model M3 we also performed the multi

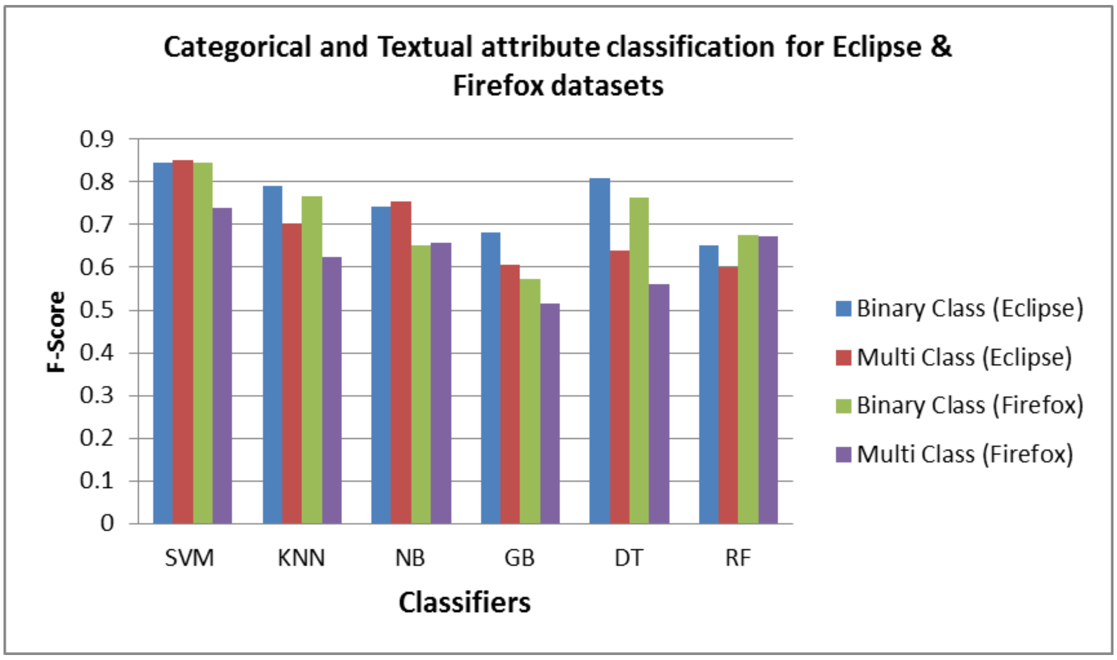


FIGURE 4.5: Textual and Categorical classification results on Firefox and Eclipse bug reports datasets

classification of software bugs reports; in multi classification we classify software bugs into four classes using four classes, such as logical bugs, GUI bugs and enhancement bugs. Figure 4.5 shows classification results of six different classifiers. In multi classification process most of the classifiers performed well on the Firefox dataset. However, their results were lower than that of the binary classification. In Figure 4.5 results SVM, Naïve Bayes and Random Forest achieved an F-score of 0.73, 0.65 and 0.67 respectively. Rest of the classifiers (i.e. K-NN, GB and Decision trees) achieved an F-score between 0.62 and 0.56. In case of the precision and recall support vector machine achieved the highest results of 0.78 and 0.79 respectively. In the multi classification Gaussian Bayes performed poorly in all the evaluation measures.

Multi classification using both the categorical and textual attributes of Firefox software bug reports achieved lower results as compared to the binary classification. More number of classes can have impact on the overall performance of the classification. another, important reason for the lower performance of multi classification than binary classification is that extracting class wise feature set from textual attributes is very challenging task due to the nature of text in the bug reports.

Another dataset we use for evaluating our proposed approach is Eclipse platform dataset. Figure 4.5 shows binary classification of eclipse bugs reports dataset. Both textual and categorical attributes of bug reports are considered for classifying the issues into two classes such as actual bugs and non-bugs. Most of the classifiers performed well and yielded better results except Gaussian Bayes and Random forest. For instance, support vector machine, KNN and decision tree obtained an F-score of 0.84, 0.79 and 0.80 respectively. Naïve Bayes, GB and random forest lower F-score (i.e. 0.74, 0.68 and 0.65) as compared to the rest of the classifiers. In case of accuracy SVM, KNN Naïve Bayes and decision tree classifiers better results as compared to Gaussian Bayes and random forest. In terms of the precision and recall most of the classifiers achieved significantly high result except Gaussian Bayes and random forest which scores precision of 0.65 and 0.68 and recall of 0.66 and 0.68.

Hence, the experimental results on eclipse dataset for binary classification with both categorical and textual attributes of software bug reports, demonstrated that all the classifiers achieved high performance on all the evaluation measures.

Figure 4.5 shows the attained results of the selected machine learning classification algorithms for multi classification of the Eclipse bug reports dataset. We used four number of classes for classifying Eclipse software bug reports, these classes are non-bugs and three sub categories of actual bug's class, that are logical bugs, GUI bugs and enhancement related bugs. For multi classification KNN, Naïve Bayes and SVM obtained highest results in terms of accuracy, precision, recall and F-score. The F-score of these classifiers are 0.85, 0.71 and 0.75. SVM, KNN and naïve Bayes recorded the highest accuracy of 0.88, 0.75 and 0.78 respectively. On the other hand Gaussian Bayes, decision tree and random forest scored varying results on all evaluation measures. For instance GB, DT and RF achieved F-score of 0.62, 0.65 and 0.6o respectively. The obtained results of classifiers on the categorical and textual attributes of eclipse bugs indicates that using the combination of both categorical and textual attributes of software bugs provides effective results as compared to using these attributes alone.

4.11 Class Wise Comparison

To evaluate our proposed mechanism of software bug reports classification, we performed different types of experiments and evaluation techniques. For example, we tested and evaluated our approach using varying number of classes, i.e. total number of classes is 4, and we vary the number of classes during experiments from 2-4 and observed the behaviour and performance of selected classifiers on Firefox and Eclipse bugs reports dataset. We develop 3 classification models based on the categorical and textual attributes of the bug reports. Furthermore, the class wise evaluation shows the number of classes that can yield better performance in terms of accuracy, F-measure, precision and recall without degrading the classification performance.

4.11.1 Classification Performance using Two Classes

We evaluate the number of classes and its impact on classification of software bugs reports using machine learning algorithm. In model 1 we classify software bugs into two classes that are actual bug's class and non-bugs class. Actual bug's class contains those bugs that occurred within the software. While those classes which does not directly affects the functionality of the software system are non-bugs. Form experimental analysis performed on the two bug's datasets (i.e. Firefox and Eclipse platform) we observed that all the machine learning algorithms performed effectively and efficiently on both Firefox and Eclipse datasets. We used different combinations of categorical and textual attributes for classification, such as using textual and categorical attribute separately and combining textual and categorical attributes. The combination of textual and categorical attributes yielded better results in terms of evaluation measures. Binary Classification results for textual, categorical and combination of textual and categorical attributes of software bug reports from both Firefox and Eclipse datasets is depicted in the following figures 4.6.

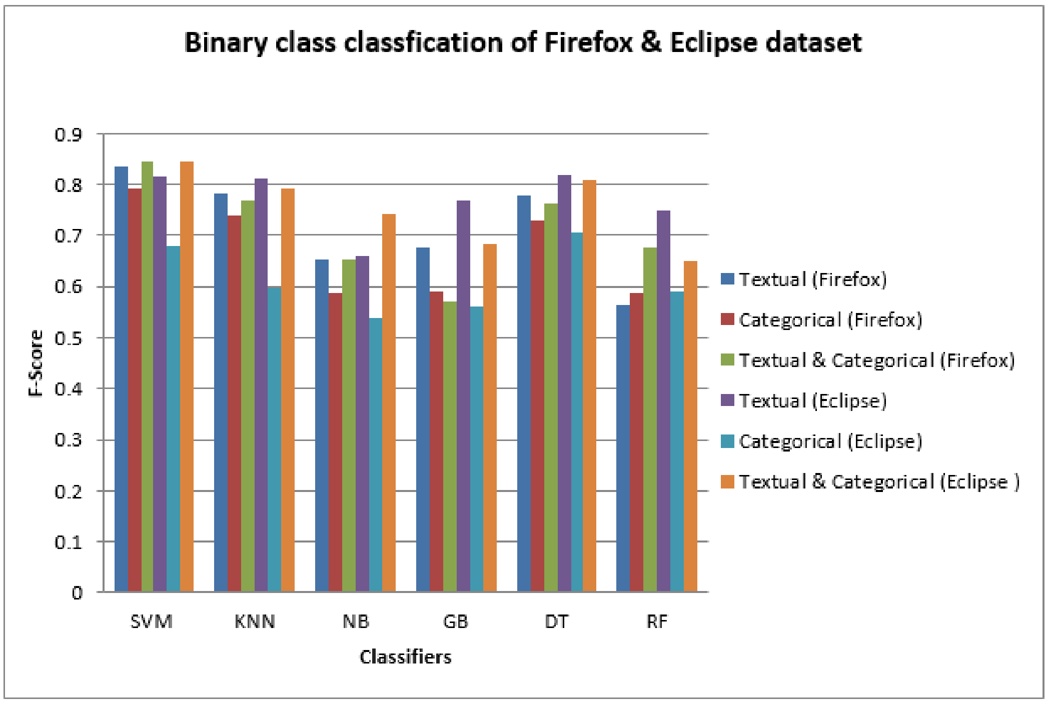


FIGURE 4.6: Binary class Classification results of Firefox and Eclipse dataset in binary classes

4.11.2 Multi class Classification Results using Four Classes

The basic intuition behind the use of varying number of classes is to evaluate their performance and suitability for classification. The following figure shows the obtained results for classification in terms of four evaluation measures such as F-measure, Precision, recall and accuracy. However, we only report the F-measure score in the graphs in order to make it more simple and readable. We used four numbers of classes (i.e. actual bugs, logical, graphical user interface and enhancement bugs) for classification. In model M3 we performed experiments using textual and categorical attributes of the bug reports. We observed form obtained results that when textual and categorical attributes were used combined generates better results as compared to the use of textual and categorical separately. The multi classification results of textual, categorical and combining the textual and categorical attributes of software bug reports are shown in figures 4.7

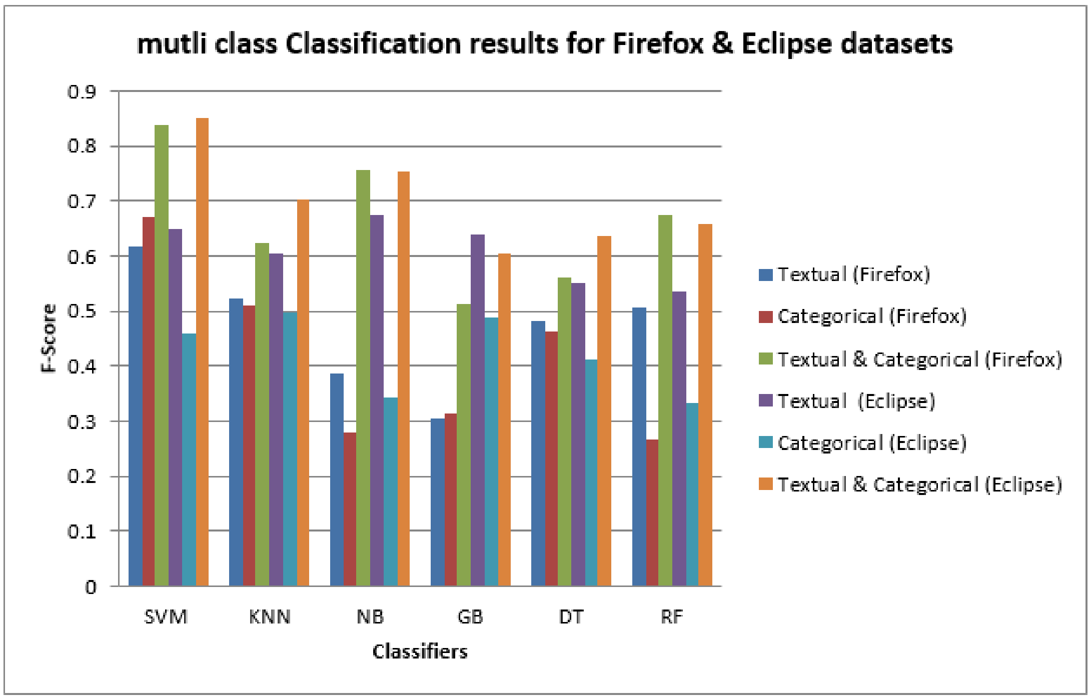


FIGURE 4.7: Multi class classification results on Firefox and Eclipse dataset

4.12 Algorithm Wise Comparison

In this work we performed experiments using six different machine learning algorithms, such as Support Vector Machine, Naïve Bayes, Decision Trees, Random Forest and Gaussian Bayes. We compare the performance of these classification algorithms on two different software bug reports datasets (i.e. Firefox and Eclipse bugs reports). We evaluate the performance and applicability of these six machines learning in order to be used for software issue reports classification. The basic motivation behind the use of the six machine learning algorithms for software bug reports classification is to evaluate the effectiveness and prediction performance of the different machine learning algorithms. Most of the machine learning classification algorithms achieved betters results when we use textual and categorical attributes of bug reports combined, for instance, support vector machine, random forest and decision and Naïve Bayes were the best performing classifiers in terms of precision, accuracy, recall and F-measure. However, when we used textual and categorical attributes separately, in such cases only support vector machine (SVM) is the best performing machine learning algorithm in terms performance on Firebox

Table 4.1: classification algorithms comparsion for binary classification on Eclipse dataset

Classifiers name	Textual				Categorical				Textual & Categorical			
	Acc	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**
SVM	0.882	0.898	0.852	0.814	0.636	0.646	0.696	0.59	0.856	0.862	0.856	0.844
KNN	0.818	0.856	0.818	0.812	0.516	0.51	0.516	0.498	0.802	0.854	0.802	0.792
Naïve Bayes	0.666	0.67	0.666	0.66	0.542	0.542	0.542	0.538	0.752	0.76	0.752	0.742
Gaussian Bayes	0.67	0.774	0.77	0.768	0.57	0.572	0.57	0.562	0.652	0.662	0.64	0.682
Decision tree	0.817	0.826	0.818	0.818	0.572	0.568	0.572	0.566	0.808	0.822	0.818	0.808
Random forest	0.754	0.756	0.754	0.748	0.53	0.498	0.5	0.492	0.808	0.68	0.678	0.65

Table 4.2: comparison of classifiers performance on Firefox dataset for binary classification

Classifiers name	Textual				Categorical				Textual & Categorical			
	Acc	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**
SVM	0.854	0.866	0.854	0.834	0.792	0.796	0.792	0.792	0.852	0.854	0.852	0.844
KNN	0.788	0.828	0.788	0.784	0.748	0.776	0.748	0.738	0.778	0.824	0.778	0.768
Naïve Bayes	0.671	0.652	0.662	0.652	0.592	0.598	0.592	0.586	0.66	0.664	0.66	0.652
Gaussian Bayes	0.776	0.576	0.676	0.676	0.596	0.602	0.596	0.592	0.576	0.578	0.576	0.572
Decision tree	0.782	0.792	0.782	0.778	0.728	0.738	0.728	0.728	0.766	0.774	0.766	0.762
Random forest	0.568	0.568	0.568	0.566	0.592	0.68	0.592	0.588	0.666	0.671	0.665	0.676

Table 4.3: comparison of classifiers performance on Eclipse dataset for multi-class classification

Classifiers name	Textual				Categorical				Textual & Categorical			
	ACC	**PR**	**RE**	**FM**	**ACC**	**PR**	**RE**	**FM**	**ACC**	**PR**	**RE**	**FM**
SVM	0.682	0.576	0.672	0.65	0.264	0.24	0.264	0.2	0.882	0.876	0.882	0.852
KNN	0.64	0.626	0.64	0.604	0.322	0.344	0.322	0.298	0.75	0.728	0.75	0.702
Naïve Bayes	0.596	0.588	0.616	0.674	0.27	0.276	0.27	0.242	0.784	0.786	0.784	0.754
Gaussian Bayes	0.618	0.612	0.618	0.639	0.3	0.308	0.3	0.288	0.626	0.618	0.626	0.606
Decision tree	0.564	0.55	0.564	0.55	0.324	0.34	0.324	0.312	0.652	0.64	0.652	0.638
Random forest	0.59	0.588	0.59	0.536	0.246	0.266	0.246	0.22	0.608	0.616	0.608	0.6

Table 4.4: classifier performance on Firefox dataset for multi classification

Classifiers name	Textual				Categorical				Textual & Categorical			
	Acc	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**	**Acc**	**PR**	**RE**	**FM**
SVM	0.576	0.576	0.576	0.516	0.608	0.698	0.608	0.67	0.729	0.786	0.79	0.738
KNN	0.552	0.53	0.552	0.522	0.536	0.526	0.536	0.51	0.658	0.636	0.658	0.624
Naïve Bayes	0.404	0.42	0.44	0.388	0.322	0.318	0.322	0.278	0.694	0.696	0.694	0.658
Gaussian Bayes	0.318	0.316	0.318	0.304	0.326	0.32	0.326	0.314	0.53	0.526	0.63	0.514
Decision tree	0.496	0.484	0.496	0.482	0.478	0.466	0.478	0.464	0.576	0.558	0.576	0.56
Random forest	0.506	0.504	0.506	0.508	0.318	0.308	0.318	0.268	0.61	0.612	0.61	0.674

and Eclipse bugs reports dataset. The following table 4.1 1and 4.2 depicts the detail performance comparison of machine learning classifiers for eclipse and Firefox dataset for binary classification. Table 4.3 and 4.4 shows the detail performance of the machine learning algorithms we used in this work. It also compares the results of the classifiers for Eclipse and Firefox dataset. Moreover, these results are based on multi class classification.

4.13 Attributes Wise Performance Comparison

In our work we have also performed attribute wise comparison of software bug reports. The basic intuition behind attributes wise comparison is to evaluate the performance of different bug report attributes that can help in achieving better performance and accurate results in bug reports classification process. We performed experiment on different combinations of our selected attributes (i.e. title, description, summary, expert comments, severity, priority, product, version and product). In order to determine the best and suitable software bug reports that can yield better performance results we have applied a wrapper (i.e. forward selection) subset evaluation and selection technique. The basic intuition behind the use of wrapper technique is to determine the best performing attributes (i.e. textual and categorical) for the software issue reports classification and training.

In order to identify the effective subset attributes we used the wrapper method for subset evaluation. We used the classifier subset evl to apply wrapper method for the selection of effective set of attributes from the list of attributes. Subset evaluation can be performed manually as well. However, manual comparison and creating subsets is time consuming task and also human errors and bias can have an impact on the outcomes

4.13.1 Textual Attributes Wise Comparison

The first combination we used in our comparison consists of the title, summary and comments attributes of software issue reports. F-score is used as a performance evaluation measure. Support vector machine (SVM) is used in the experimentation and performance evaluation process. Figure 4.8 shows the obtained results of textual attributes. From figure 4.8 it is very clear that title and summary attributes performed better then expert comment attribute of the software issue reports. For instance, title and summary attributes both achieved 0.74 and 0.76 F-score on both the Firefox and Eclipse datasets. On the other hand the worst performing attribute was “comments or expert comments” that yielded an F-score of 0.51 and 0.48 on Firefox and Eclipse datasets respectively.

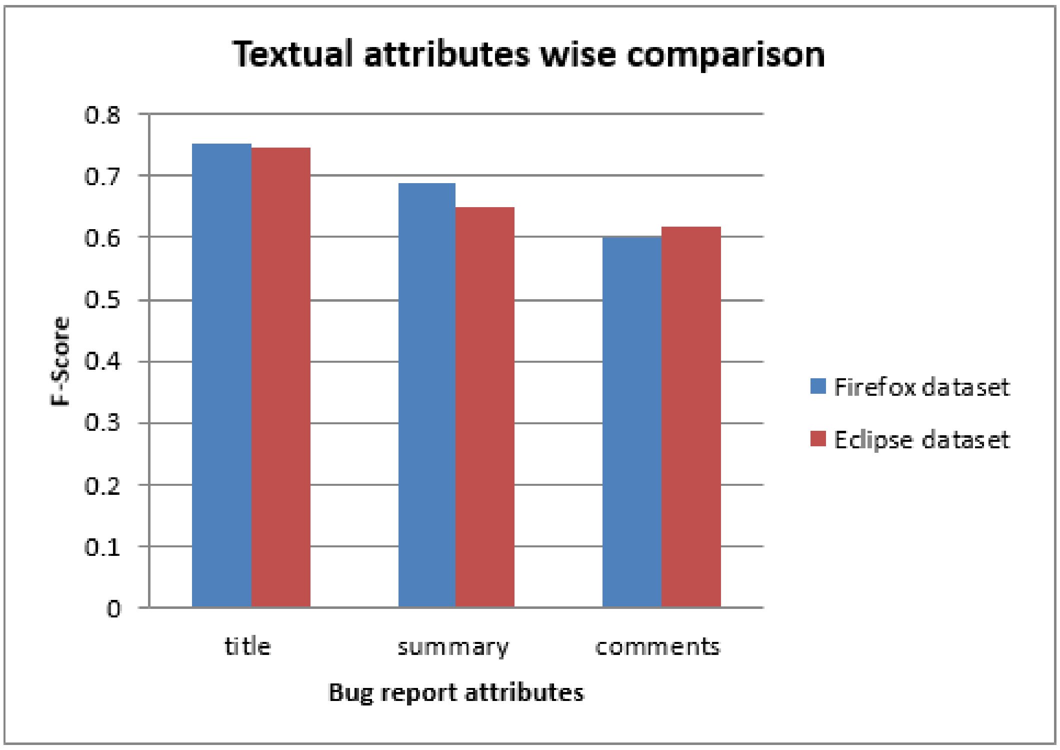


FIGURE 4.8: Results for textual attributes comparison

TABLE 4.5: classifier performance on Firefox dataset for multi classification

S.No	Subset	Complete set
1.	Title	Title
2.	Summary	Summary
3.		Comments

4.13.2 Performance of Textual Attributes using Wrapper Technique

In order to evaluate and determine the best performing textual attributes of our selected textual attributes from software bug reports. For this purpose we applied the wrapping technique that determines the best possible subset from all the features and gives its results. the wrapper technique selected the following subset from the complete. Figure 4.9 depicts the textual attributes performance after applying the wrapper subset evaluation technique. Figure 4.9 depicts the performance of the textual attributes that were selected as the best performing attributes from the set of textual attributes we are using in this work. The wrapper technique selected title and summary from the textual attributes of software

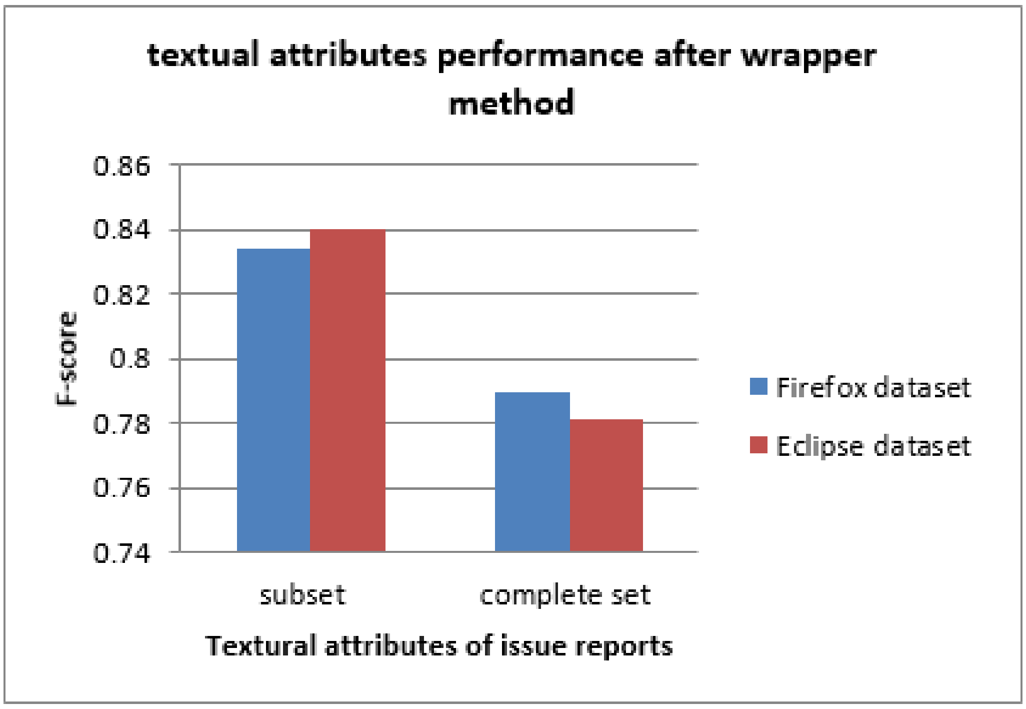


FIGURE 4.9: Results for textual attributes comparison

bug reports. We used support vector machine as performance evaluator. The subset features yielded better results as compared to the complete list of textual attributes used in experiments. Subset of attributes yielded an F-score of 0.83 and 0.84 respectively for Firefox and Eclipse dataset. On the other hand side complete set of our selected attributes also yielded better results (i.e. 0.77 and 0.075) for Firefox and Eclipse dataset. Hence, by using the wrapper technique the results were improved significantly as compared to the results when all of the selected attributes were used for classification.

4.14 Categorical Attribute's Performance Comparison

We selected all the attributes from categorical category (i.e. severity, priority, product, version and platform) of software issue reports. Most of the categorical attributes obtained lower F-score as compared to textual attributes of software issue reports. For instance, the best performing categorical attribute was severity, product and priority that yielded an F-score of 0.61, 0.58 and 0.62 respectively on Firefox and Eclipse datasets. Version and platform attributes performed poorly in terms of F-score. Moreover, figure 4.10 results also indicates that it only using categorical attributes of bug reports is not suitable for classification purpose. However, combining categorical and textual for issue reports classification, it gives better performance and classification results as compared to only categorical attributes.

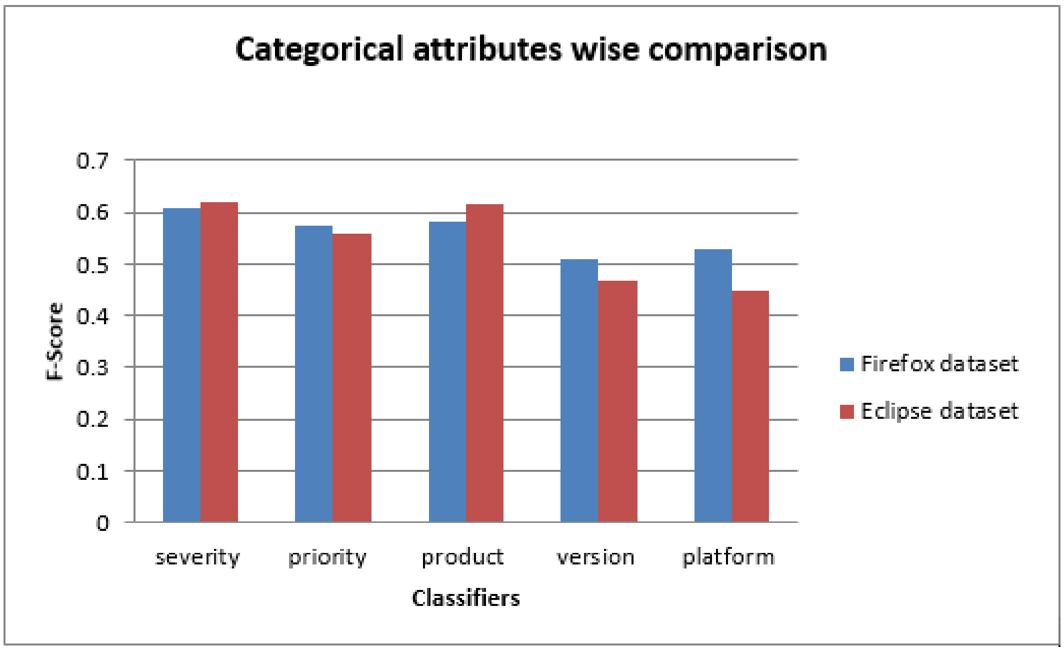


FIGURE 4.10: Results for software categorical attributes after applying wrapping method

4.14.1 Categorical Attributes Performance after Applying Wrapper Technique

We also applied the wrapping technique on the categorical attributes of software bug reports in order to determine the best performing categorical attributes from the list of all the categorical attributes we used in our research work. The wrapper technique selected the following subset from the complete. Figure 4.11 sows the obtained results terms of F-score when wrapper technique was applied to only

TABLE 4.6: classifier performance on Firefox dataset for multi classification

S.No	Subset	Complete set
1.	Severity	Severity
2.	Priority	Priority
3.	product	Product
4.		version
5.		Platform

categorical attributes. The wrapping select only severity, priority and product attribute form the list of all five categorical attributes we have used in our research work. From the obtained results it's very obvious and clear that by applying the wrapper technique for subset evaluation generated improved results as compared to the results obtained by employing all the categorical attributes. Severity and priority attributes were the best performing attributes. Subset of categorical attributes attained an F-score of 0.72 and 0.71 respectively for Firefox and Eclipse dataset as compared to complete set of attributes that generated an F-score of 0.67 on Firefox dataset, while on Eclipse dataset it was 0.65.

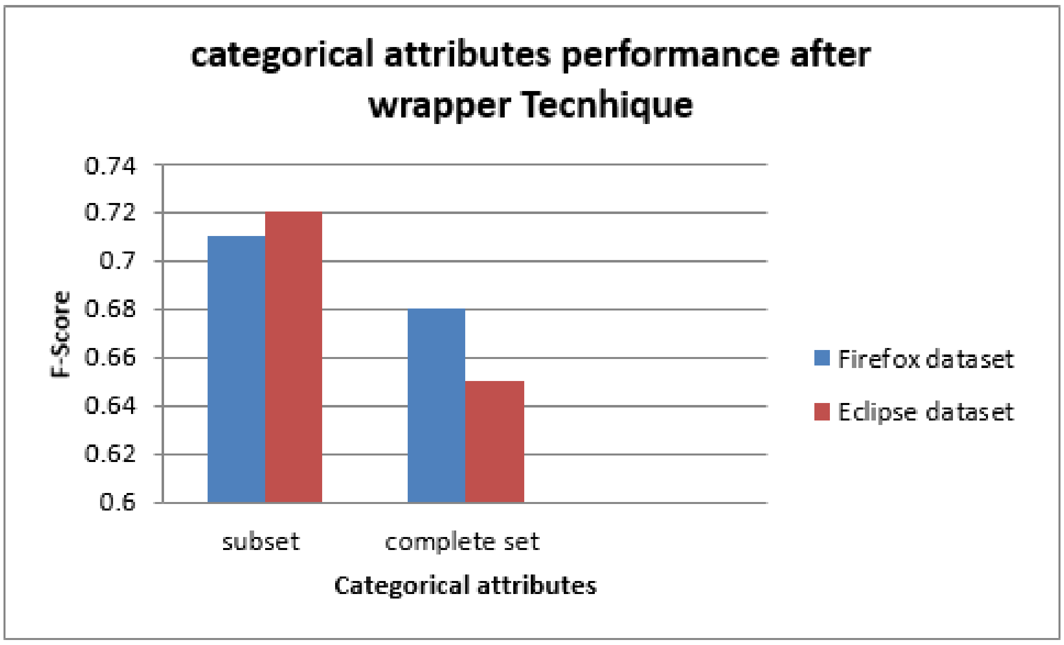


FIGURE 4.11: categorical attributes performance after applying wrapper technique

4.15 Textual and Categorical Attributes Performance Comparison

This Combination consists of all the selected software bug report attributes that we have used in the previous combinations. In combination we combined all the selected attributes from the both categorical and textual categories of software issue reports. This combination consists of different software issue report attributes such as title, summary, comments, severity, priority, product, platform and version. Figure 4.12 depicts the experimental results of the of bot textual and categorical attributes of software issue reports using support vector machine. In Figure 4.12 attributes like title, summary, severity and priority achieved F-score between 0.66 and 0.81 on Firefox and Eclipse datasets respectively. Version and platform attributes from the categorical attributes achieved moderate F-score of 0.65 and 0.68 respectively. hence, these performance results indicates that using suitable number of textual and categorical attributes can significantly improves the performance and classification of software bug reports into different classes and categories.

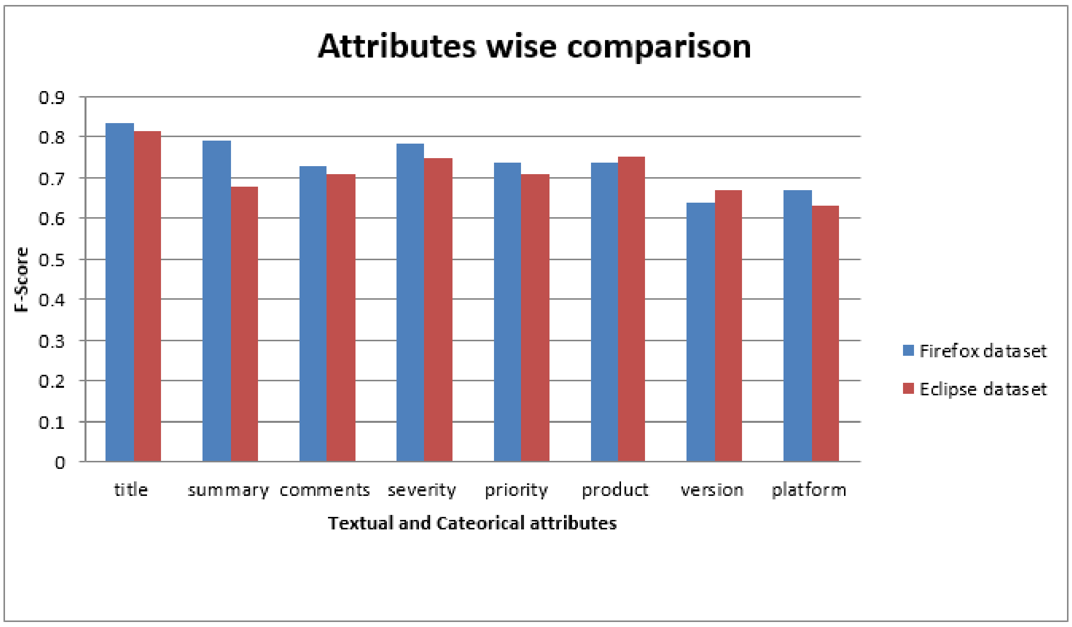


Figure 4.12: Results for textual and categorical attributes

Table 4.7: classifier performance on Firefox dataset for multi classification

S.No	Subset	Complete set
1.	Severity	Severity
2.	Priority	Priority
3.	product	Product
4.	Title	Version
5.	Summary	Platform
6.		Title
7.		Summary
8.		Platform

4.15.1 Textual and Categorical Attributes Performance after Applying Wrapper Technique

We also performed the comparison of all the selected textual and categorical attributes issues reports we used in this research work. In order to compare and determine the best performing attributes in the classification process, we applied the wrapper technique on all the bug reports attributes. Figure 4.13 depicts the results of subset attributes after applying wrapper technique. The wrapper technique selected four best performing attributes that yielded better results in terms of F-score while using support vector machine as classifier. These attributes are bug title, summary from textual category, severity and priority from categorical category. Subset of features from both the categorical and textual attributes category obtained an F-score of 0.841 and 0.847 for Firefox and Eclipse datasets respectively. While the complete set of attributes produced an F-score of 0.832 and 0.824 on both the Eclipse and Firefox datasets respectively. From these experimental results after applying wrapper technique we observed that when textual and categorical attributes are combined for issue reports classification, it generates better results as compared to using textual and categorical attributes separately.

4.16 Analysis and Discussion

In this section we present the detail analysis of results obtained by our proposed approach. We also answer the research questions that we have formulated in the

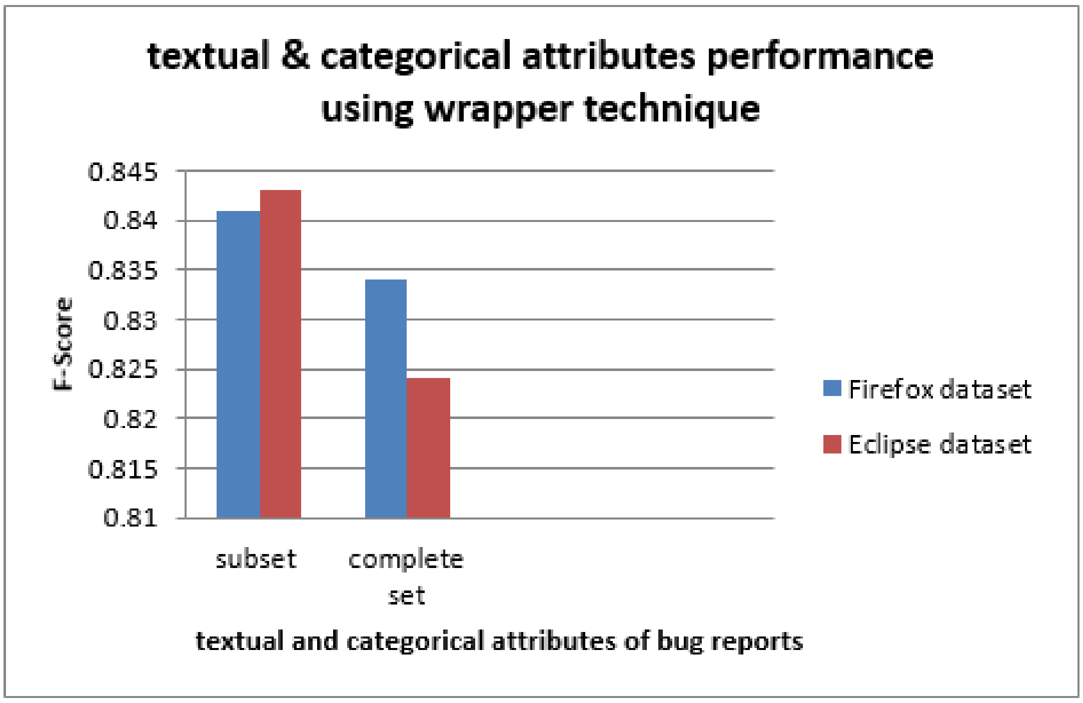


FIGURE 4.13: Textual and categorical attributes performance after applying wrapper technique.

first chapter of this thesis document. In this work we use bug reports from two software bug tracking systems datasets (i.e. Firefox and eclipse). We applied six machine learning algorithms on textual, categorical attributes of software bugs reports. most of the classifiers used in this work yielded better results on binary classification as compared two multi classifications when textual and categorical attributes of software bugs used separately for classification. However, when both the categorical and textual attributes are used combined its yields significant results. Furthermore, most of the classifiers achieved better results by combining textual and categorical attributes of software bug reports.

RQ1: what is impact of using only textual attributes and using the combination of both textual and categorical attributes on bug's classification?

To answer research question two we have analyzed the impact and behavior of both the textual and categorical attributes of the bug reports for classification. Textual features are one of the most important and rich attributes that describes the occurrence of bug in much details. Textual attributes provides verity of information about a particular bug report. However, sometimes the extraction of meaningful from the free text is a challenging task due to the varying information present in the bug's reports. We experimented with both textual and categorical attributes and analyzed their impact and significance in bug's classification process. The presence of textual and categorical attributes in bug's reports plays an important role in the classification process. We used both the textual and categorical attributes of the software bugs reports our classification approach. From the experimental results we concluded that by using both the categorical and textual attributes of Firefox and Eclipse bug's reports. Moreover, all the classifiers achieved the highest results on using categorical and textual attributes as compared to separately using these attributes. For instance, using both categorical and textual attributes for classification resulted in highest F-score of 0.88 and 0.85 respectively in binary as well as multi classification of software bugs reports.

RQ2: what is the suitable number of classes for classifying software issue reports?

The proper and suitable selection of classes for classification of software bugs is a challenging and critical task; we use varying number of classes in order to determine the accurate number of classes that can be used for classification. We demonstrated the performance of our proposed approach using binary and multi classes. The number of multi classes is kept at four, which are non-bugs, logical bugs, GUI bugs and enhancement bugs. From the experimental results we observed a slight degradation in the performance of the entire classifiers when the number of classes was increased from two to four. However, when we selected only categorical attributes of the bug's reports their results were significantly poor as compared to that of using both categorical and textual attributes. However, in case of the binary classification categorical attributes produced better results as compared to multi classification.

RQ 3: what is the impact of machine learning algorithms in the classification process?

In this work we have performed bug reports classification using six machine learning algorithms (i.e. SVM, KNN, NB, GB, DT and RF). Most of the classifiers achieved better classification results except random forest and Gaussian naïve

Bayes. SVM, KNN and decision tree were the highest performing algorithms that produced better results on both the datasets. On the other hand the worst performing algorithm was Gaussian naïve Bayes which consistently performed poorly on both Eclipse and Firefox datasets

RQ4: what are the most effective textual and categorical attributes of software issue reports that can yield better classification results?

In order to answer to research question five we have applied a wrapper technique that results in a subset of features set that performs better as compared to the complete of feature set that has been used in this research work. We applied the feature subset selection technique on both the textual and categorical attributes of bug reports. We observed from the experimental results that by applying wrapper technique the classification performance can be improved up to some extent. In our case wrapper technique selected a subset of features consisting of bug title, summary, severity and priority that resulted in better performance in terms of F-score on Firefox and Eclipse datasets.

4.17 Comparison with Existing Approaches

We also compare the performance of our approach with 4 existing techniques in the literature review by Sarkar et al [43], Kukkar and Mohana [38], Otoom et al.[43]. Our approach achieved better results in terms of accuracy and F-score against the proposed approach of Sarkar et al [43]. However, Sarkar et al [43] used a close source dataset for their evaluation. The proposed approach used total of 8 attributes, in which two attributes were from textual category wile 6 attributes were the categorical attributes of the bug reports. While, [42] used only two classes for classifying software issues into two classes which are corrective class and perfective class. Corrective class represents the actual software bugs while perfective class represents enhancement related issues. The dataset used by [42] was obtained from bugzilla bugs tracking repository. On the other side the authors in [38] classified bugs reports into two main classes (i.e. bugs and non- bugs). Moreover, our proposed approached achieved better recall than Kukkar and Mohana [38] on the

Table 4.8: Comparison with existing approaches

Approach	**Accuracy**	**Precision**	**Recall**	**F-Measure**
Otoom et al [42]	0.93	0,83	0.85	0.83
Sarkar et al [43]	0.79	0.78	0.79	0.78
kukkar [38]	Nill	0.92	0.54	0.68
Proposed Approach	0.88	0.87	0.86	0.85

same Firefox dataset. Following table shows a comparison of our proposed approach against the existing approaches in the literature. The table 4.5 shows the comparison of our proposed approach against the two mostly relevant approaches in the literature review conducted in chapter two of this thesis. For the above table it very clear that our proposed approach generated better classification results as compared to the two existing approaches. Although, the approach presented by Otoom et al [42] obtained better accuracy from our approach. However, the proposed approach used only two classes for classifying for software issue reports.. Furthermore, the dataset used in [42] also consists of Firefox bug reports, and it used title, summary and description attributes form textual category, product, platform and priority attributes were selected from the categorical attributes category. However, the number of instances in the dataset was less than the dataset we used in our experiments. On the other hand in comparison with [43] our proposed completely outperformed the existing approach terms of accuracy, precision. And recall. Moreover, Kukkar and Mohana [38] used the same dataset (i.e. Firefox) what we have used for experiment and we have achieved better recall.

Chapter 5

Conclusion and Future Work

This section will provide the conclusion of our research work and limitations for the future work.

5.1 Conclusion

In this thesis we proposed a mechanism for classifying software bug's reports into different classes and categories using the categorical and textual attributes from issue reports. due to the exponential number of bugs reported on daily basis in different open source bugs tracking systems like Bugzilla and eclipse. Due to the huge number of the bugs reported on daily basis it is infeasible to classify and triage software bug's reports manually into different classes and categories. In literature various approaches and techniques have been proposed for software issue reports classification and triaging. Most of the existing approaches use several attributes of the software bugs reports for classification. For instance, one of the most widely technique is to use the textual attributes of the bug report and apply classification algorithm on them. Some existing approaches have used both the categorical and textual attributes of the software bugs reports in order to effectively classify software bugs. However, still the existing approaches suffer due to several limitations.

In this work we proposed a software bug's issues classification technique using the categorical and textual attributes of software bugs reports. We extract feature set from the textual and attributes of software issue reports based on the common occurrence of certain keywords. We also use categorical features of software bug reports in the classification process. We use categorical attributes like product, version, priority and severity in our approach in order to classify software bug reports into different classes. Such as actual bug's class, non-bugs class, logical, GUI and enhancement related bug classes. The basic intuition behind the use of categorical and textual attributes is to evaluate their performance and effectiveness in the classification process. We classify bugs into two different combinations of classes, such as actual bugs and non-bugs logical bugs, non-bugs, and enhancement class. We also performed subset selection of features that yields better results as compared to complete set of feature we used for issue reports classification.

Experiments were performed based on three models, model M1 uses only the textual attributes of bug reports for classification, while model M2 considers categorical attributes of bug reports. Model M3 employs both textual and categorical attributes of software bug reports. We used six machine learning classification algorithms for classifying software bugs into different classes and categories, these classifiers are support vector machine (SVM), K nearest Neighbour (KNN), Naïve Bayes (NB), Gaussian naïve Bayes (GB), Decision tree (DT) and random forest (RF). Four different evaluation measures (i.e. Accuracy, precision, recall and F-score) are applied for evaluating the performance of our proposed approach. Furthermore, the performance of our proposed approach was demonstrated on widely used open source datasets obtained from Bugzilla and Eclipse bugs tracking systems. We used the Weka API in python for implementing our proposed technique. We also assessed and evaluated the performance of different categorical and textual attributes of issue reports by using several combinations of textual and categorical attributes in the experimentation.

Most of the machine learning algorithms achieved significant results on both Firefox and eclipse datasets. For instance, SVM, KNN and DT achieved the highest results in between 0.88, 0.86 and 0.84 in terms of F-score when both the categorical and textual attributes of the software bugs reports were used in the experiments. Apart from, other classifiers also yielded better results except Gaussian naïve Bayes

which consistently achieved poor results on both Firefox and Eclipse dataset. in case of categorical bug attributes did not yielded better results in both binary and multiclass classification when they were used separately. However, when we combined the categorical and textual attributes they generated higher results.
Apart from this we used varying number of classes in order to investigate the accurate number of classes that can be used for software bugs reports classification. from the experimental analysis we observed that both binary and multiclass classification achieved better results when the categorical and textual attributes were used combined. Hence, it means that by using both appropriate textual and categorical attributes can lead to better classification results when the number of the classes kept at minimum of four.

5.2 Future Work:

Software bugs reports classification and triaging is important activity in any software development and engineering environment. In this work we performed bugs reports classification on two bug's datasets (i. e. Firefox and Eclipse dataset). We applied six machine learning algorithms on Firefox and eclipse dataset in order to classify software bugs into different categories. In future we aim to classify software bugs reports using Deep learning algorithms which are considered one of the most powerful classifiers. Although a few approaches have applied deep learning for bugs triaging, however, they have only considered the textual attributes of software bugs reports, we aim to apply deep learning algorithms on both textual and categorical attributes of software issue reports. We also aim to consider other problems related to software bug reports triaging and classification domain, some of the problems and issues that can be considered for further research work cold start problem, bugs fix time and analysis, developer weight analysis and software bug prioritization. Another important direction in the classification and triaging process would be to consider all the fields from software bug reports and to perform feature selection on tem, and ten to classify software bugs based on the feature selection in order to identify the best and suitable features or attributes that helps in bug reports classification process.

Bibliography

[1] S. Hangal and M. Lam, “Tracking down software bugs using automatic anomaly detection.” *In Proceedings of the 24th International Conference on Software Engineering.*, 2002.

[2] A. Hassan, “The road ahead for mining software repositories.” *Frontiers of Software Maintenance (pp.*, vol. 22, no. 3, pp. 48–57, 2008.

[3] R. Charette, “Why software fails [software failure]. ieee,” *IEEE spectrum*, vol. 42, no. 5, pp. 42–49, 2011.

[4] W. J. E. Kim, S. and Y. Zhang, “Classifying software changes: Clean or buggy?. ieee,” *IEEE Transactions on Software Engineering, 34(2),*, vol. 54, no. 3, pp. 181–196, 2008.

[5] M. Lyu, “ handbook of software reliability engineering (vol.” *IEEE computer society press.*, vol. 22, no. 43, pp. 50–65, 1996.

[6] H. L. Anvik, J. and G. Murphy, “Who should fix this bug?.” *In Proceedings of the 28th international conference on Software engineering*, vol. 22, no. 3, pp. 361–370, 2006.

[7] “https://bugzilla.mozilla.org/describecomponents.cgi.”

[8] “https://www.mantishub.com/.”

[9] “https://jbossbuild.jboss.org/issue-tracking.”

[10] “https://bugs.chromium.org/hosting/.”

[11] D. B. K. S. S. E. Leif Jonsson, Markus Borg and P. Runeson., "Automated bug assignment: Ensemble-based machine learning in largescale industrial contexts." *Empirical Software Engineering,*, vol. 21, no. 4, pp. 1533–1578,, 2016.

[12] Z. M. K. Ramin Shokripour, John Anvik and S. Zamani., "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation." *In Working Conference on Mining Software Repositories,*, vol. 25, no. 9, pp. 2–11, 2013.

[13] J. A.-K. Ahmed Tamrawi, Tung Thanh Nguyen and T. N. Nguyen, "Fuzzy set- based automatic bug triaging: Nier track." *In International Conference on Software Engineering*, vol. 43, no. 2, pp. 884–887,, 2011.

[14] N. Nagwani and S. Verma, "A comparative study of bug classification algorithms," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 1, pp. 111–138, 2019.

[15] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement? a text-based approach to classify change requests," 2008.

[16] J. H. S. C. K. D. Lo, H. Cheng and C., "Classification of software behaviors for failure detection: A discriminative pattern mining approach," *in Proceedings of 136 N. K. Nagwani, S. Verma.*

[17] D. G. G. Bougie and M., "A. storey, a comparative exploration of freebsd bug lifetimes," *in 7th IEEE International Working Conference on Mining Software Repositories,*, vol. 65, no. 8, p. 106–109., 2010.

[18] Z. T. P. R. Schröter, A. and A. Zeller, "If your bug database could talk." *In Proceedings of the 5th international symposium on empirical software engineering (Vol.2008 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, no. 1, pp. 18–20, 2006.

[19] G. Vijayaraghavan and C. Kaner, “Bug taxonomies: Use them to generate better tests,,” *in Software Testing Analysis and Review Conference,*, vol. 51, no. 3, pp. 1–40, 2003.

[20] “A con¯gurable project tracking tool, available at http://www.atlassian.com/software/jira/(accessed jan, 2020).”

[21] A.-j. S. Otoom, A.F. and M. Hammad, “Automated classification of software bug reports.” *In Proceedings of the 9th International Conference on Information Communication and Management*, vol. 43, no. 3, pp. 17–21, 2019.

[22] R.-P. Sarkar, A. and B. Bartalos, “Improving bug triaging with high confidence predictions at ericsson.” *In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, vol. 13, no. 1, pp. 81–91, 2019.

[23] A. Kukkar and R. Mohana, “A supervised bug report classification with incorporate and textual field knowledge. procedia,” *Procedia computer science*, vol. 31, no. 4, pp. 352–361., 2018.

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

[25] J. Anvik, “Automating bug report assignment.” *In Proceedings of the 28th international conference on Software engineering*, vol. 33, no. 5, pp. 937–940, 2006.

[26] T. Menzies and A. Marcus, “Automated severity assessment of software defect reports.” *In 2008 IEEE International Conference on Software Maintenance*, vol. 65, no. 4, pp. 346–355, 2008.

[27] H.-L. Anvik, J. and G. Murphy, “Who should fix this bug?.” *In Proceedings of the 28th international conference on Software engineering*, vol. 49, no. 5, pp. 361–370, 2009.

[28] G.-D. G. M. Hindle, A. and R. Holt, “Automatic classication of large changes into maintenance categories.” *In 2009 IEEE 17th International Conference on Program Comprehension* , vol. 51, no. 3, pp. 30–39, 2009.

[29] D.-S. G. E. Lamkanfi, A. and B. Goethals, "Predicting the severity of a reported bug." vol. 34, no. 3, pp. 1–10, 2010.

[30] K. Chaturvedi and V. Singh, "Determining bug severity using machine learning techniques." vol. 44, no. 3, pp. 1–6, 2012.

[31] M. Javed and H. Mohsin, "An automated approach for software bug classification." *In 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, vol. 3, no. 7, pp. 414–419)., 2012.

[32] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage. journal of computer science and technology," vol. 27, no. 2, pp. 397–412., 2012.

[33] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?." *In 2013 12th International Conference on Machine Learning and Applications*, vol. 29, no. 4, 2013.

[34] Y.-G. L. B. Zhang, T. and A. Chan, "Predicting severity of bug report by mining bug repository with concept profile." *In Proceedings of the 30th Annual ACM Symposium on Applied Computing (pp.*, vol. 35, no. 3, pp. 1553–1558, 2015.

[35] A.-N. Goyal, N. and M. Dutta, "A novel way of assigning software bug priority using supervised classification on clustered bugs data." *In Advances in Intelligent Informatics*, vol. 76, no. 5, pp. 493–501, 2015.

[36] S.-G. Gujral, S. and S. Sharma, "Classifying bug severity using dictionary based approach." *In 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, vol. 12, no. 12, pp. 599–602, 2015.

[37] M. Pushpalatha and M. Mrunalini, "Predicting the severity of bug reports using classification algorithms." *In 2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*, vol. 22, no. 1, pp. 1–4, 2016.

[38] C.-J. Y. G. L. B. Zhang, T. and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs. journal," *bugs. Journal of Systems and Software, 117*, vol. 23, no. 5, pp. 166–184., 2016.

[39] D.-A. Y. G. L. B. Jin, K. and J. Lee, "Improving predictions about bug severity by utilizing bugs classified as normal. contemp," *Contemp Eng Sci*, vol. 9, no. 19, pp. 933–942., 2016.

[40] J.-H. R. Z. Y. J. Xuan, J. and Z. Luo, "Automatic bug triage using semi-supervised text classification. arxiv," vol. 3, no. 9, 2017.

[41] S.-A. Mani, S. and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging." *In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data* , vol. 65, no. 4, pp. 171–179, 2018.

[42] M. Pushpalatha and M. Mrunalini, "Predicting the severity of closed source bug reports using ensemble methods." *In Smart Intelligent Computing and Applications*, vol. 47, no. 12, pp. 589–597, 2019.

[43] U.-Q. Y. X. Z. C. Ramay, W.Y. and I. Illahi, "Deep neural network- based severity prediction of bug reports. ieee," *IEEE Access,* , vol. 31, no. 4, pp. 46 846–46 857., 2019.

[44] "https://bugzilla.mozilla.org/describecomponents.cgi."

[45] "https://bugs.chromium.org/hosting/."

[46] Z.-A. F. G. Z. C. Rößler, J. and G. Candea, "Reconstructing core dumps." *In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, vol. 32, no. 2, pp. 114–123)., 2013.

[47] M. Porter, "Snowball: A language for stemming algorithms." 2001.

[48] K.-S. Pan, K. and E. Whitehead, "Toward an understanding of bug fix patterns." *Empirical Software Engineering*, vol. 14, no. 4, pp. 286–315., 2009.

[49] V. Vapnik, "the nature of statistical learning theory," 1995.

[50] Y. EL-Manzalawy, "Integrating libsvm into weka environment, software," vol. 53, no. 3, 2005.

[51] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classi¯ers,," *in Proceedings of Eleventh Conference on Uncertainty in Arti¯cial Intelligence*, vol. 27, no. 3, pp. 10–19, 1995.

[52] A. Mccallum and K. Nigam, "A comparison of event models for naive bayes text classi¯- cation,," *in Proceedings of AAAI-98 Workshop on Learning for Text Categorization,*, vol. 31, no. 6, pp. 21–29, 1998.

[53] M. Friedl and C. Brodley, "Decision tree classification of land cover from remotely sensed data. remote," 1997.

[54] R. Raizada and Y. Lee, "Smoothness without smoothing: why gaussian naive bayes is not naive for multi-subject searchlight studies," *PloS one*, vol. 8, no. 7, pp. 31–43, 2013.

[55] A. Liaw and M. Wiener, "Image segmentation by probabilistic bottom-up aggregation and cue integration," *R news* , vol. 2, no. 3, pp. 18–22, 2002.

[56] D. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation." vol. 43, no. 3, pp. 551–559, 2011.