Automatic Labeling of Security Bug Reports

Author

Mohammad Umer Sadiq

FALL 2017 - MS-17 (CSE) 00000205473

Supervisor

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

NOV 2020

Automatic Labeling of Security Bug Reports

Author

Mohammad Umer Sadiq

FALL 2017 - MS-17 (CSE) 00000205473

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature: -____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

NOV 2020

DECLARATION

I authorize that this research work titled *"Automatic Labeling of Security Bug Reports"* is my own work under the supervision of Dr. Wasi Haider Butt. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Mohammad Umer Sadiq

FALL 2017 - MS-17 (CSE) 00000205473

PLAGIARISM CERTIFICATE (TURNITIN REPORT)

The plagiarism of this thesis has been checked. Turnitin report approved by the supervisor is attached.

Signature of Student

Mohammad Umer Sadiq

FALL 2017 - MS-17 (CSE) 00000205473

Signature of Supervisor

Dr. Wasi Haider Butt

LANGUAGE CORRECTNESS CERTIFICATE

This thesis has been read by an English expert and is free of semantic, typing, grammatical, spelling and syntax mistakes. Thesis is also conferring to the format provided by the university.

Signature of Student

Mohammad Umer Sadiq

FALL 2017 - MS-17 (CSE) 00000205473

Signature of Supervisor

Dr. Wasi Haider Butt

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I am extremely thankful to **ALLAH** Almighty for his bountiful blessings throughout this work. Indeed, this would not have been possible without his substantial guidance through every step, and for putting me across people who could drive me though this work in a superlative manner. Indeed, none be worthy of praise but the Almighty. In addition, my admirations be upon Prophet **Hazrat Muhammad (PBUH)** and his Holy Household for being source of guidance for people.

I like to show my special thanks to my supervisor **Dr. Wasi Haider Butt** for his generous help during my thesis, and for being available even for the minor issues. My thanks for a cautious evaluation of the thesis, and direction on how to correct it in the best way possible.

I am profusely thankful to **Dr. Arslan Shaukat** and **Dr. Urooj Fatima** for an excellent guidance throughout this journey and for being part of my evaluation committee.

It is indeed a privilege to thank my Mother, my father **Mohammad Sadiq** and my elder brother **Ahmed Hassan Sadiq** for their constant encouragement throughout my degree and research period. The sense of belief that they instilled in me has helped me sail through this journey. I would like to thank my family & friends, who have rendered valuable assistance to my study.

Finally, I would like to show my thanks to all the individuals who have rendered valuable assistance in this period.

Dedicated to my extraordinary parents, excellent siblings and my best friend whose amazing support and assistance led me to this astonishing accomplishment. I am truly indebted to you all.

ABSTRACT

There are many stages in software development life cycle and each stage is associated with different kind of artifacts. Bug reports used for many software development activities like severity and priority assignment and triaging of bugs. It’s difficult for the developers to resolve all bug reports due to the limited resources. Developers usually need to prioritize bug reports to resolved bugs of various software projects hurriedly. There are various types of bug reports such as security, performance, regression, usability and crash. Among these, security bug reports are highly crucial. These types of bug reports can express security debt that could abused by the hackers if they disclosed before they resolved. A security bug can becomes the reason of an unauthorized access to the software applications. These bugs are great threat to the privacy and security of users. Therefore, these bugs are needed to be resolved as early as possible. A bug reports contains many different fields, showing information about bugs. Certain fields are optional, and some are mandatory. JIRA consists a column named “type”, which may be a bug, an advanced feature, an improvement or a support request. In BUGZILLA, key-word field is tagged with category of bug such as ‘perf’ for performance bug. Label or Type field describe the type of a bug report. Label can give an understanding about the bug reports and also be used for the priority of bug reports. Previous Studies show that many bug reports are not labeled, if some are label, they may be not accurate. In this research, we purposed an approach for automatic labeling of security bug reports. At first, we conducted the systematic literature review (SLR), this SLR consists distribution of papers according to approaches used by authors. Identify different NLP techniques, libraries and technologies used to develop tools. Then we identified thirteen (13) tools that are purposed or developed by different researchers and a comparison is performed. After performing SLR, we purposed a novel approach for the automatic labelling of security bug reports by using natural language processing’s (NLP) techniques and machine learning (ML) algorithms. Our approach named ALSBR is implemented in Python using Natural language Toolkit (NLTK), Sklearn and Imblearn libraries. In our purposed methodology, first of all preprocessing of bug reports is performed. After the preprocessing, features are selected by TF-IDF values. Top hundred terms according to TF-IDF values are selected as features. After feature selections, a random under sampling technique is applied to balance the majority and minority classes. Three machine learning algorithms named Logistic Regression, Decision Tree and Naïve Bayes is utilized as classification model. A voting strategy is also applied to get the more accuracy. For the validation of our approach, 10-Fold cross-validation is applied. We used bug reports of five projects for the evaluation of our approach. Among these projects, four are from Ohira and one is a subset of bug reports that is selected from Chromium project. At the end, we compared purposed approach with state-of-the-art approach named FARSEC model and achieved improved results in terms of Precision, Probability of detection (Recall), Probability of false alarm, F-measure and G-measure.

Keywords: *Security Bug Reports, Machine Learning, Natural Language Processing*

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

There are many stages in software development life cycle and each stage is associated with different kind of artifacts. One of those artifacts is bug report. These are essential for development of any kind of software. We can define a software bug as [1].

“A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result or to behave in unintended ways.”

These bugs are documented in a report called bug reports. These reports permit users to notify-the developers about problems they faced when using any software. These reports consist of details of a failure and they usually give information about the location of a failure in the code. A bug reports contains many different fields, showing information about bugs. Certain fields are optional, and some are mandatory. Label or Type field describe the type of a bug report. Therefore, these reports are utilized for many software development activities like severity, priority assignment and triaging of bugs [1,2].

There are various categories of bug reports such as security, performance, usability, crash and regression [31]. Among these, security bug reports are highly crucial [9]. These reports can express security debt that could abused by the hackers if they disclosed ahead they resolved. It can cause an unauthorized access to software [8]. These bugs are great threat to the privacy and security of users [13]. Therefore, these bugs are needed to be fixed as early as possible. It can be defined [9] as

“A security bug is a security vulnerability that allows a user to have inappropriate access to the system and thus cause harm or damage to the software or to persons using the software.”

Software projects used bug tracking system to document and follow the progress of every bug which is identified by developers, testers and user of software system [8]. Bug tracking system (BTS) is a software tool which saves the record of reported bug reports during software developments projects. Many BTS grant permission to end users to report bug directly and few are only used within the organization during software developments. These tracking systems are combined with the other tools of project management. A BTS is a significant element of many software development companies and use of these tracking systems are considered “hallmarks of a good software team". Bugzilla, Jira, Trac and GitHub are mostly used bug tracking systems.

Machine learning (ML) is a “study of statistical models and algorithms which computer used to operate specific tasks without using explicit instructions”. It is an Artificial Intelligence (AI)’s application that allows computers to learn accordingly and improved from the experience without any programming. ML algorithms are usually categorized into two type. One is supervised machine learning and other one is unsupervised machine learning. In first type, class labels of some data are known, this data is called training data. New data is classified depend on the training data; this new data is called testing data. There are one or many inputs and liked output for each training example. In unsupervised machine learning algorithms, class labels of training data are unknown.[10].

Natural language processing (NLP) is the field of artificial intelligence which is basically concerned with automatic analysis of plain natural language [62]. There are several techniques in the natural-language-processing e.g. tokenization, POS tagging, sentence splitting [62].

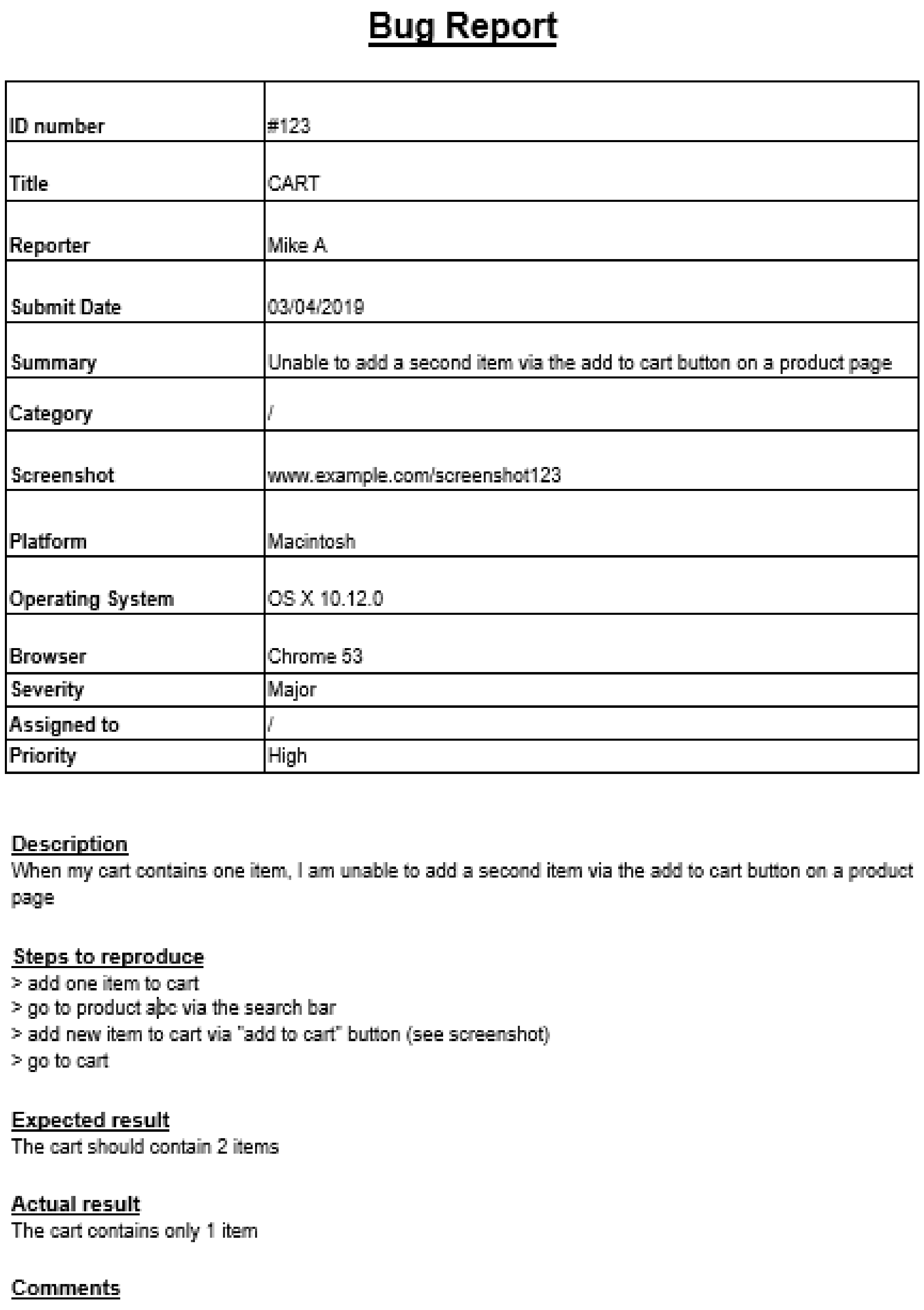
Tokenization is an NLP Technique which is used to split the natural language paragraph into sentences, then these sentences are broken into tokens [61]. POS Tagging technique is used to designate part of speech to each word in a sentence like noun, verb, adverb etc. [61]. Parsing is used to build a syntax tree of the sentences, stemming returns the words to its base form and TF-IDF is used to identify the value of a word in a paragraph [62].

1.1 Background, Scope and Motivation

Bug reports used for many software development activities like severity and priority assignment and triaging of bugs[1]. The reliability and quality of these tasks highly depend on information available in bug reports. Previous study[2] show that bug reports usually contain incorrect and incomplete information. Therefore, developers spend more time and effort on the inspection of errors [2]. Over 45% of development time of IT companies spend in repairing various bugs during software development [3][4]. As the complexity and scale of projects increase, large no of bug-reports obtained everyday by BTS. They allow the reporters to report any kind of bugs they faced while using various software products. A bug report is shown in figure 1.1.

It’s difficult for the developers to resolve all bug reports due to the limited resources. Developers usually need to prioritize bug reports to resolved bugs of various software projects hurriedly. A bug reports contains many different fields, showing information about bugs. Certain fields are optional, and some are mandatory. Label or Type field describe the type of a bug report. JIRA (a bug tracking system) consists a column named “type”, which may be a bug, an advanced feature, a support request or an improvement. In BUGZILLA, key-word field is tagged with category of bug such as ‘perf’ for performance bug. Label can give an understanding about the bug reports and also be used for the priority of bug reports. Previous Studies show that many bug reports are not labeled, if some are label, they may be not accurate [5].

Among these, security bug reports are of greater importance to the developers and the users [9]. A BTS contains many bug reports, only a few are security related bug reports. A BTS does not give any procedure to separate these from lot of reports. Our research aimed to create a model for the automatic labeling of security bug reports. For the labeling of these reports, we used natural language processing (NLP) facility with ML techniques.



Bug Report

ID number	#123
Title	CART
Reporter	Mike A
Submit Date	03/04/2019
Summary	Unable to add a second item via the add to cart button on a product page
Category	/
Screenshot	www.example.com/screenshot123
Platform	Macintosh
Operating System	OS X 10.12.0
Browser	Chrome 53
Severity	Major
Assigned to	/
Priority	High

Description
When my cart contains one item, I am unable to add a second item via the add to cart button on a product page

Steps to reproduce
> add one item to cart
> go to product abc via the search bar
> add new item to cart via "add to cart" button (see screenshot)
> go to cart

Expected result
The cart should contain 2 items

Actual result
The cart contains only 1 item

Comments

Figure 1.1: Bug Report

1.2 Aims & Objectives

Main aims and objectives of this research are shown below:

- We have performed a complete systematic literature review (SLR) of the current research relevant to automatic labeling or classifications of bug reports
- We have identified and analyzed different techniques used for automatic labeling/classifications of bug reports
- We have identified and compared the different tools built for automatic labeling or classifications of bug reports
- We have formalized a practical approach which used different ML algorithms and techniques offered by NLP
- We have implemented defined methodology to a data set and compared the results with previous systems to validate the improvements
- We have used Python for the implementation of our approach

1.3 Structure of Thesis

Chapter 1: Consists introduction, background, scope, motivation, aim and objectives. It also includes thesis's structure. Chapter 2: Comprises detailed SLR that contains different research questions, review protocols, classifications of studies which are selected w.r.t different techniques used, identification and comparison of different tools in the domain of labeling or classifications of bug reports and the answers to the research questions.

Chapter 3: Includes the purposed methodology for the automatic labeling of security bug reports. It also includes detailed implementation of our approach. This chapter also contains information about the different algorithms used. Chapter 4: Includes the results and evaluation of our purposed methodology using standard data sets. This chapter also includes the comparison of our approach to previously purposed approach.

Chapter 5: Consists a discussion on whole work done with the drawbacks of our research. Chapter 6: Includes the conclusion of research and suggests a work for the future. The thesis's outline is shown in figure 1.1.

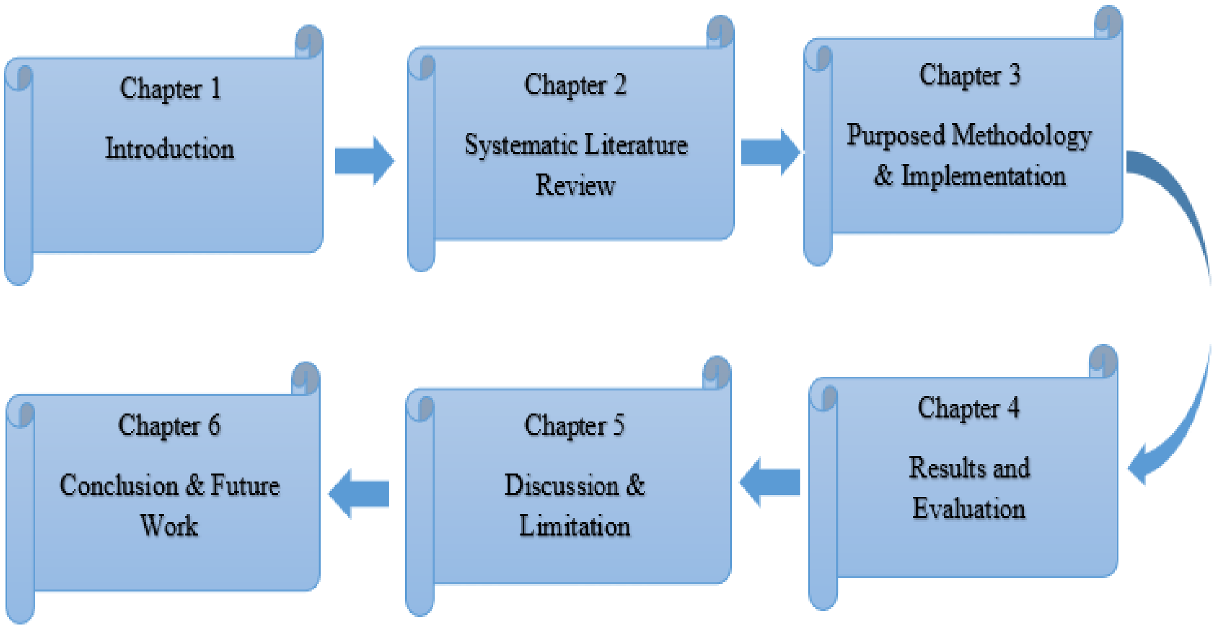


Figure 1.1: Thesis Outline

Chapter 2

Systematic Literature Review

CHAPTER 2: SYSTEMATIC LITERATURE REVIEW

This chapter contains the systematic literature review performed for our research. **Section 2.1** consists introduction to the SLR. Research methodology of literature review is explained in **Section 2.2**. The results and analysis are interpreted in **Section 2. 3**. Answers to the research questions are explained in the **Section 2.4**. Conclusion of SLR is discussed in **Section 2.5**.

2.1. Introduction

The bug reports used for many software development activities like severity, priority assignment and triaging of bugs [1]. The reliability and quality of these tasks highly depend on information available in bug reports. Previous study [2] show that bug reports usually contain incorrect and incomplete information. Therefore, developers spend more time and effort on the inspection of errors. Over 45% of development time of IT companies spend in repairing various bugs during software development [3][4]. As the complexity and scale of projects increase, large no of bug reports obtained everyday by bug-tracking-systems. Bug-tracking systems allow the reporters to report any kind of bugs they faced while using various software products.

It's difficult for the developers to resolve all bug reports due to the limited resources. Developers usually need to prioritize bug reports to resolved bugs of various software projects hurriedly. A bug reports contains many different fields, showing information about bugs. Certain fields are optional, and some are mandatory. Label or Type field describe the type of a report. JIRA (a BTS) consists a column named "type", which may be a bug, an advanced feature, a support request or an improvement. In BUGZILLA, key-word field is tagged with category of bug such as 'perf' for performance bug. Label is an optional field in google chrome bug repository.

Label can give an understanding about the bug reports and also be used for the priority of bug reports. Previous Studies show that many bug reports are not labeled, if some are label, they may be not accurate [5]. To resolve the issue of labeling of bug reports, many tools and techniques has been developed by different researchers. But a few works is performed for systematic analysis of different tools and techniques developed for the labelling of bug reports.

Hence, there is a strong need to perform a SLR of state of the art literatures related to labeling/classification of bugs. So, in this paper we performed a SLR from 2012 to 2020 and broadly examine the latest techniques and tools related to domain. We defined five (5) research questions (RQ) for SLR.

RQ1: What are the primary approaches used for the labeling/classifications of bug reports during 2012-2020?

RQ2: What are the tools purposed/developed for the labeling/classifications of bug reports during 2012-2020 and what is the difference between these tools?

RQ3: What are the primary NLP techniques used for the labeling/classifications of bug reports during 2012-2020?

RQ4: What are the libraries/technologies used for the labeling/classifications of bug reports during 2012-2020?

RQ5: How did these modern tools and techniques helped in fixing of bugs during 2012-2020?

We conducted a systematic review on 54 research studies which include both journals and conference papers, published between 2012-2020. The overview of SLR is shown in figure 2.1. The main contribution of this study is following

- At first, this study selects 54 research papers published between 2012-2020 related to labeling/classification of bug reports.
- Secondly, this study analyzes the five categories of approaches, NLP techniques, libraries/technologies and purposed/developed tools.
- At end, this study shows that how it will help the developers in bug fixing process

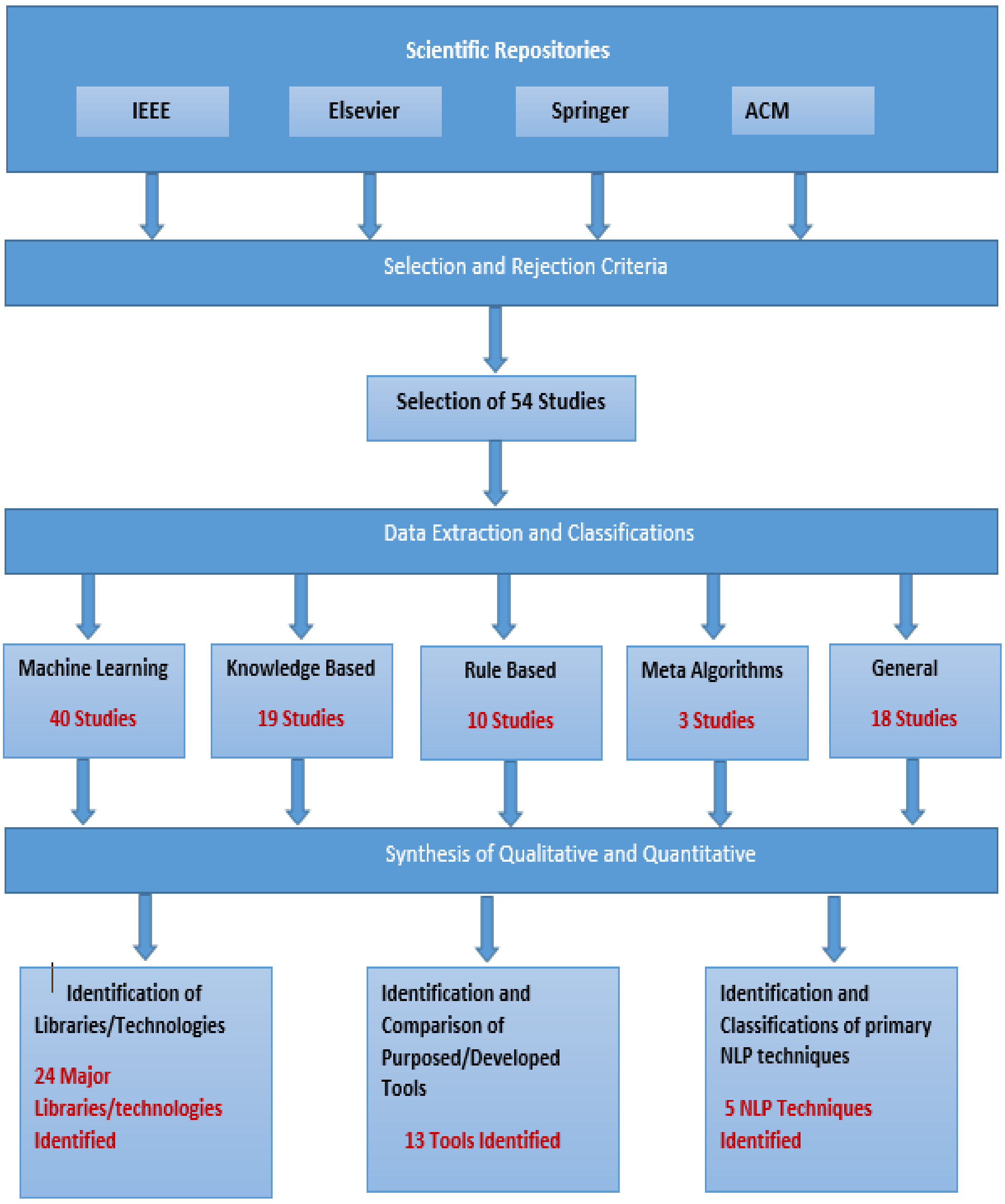

Scientific Repositories
IEEE
Elsevier
Springer
ACM
Selection and Rejection Criteria
Selection of 54 Studies
Data Extraction and Classifications
Machine Learning
40 Studies
Knowledge Based
19 Studies
Rule Based
10 Studies
Meta Algorithms
3 Studies
General
18 Studies
Synthesis of Qualitative and Quantitative
Identification of Libraries/Technologies
24 Major Libraries/technologies Identified
Identification and Comparison of Purposed/Developed Tools
13 Tools Identified
Identification and Classifications of primary NLP techniques
5 NLP Techniques Identified

Figure 2.1: Overview of SLR

2.2 Review Methodology

We have followed the Kitchenham[6]'s pattern for SLR in our study to get more precise and accurate answers to our questions. Kitchenham's methodology to conduct systematic literature (SLR) consists following five steps.

1) Defining Categories
2) Defining Selection and the Rejection Criteria
3) Searching Process
4) Quality Assessments
5) Data Extraction and Synthesis

2.2.1. Categories Definition

For the organization of selected researches, we have defined five categories. This will improve the efficiency of answers of the research questions. The explanation for each category is given below.

2.2.1.1 Machine Learning Algorithms

Machine learning (ML) is a "study of statistical models and algorithms which computer used to operate specific tasks without using explicit instructions". It is an Artificial Intelligence (AI)'s application that allows computers to learn accordingly and improved from the experience without any programming. ML algorithms are usually categorized into two type explained below.

2.2.1.1.1 Supervised Machine Learning Algorithms

In this type class labels of some data are known; this data is called training data. The training data is used to classify new data, this new data is called testing data. There are one or many inputs and desired output for each training example.

2.2.1.1.2 Unsupervised Machine Learning Algorithms

In this type class labels of training data are unknown. These approaches get a set of data which consists inputs only and find the structure of data such as clustering or grouping of the data points. Therefore, these approaches learned from the data which is not labeled, categorized or classified.

2.2.1.2 Knowledge Based Algorithms

Some researchers used knowledge-based approaches to label the bug reports. Knowledge based systems that is used to label the bug reports, consists a large amount of data as knowledge based. These knowledge bases are like dictionary that contain information, facts and meaning of various words that used in real word. This knowledge and information can have applied for the labelling of bug reports.

2.2.1.3 Rule Based Algorithms

Many researchers have used rule-based approaches for the labelling of bug reports. In this approach, rules are defined to find the required content from natural language. Researchers have defined many rules for the labelling of bug reports. In this approach, NLP prcesses such as stop words removal, stemming and POS Tagging are Enforced on the sentences then outcomes are matched with the pre-defined rules to categorize the bug reports.

2.2.1.4 Meta Algorithms

In meta algorithms, an algorithm is applied on many other algorithms to get the highest accuracy. In this, many other algorithms are combined to get more accurate results like in machine learning, Naïve Bayesian, KNN and SVM's results are combined to get more accuracy. Some of the researchers used these techniques for the labelling of bug reports.

2.2.1.3 General

In general category, those techniques are included which are not belong to any specific category. Some of the researchers used techniques which can't belong to any categories mentioned above like latent-semantic-indexing (LSI) is an indexing-technique which can't belongs to any categories.

2.2.2 Selection and Rejection Criteria

We used a well-defined selection and rejection criteria for conformation of the correctness of our research questions. We have used following five parameters to get accurate and precise answers to our research questions.

2.2.2.1 Subject Relevant

We have selected only those research papers which really answer our research questions. All these research papers are most relevant to our research questions. The research papers which are not relevant are excluded.

2.2.2.2 2012-2020

We have selected only those research papers which published from 2012 to 2020. We have rejected all those research papers which are published before 2012.

2.2.2.3 Publisher

We have selected only those research papers that are published from well-known scientific databases named IEEE, Elsevier, Springer and ACM.

2.2.2.4 Result Oriented

We have focused on a point that each research paper should be able to generate better and precise result. We have ensured that laws, experimentation and concrete facts that used in selected papers must be strong to produce a good Systematic Literature Review (SLR).

2.2.2.5 Repetition

We have ensured that selected research papers should have different context and content to produce unbiased results. Therefore, reject those researches that are undistinguishable in a specific context and select only one of them.

Table 2.1: Search Terms with Results

Sr. #	Search terms	Operators	Number of Search Results			
			IEEE	Elsevier	Springer	ACM
1	Bug Reports	N/A	881	12,089	20,981	118,158
2	Bug Reports Labeling	AND	43	3,478	6,260	38,356
		OR	25,149	753,889	23,355	140,702
3	Automated labeling Bug Reports	AND	12	1,186	2,306	12,645
		OR	61,812	936,328	551,625	158,439
4	Security Bug reports	AND	127	2,969	6,925	30,125
		OR	118,005	245,848	259,929	139,580
5	Label Security Bug reports	AND	7	990	2,121	9,341
		OR	141,024	945,363	634,662	158,260
6	Bug reports labeling Tools	AND	11	2,377	4,685	23,839
		OR	364	1,743,908	27,494	182,282
7	Bug reports labeling Natural language processing	AND	7	851	2,086	37,243
		OR	39,566	822,862	12,594	218,304

2.2.3 Searching Process

Selection and rejection criteria for the research papers is already discussed in above section. We used a state-of-the-art search process for the selection of quality researches according to the mentioned selection and rejection's standard. We have utilized four databases (IEEE, ACM, Elsevier, Springer) to select the Journals of high impact factor and conference papers to generate a quality systematic literature review (SLR). We have utilized many search terms and keywords like "Bug Reports", "Labeling bug reports", "Security bug reports" etc. for the search process. The search terms with the number of results against each term and along databases name are shown in table.

We have utilized advanced search options for each database to carry out better search process. We utilized time span filter to select research papers which are between "2012-2020". We also utilized two Boolean operators such as AND & OR to get better results. 'AND' operator is used to get more exact and precise results while ‘OR’ operator is used to large amount of results, which can help us for SLR. After the inspection of primary results, we have selected only those researches which are most relevant to our topic. We got 54 research papers by following some steps shown in figure 2.2.

- We selected 5,422 research studies from four well known scientific databases by putting different keywords.
- Then 2,186 research papers are selected and 3236 are rejected on basis of their titles.
- After this, we have selected 726 research papers rejected the 1460 studies after reading the abstract of these studies.
- After general study of remaining 726 studies, we have removed another 300 studies and after detailed analysis finally selected 54 research studies. These 54 research studies fully satisfied the previously mentioned inclusion and exclusion criteria.

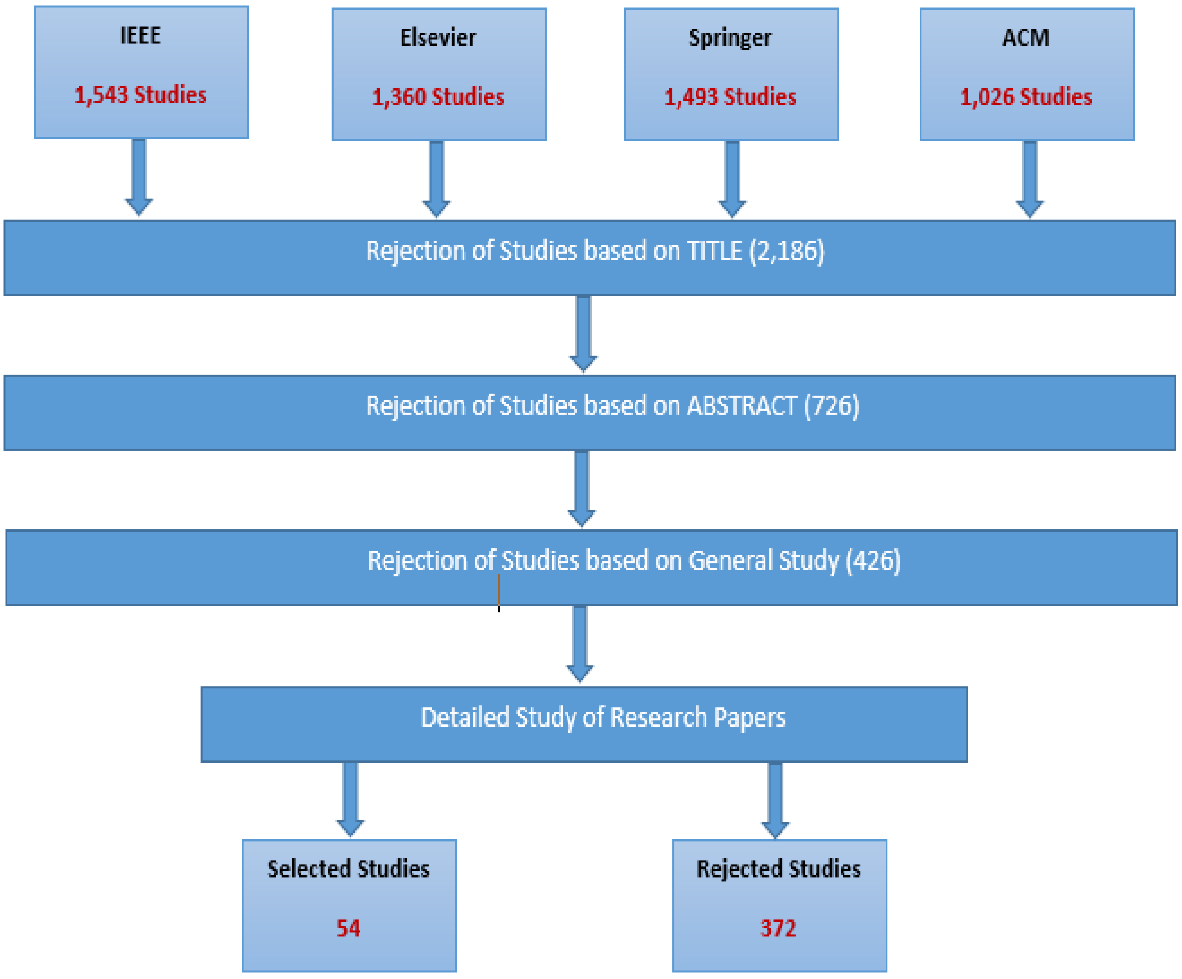

IEEE
1,543 Studies
Elsevier
1,360 Studies
Springer
1,493 Studies
ACM
1,026 Studies
Rejection of Studies based on TITLE (2,186)
Rejection of Studies based on ABSTRACT (726)
Rejection of Studies based on General Study (426)
Detailed Study of Research Papers
Selected Studies
54
Rejected Studies
372

Figure 2.2: Search Process

Table 2.2: Selected research papers w.r.t to publication type and databases

Database	**Type**	**Number of studies**	**References**	**Total**
IEEE	Journal	7	[9][23][24][34][37][40][43]	33
	Conference	26	[7][8][10][11][12][13][14][15][16][21][22][25][27][28][29] [30][31][32][33][34][35][38][39][44][46][54]	
Elsevier	Journal	3	[17][56][53]	4
	Conference	1	[18]	
Springer	Journal	8	[47][48][49][50][51][52][53][60]	10
	Conference	2	[20][55]	
ACM	Journal	0		7
	Conference	7	[19][26][41][42][45][58][59]	

2.2.4 Quality Assessment

Only high impact research studies are selected from highly recognized databases to assure the reliable results of this systematic literature review. Thirty-three (33) studies are selected from IEEE database, ten (10) from Springer, seven (7) from ACM and four (4) from Elsevier, making a total 54 selected studies. The results shown in Table indicate that we have tried our best to select latest and high impact studies. The summary of selected research papers w.r.t publication type and database is shown in Table 2.2. Database show the name of different research papers repositories; type field represents whether research studies are Journal or conference papers. References for the given studies are given. Total field represents the total no of journal and conference papers selected for each database to perform this SLR.

Table show that seven (7) journals and 26 conference papers selected from IEEE repository, three (3) journals and one (1) conference paper selected from Elsevier repository, eight (8) journals and two (2) conference papers selected from Springer link and seven (7) conference papers selected from ACM database. We have selected research studies from 2012 to onward. We have selected all the most relevant journals and conference papers from 2013 to onward which included four (4) studies from 2012, five (5) from 2013, seven (7) from 2014, five (5) from 2015, four (4) from 2016, eight each from 2017 & 2018 and thirteen (13) from 2019 as shown in figure 2.3.

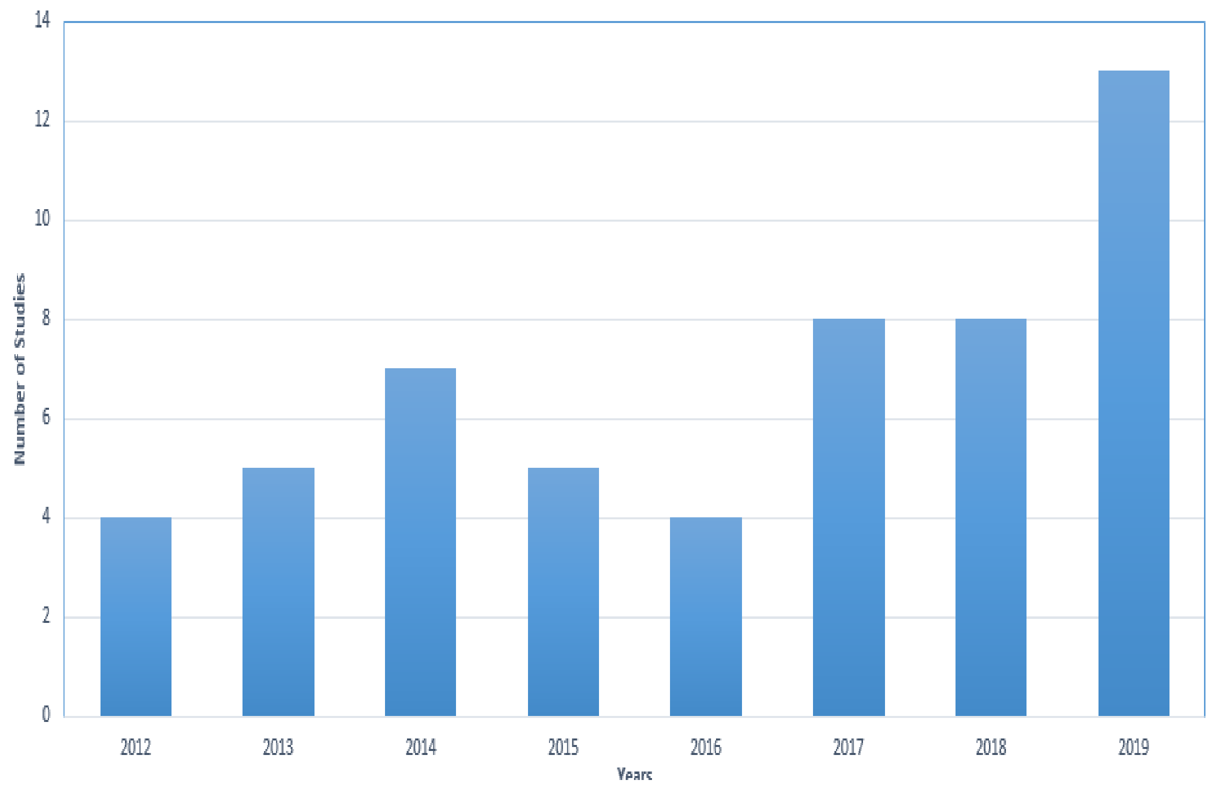


Figure 2.3: Distribution of selected researches w.r.t year of publication

2.2.5 Data Extraction and Synthesis

Table 2.3 shows this process. The mining process for bibliographic information of selected research studies are performed. We have defined overview, results, assumptions and validation in data extraction process to ensure the preciseness of research questions. For the data synthesis process, we have defined four points such as “Identification and categorization w.r.t techniques used”, “Identification of NLP techniques used”, “Identification and classifications of used/purposed tools” and “Comparison of the tools”. In first point, all the techniques used for the labeling of bug reports are identified and classified shown in Table 2.4. Secondly, all the NLP techniques used for the labeling of bug reports are identified, shown in table 2.12. All the purposed/developed tools also found and classified, shown in table 2.13. Tool comparison is shown in table 2.14.

Table 2.3: Data extraction and Synthesis

Data Extraction		
Sr. #	**Description**	**Details**
1	Bibliographic Information	Title, authors, year of publication and research study types are analyzed
2	Overview	The goal and idea of the selected studies
3	Results	Obtained Results from the selected studies
4	Assumption	Assumptions to validate the outcomes
5	Validation	Validation Techniques used

Data Synthesis		
Sr. #	**Description**	**Details**
1	Identification and categorization w.r.t techniques used	Identification and categorization of the papers with respect to the techniques author used
2	Identification of NLP techniques used	Identifications of NLP techniques used
3	Identification and classifications of used/purposed tools	All the purposed/used tools identified and classified
4	Comparison of the tools	Tools are compared

2.3 Results and Analysis

Basic goal of this research is to examine the related literatures conforming to the research questions. Out of 54 researches, 36 are published in conferences and 18 are published in journals. It is also noticed that journals such as “information and software technology”, “the journal of systems and software”, “automated software engineering”, “journal of computer science and technology”, “empirical software engineering”, “IEEE transaction on software engineering”, “IEEE transaction on fuzzy system”, “IEEE transaction on reliability” and “IEEE access are really contributing to study”. Also many conferences such as “International Conference on Reliability”, “Optimization and Information Technology”, “Conference on Informatics”, “Electronics & Vision”, “IEEE International Conference on Smart Computing”, “IEEE International Conference on Software Quality, Reliability and Security”, “IEEE Recent Advances in Intelligent Computational Systems” and “Asia-Pacific Software Engineering Conference” are highly contributing to our study.

2.3.1 Classification with respect to Approaches

All these studies are divided into five categories according to approaches used, shown in table 2.4.

Table 2.4: Categorization with respect to techniques

Sr. #	Category	References	Total
1	Machine Learning	[7][9][10][11][12][14][17][18][19][20][21][22][23] [24][25][26][27][29][33][34][35][36][37][39][41][42] [43][44][46][55][56][57][58][48][49][50][51][52][54] [53]	40
2	Knowledge Based	[9][11][17][20][28][30][32][35][38][40][43][45][56][59] [47][49][52][54][53]	19
3	Rule Based	[13][17][19][23][27][34][37][45][56][50]	10
4	Meta Algorithms	[9][28][23][56]	3
5	General	[8][13][14][16][17][19][20][22][24][29][37][39][41][43] [56][48][50][54]	18

2.3.1.1. Machine Learning Algorithms

ML is a "study of statistical models and algorithms which computer used to operate specific tasks without using explicit instructions". It is an Artificial Intelligence (AI)'s application that allows computers to learn accordingly and improved from the experience without any programming. ML algorithms are usually categorized into two type explained below.

2.3.1.1.1. Supervised Machine Learning Algorithms

In this type class labels of some data are known; this data is called training data. The training data is used to classify new data, this new data is called testing data. There are one or many inputs and desired output for each training example. It's techniques with references shown in table 2.5.

Table 2.5: Research studies using Supervised Machine Learning Algorithms

Sr. No	Algorithms	References	Objective
1	Naïve Bayesian	[7][9][10][11][14][17][18][19][21][22][23][24][25][33][34] [35][36][37][41][43] [55][58][48][49][50][51][52][54] [53]	For text categorization Tasks
2	Decision Tree	[10][22] [23] [24][25][36][41] [43] [46] [50][52]	Used for classification
3	Logistic Regression	[9][10][12][17][22][29][37][46] [55][56] [54]	Binary classification
4	K Nearest Neighbor	[9][11][18][19][21] [23][24][34][43][49][50][52]	classification and regression
5	Random Forest	[9][11][17][19][26][29][36][50][52]	classification and regression
7	Support Vector Machine (SVM)	[11][17] [19][20] [21][22] [23] [24][37] [39] [41] [55][58] [49][50][51][52] [53] [54]	Classify the data points using hyperplane
8	Artificial Neural Network	[9][12][19][21][22][24][27] [56][50] [60]	Used for classification

2.3.1.1.2. Unsupervised Machine Learning Algorithms

In this type class labels of training data are unknown. These approaches get a set of data which consists inputs only and find the structure of data such as clustering or grouping of the data points. Therefore, these approaches learned from the data which is not labeled, categorized or classified. It's techniques with references shown in table 2.6.

Table 2.6: Research studies using Unsupervised Machine Learning Algorithms

Sr. #	Techniques	References	Objective
1	Clustering	[17] [44]	Classification purpose
2	Hierarchical Dirichlet Process (HDP)	[44]	Grouped the data
3	Latent Dirichlet Allocation (LDA)	[26][33][41][42][46]	Grouping of textual data
4	Markov model	[55][57]	Model the Probabilities of different states an transitions

2.3.1.2. Knowledge Based Algorithms

Some researchers used knowledge-based approaches to label the bug reports. Knowledge based systems that is used to label the bug reports, consists a large amount of data as knowledge based. This knowledge base is like dictionary that contain information, facts and meaning of various words that used in real word. This knowledge and information can have applied for the labelling of bug reports. Knowledge based techniques with references shown in table 2.7.

Table 2.7: Research studies using Knowledge Based Algorithms

Sr. #	Techniques	References	Objective
1	Knowledge Based	[9][11][17][20][28][30][32][35][38][40][43][45][56][59][47] [49][52][54][53]	Use knowledge For categorization

2.3.1.3. Rule Based Algorithms

Many researchers have used rule-based approaches for the labelling of bug reports. In this approach, rules are defined to find the required content from natural language. Researchers have defined many rules for the labelling of bug reports. In this approach,

NLP techniques like stop words removal, stemming and POS Tagging are enforced on the sentences then outcomes are matched with the pre-defined rules to categorize the bug reports. Rule based techniques with references shown in table 2.8.

Table 2.8: Research studies using Rule Based Algorithms

Sr. #	Techniques	References	Objective
1	Rule Based	[13][17][19][23][27][34][37][45][56][50]	Define rules For categorization

2.3.1.4. Meta Algorithms

In meta algorithms, an algorithm is applied on many other algorithms to get the highest accuracy. In this, many other algorithms are combined to build a predictive model to get more accurate results like in machine learning, Naïve Bayesian, KNN and SVM's results are combined to get more accuracy. Some of the researchers used these techniques for the labelling of bug reports. Meta algorithms with references shown in table 2.9.

Table 2.9: Research studies using Meta-Algorithms

Sr. #	Techniques	References	Objective
1	Boosting	[19][23]	To improve accuracy
2	Bagging	[23]	For the improvement of accuracy
3	Voting	[23][56]	Improve accuracy
9	Stacking	[19]	Improve accuracy

2.3.1.5. General

In general category, those techniques are included which are not belong to any specific category. Some of the researchers used techniques which can't belong to any categories mentioned above like latent-semantic-indexing (LSI) is an indexing-technique which can't belongs to any categories shown in table 2.10.

Table 2.10: Research studies using General

Sr. #	Techniques	References	Objective
1	Latent semantic indexing (LSI)	[14]	Identify patterns in the relationship between terms and concepts
2	Grid Search	[17][20]	Used for objective function with no parameters
3	Abstract Syntax Tree (AST)	[13][54]	For the static analysis of code

4	Spearman's Rank Correlation Coefficient	[16]	Describe relationship between two variables
5	Wilcoxon Rank Sum Test	[16]	Calculate difference between set of pairs
6	MATHEW Correlation Cofficient	[17][29]	Measure the quality of binary classifications
7	Kullbach Leibler Divergence (lda-kl)	[8][23][24][39][41][48][50]	Use for recommendations
8	Relief-F attribute selection(RFS)	[23]	Attribute Selection
9	Chi Squared attribute selection (CHI)	[8][23][24][29]	Attribute selection
10	particle swarm optimization (PSO)	[24]	Trying to improve a candidate solution
11	Hamming-loss	[37]	calculate the error
12	TSVD approach	[43]	For solving the linear discrete ill-posed problem

2.3.2. Classification with respect to library/ technology used

All the researches also analyzed for technology and library used for the labeling of bug reports. Different kinds of technology or library used by different studies are shown in table. Natural language toolkit (NLKT) is used by seven (7) researches which is most used library, Scikit-learn library is used by five (5) researches, Porter stemmer is used by five (5) studies, Word2vec is used by four (4) researches and all the remaining libraries is used by one or two studies shown in table 2.11.

Table 2.11: Libraries/technologies used for labeling

Sr. #	Technologies/libraries	References	Total
1	Scikit-learn	[8][20][24][50][56]	5
2	Natural language toolkit (NLKT)	[8][11][20][24][27][37][54]	7
3	Word2Vec	[12][17][19][20]	4
4	Doc2Vec	[17][25]	2
5	Wordnet	[37][53]	2
6	lingPipe	[14][41]	2
7	LibSVM	[20]	1
8	Stanford Topic Model-ing Toolbox (TMT)	[26]	1

9	Mulan	[34]	1
10	JOERN	[39]	1
11	CodeSonar	[40]	1
12	Coverity	[40]	1
13	oTranscribe	[40]	1
14	Indo-european tokenization factory	[41]	1
15	Porter Stemmer	[8][21][34][46][58]	5
16	Lovins Stemmer	[21][49]	2
17	NumPy	[50]	1
18	SciPy	[50]	1
19	Mahout	[54]	1
20	Lucene	[54]	1
21	Stanford Natural Language Inference corpus (SNLI)	[55]	1
22	Paice Husk Stemmer	[21]	1
23	Weka	[22][23][29][39][44][58][54]	7
24	LLVM	[13]	1

2.3.3. Classification with respect to NLP Techniques

A number of NLP techniques are used by researches for the processing of textual data. We have reviewed all selected studies and identify five (5) NLP techniques used for the labeling of bug reports. We have categorized the selected literatures into five (5) categories with respect to NLP techniques, shown in table 2.12.

Table 2.12: NLP Techniques used for Labeling

Sr. #	**NLP Techniques**	**References**	**Total**
1	Tokenization	[8][9][10][14][18][20][22][23][25][27][29][34][36][37][41][42][43][44][55][58][48][53][54]	23
2	POS Tagging	[10][11][27][29][37][44][55]	7
3	Parsing	[10][29][43][44][46][52]	5
4	Stemming	[8][11][14][17][18][21][22][23][25][34][43][46][58][49][51][52][53][54]	18

5	TF-IDF	[7][9][10][11][14][17][18][24][35][41][48][49][50][55]	14

Tokenization is an NLP Technique which is used to split the natural language paragraph into sentences, then these sentences are broken into tokens. POS Tagging technique is used to designate part of speech to each word in a sentence like noun, verb, adverb etc. Parsing is used to build a syntax tree of the sentences, stemming returns the words to its base form and TF-IDF used to identify the value of a word in a paragraph.

Tokenization is most used NLP technique in the selected literatures which is used in twenty-three (23) studies. Stemming used in eighteen (18) researches, TF-IDF used in fourteen (14) studies, POS Tagging used in seven (7) studies and parsing is used in five (5) research papers.

2.3.4. Identification of Purposed/developed tools

We identified thirteen (13) purposed or developed tools, shown in table 2.13. These tools have been used by different researchers for labeling of bug reports. Bug mining system (BMS) used ML techniques for the identification of security bug reports. FARSEC is used for the filtering and ranking of security bug reports. CTES used rule based technique for the classification of the bug reports. AVIS used ML techniques to detect the vulnerability. USES utilized ML techniques to identify the Mandel and Bohr bugs. Im-ML.KNN is multi label learning tool used to achieve better performance. [17] purposed a tool by using ML and rule-based techniques to classify the bug reports. Flower is a tool that help the developers to navigate the flow of program. [42] developed a tool by using ML techniques for classification of the bug reports. DRONE, BUGBAND and Auto ODC developed by different researchers for the classifications of bug reports.

Table 2.13: Tools purposed/developed by different researchers

Sr. #	Tool Name	References	Purpose
1	BMS	[7]	Identify Security and Non-Security Bug Reports
2	FARSEC	[9]	A tool to reduce the mislabeling of Security bug reports
3	CTES	[13]	To automatically classify the security bugs
4	AVIS	[19]	To detect vulnerability using ML techniques
5	USES	[22]	To distinguish Mandel bugs from Bohr bugs.
6	(Im-ML.KNN)	[34]	Multi label learning Tool, to achieve better performance.
7	TOOL BY [17]	[17]	Classify bug reports according to taxonomy
8	Flower	[40]	A tool that help developers to navigate program flow
9	Tool by [42]	[42]	To classify bug reports

10	DRONE	[48]	To assign priority labels to bug reports
11	BUGBANG	[52]	To classify bug reports
12	Auto ODC	[53]	A tool to automate ODC classification
13	Tool by [56]	[56]	Classify the bug reports

2.3.5. Comparisons of purposed/developed Tools

We performed state of the art comparison of all identified tools based on some parameters, shown in table 2.14. The parameters used for the comparison are approaches used, NLP techniques, programming language and open source. Used approaches are machine learning, knowledge based, rule based, meta algorithms and general, these categories of approaches are already defined in section. These identified tools used different NLP techniques for the labeling of bug reports, we also used these techniques for the comparison of tools. Programming language used to develop the tools is also a parameter for comparison and open source field contain Yes or No values based on the source.

Table 2.14: Comparisons of Tools

Sr. #	Tool Name	Approaches Used	NLP Techniques used	Programming Language used	Open Source
1	BMS [7]	ML	TF-IDF	-/-	No
2	FARSEC [9]	ML, KB	Tokenization, TF-IDF	Python	Yes
3	CTES [13]	RB, G	-	C++	No
4	AVIS [19]	ML, MA, RB	-		N0
5	USES [22]	ML	Tokenization, Stemming	JAVA	No
6	(Im-ML.KNN) [34]	ML, RB	Tokenization, Stemming	JAVA	NO
7	TOOL BY [17]	ML, RB, MA	Stemming, TF-IDF	Python	No
8	Flower [40]	KB	-	Java	Yes
9	Tool by [42]	ML, MA	Tokenization	-/-	No
10	DRONE [48]	ML, G	Tokenization, TF-IDF	Java	No
11	BUGBANG [52]	ML, KB, RB	Stemming, Parsing	R	Yes
12	Auto ODC [53]	ML KB	Tokenization, Stemming	-/-	No
13	Tool by [56]	KB, RB, G	-	Python	Yes

2.4 Answers of Research Questions

RQ1: What are the primary techniques used for the labeling/classifications of bug reports?

Answer: To answer the research questions, 54 research studies published between 2012 and 2020 have selected as per the selection and the rejection criteria (Section 2.2). These studies are grouped in five categories with respect to approaches used by different researchers.

- Forty (40) research studies have used machine learning algorithms, discussed in section 3.1.1.
- Nineteen (19) research studies have used knowledge-based techniques mentioned in section 3.1.2.
- Rule based approaches have founded in ten (10) research papers mentioned in section 2.3.1.3.
- Some studies have used meta algorithms, discussed in section 2.3.1.4.

There are some techniques which can't belong to previously discussed four categories, these approaches have categorized as general, eighteen (18) studies have used these approaches mentioned in section 2.3.1.5

RQ2: What are the tools purposed/developed for labeling/classifications of bug reports and what is the difference between these tools?

Answer: After conducting the systematic review of the selected studies, we have found thirteen (13) tools which are purposed or developed by different researchers, these tools shown in table 2.13. The difference between these identified tools is also analyzed shown in table 2.14.

RQ3: What are the primary NLP techniques used for the labeling/classifications of the bug reports?

Answer: After the analysis of selected literatures, we have identified five (5) primary NLP techniques, researchers used for the labeling/classification of bug reports

Tokenization Have used by twenty-three (23) researches, POS Tagging have been used by seven (7) studies, Parsing have been performed in five (5) literatures, stemming performed by eighteen (18) researchers and TF-IDF have been used by fourteen (14) studies, mentioned in section 2.3.3.

RQ4: What are the libraries/technologies used for the labeling/classifications of bug reports?

Answer: After conducting the systematic review of the studies, we have found twenty-four (24) different libraries and technologies used by different researchers for the labeling/classification of bug reports, these libraries/technologies are shown in table 2.11.

RQ5: How did these modern tools and techniques help in fixing of bugs?

Answer: Due to the limited time and the human-resources, it's very problematic for the developers to resolve all bug-reports. Developers usually require to arrange the reports to resolve the bugs quickly. Label or type field in bug reports describes the type of bug reports.

Label can give an insight about the bug reports and also be used for the priority of bug reports. Some bug reports like security bug reports needs to be fixed as early as possible. In this SLR, many tools and techniques are identified that can help in the process of labeling/classifications of bug reports. These labels on bug reports helps developers understating the type of bugs. Developers used these labels for the fixing of bug of high priority bug reports.

2.5 SLR Conclusion

This research identifies the latest techniques and the tools utilized for the automatic labeling or classification of bug reports. To achieve this aim, firstly five research questions are identified. Then done a detailed SLR of 54 literatures which were selected conferring selection and rejection criterion. All the chosen studies were categorized into five categories conferring to the approaches used.

Then we have found thirteen (13) tools that are purposed or developed by different researchers. We have also found different NLP techniques used by different researchers and categorized all selected studies according to these NLP techniques. We have also identified the twenty-four libraries/technologies used for the labeling/classifications of bug reports by different literatures. Then we have performed a comparison of identified or developed tools.

Chapter 3

Proposed Methodology & Implementation

CHAPTER 3: PROPOSED METHODOLOGY & IMPLEMENTATION

This chapter includes details of methodology and the implementation. **Section 3.1** provides the core concepts which used in our purposed methodology. **Section 3.2** contains details of purposed methodology. **Section 3.3** includes the implementation of our purposed methodology.

3.1 Core Concepts Explanation

For our purposed methodology, we have utilized many core concepts from text mining and machine learning. The details of core concepts used in our methodology are explained bellow.

3.1.1 Natural Language Processing

Natural language processing (NLP) is a potential of a program to interpret language of human, understand it and can manipulate it. It is a branch of artificial intelligence (AI). Basically, it draws from many fields such as computational linguistics and computer science, try to fill the disparity between computer understanding and human communication. NLP concerns with automatic analysis of plain natural language [62]. NLP is not a new filed, it is growing rapidly because of increased in interest in human machine communications.

This field is also growing because of availability of the big data and powerful computing algorithms. It can teach computers that how to understand and process the natural language. NLP is utilized to help the programmers to govern and organize the work knowledge to perform their tasks like summarization, named entity relationship, translation, information retrieval or relationship extraction, speech recognition and topic segmentation etc. [63]. This field helps computers and create automated systems that can understand and analyze a human language like Arabic, Latin or English etc.

3.1.2 Tokenization

Tokenization is a technique of splitting or tokenizing a string or text into lists of tokens. It is a commonly used text mining technique that involves splitting the text into sentences or words [64]. The basic function of tokenization is to find and split the tokens found in a text that each word and punctuations will become a distant token [65]. Tokenization is divided into two submodules named word tokenization and sentence tokenization explained below.

3.1.2.1 Word Tokenization

Word tokenization is used for the splitting of a sentence into words. The word tokenization's output is transformed into data frame for the more understanding of text in machine learning algorithms. Machine learning algorithms need a numeric data for the training and prediction. Word tokenization can convert a text string to numeric data. Word tokenization is explained with example in table 3.1.

Table 3.1: Word level tokenization

Sentences	**Word Tokenization**
It is a simple sentence	‘It’ ‘is’ ‘a’ ‘simple’ ‘sentence’
That is exactly what we want to learn	‘That’ ‘is’ ‘exactly’ ‘what’ ‘we’ ‘want’ ‘to’ ‘learn’
That is not a pencil	‘That’ ‘is’ ‘not’ ‘a’ ‘pencil’

3.1.2.2 Sentence Tokenization

It is a technique of dividing a paragraph into many sentences. Sentence tokenization is used when we want to count average words per sentence. It performed less well in for electronic health records that includes abbreviations, medical terms measurements and other forms not found in standard written English. Sentence tokenization is explained with examples in table 3.2.

Table 3.2: Sentence level tokenization

Paragraph	**Sentence Tokenization**
It is a technique of dividing a paragraph into many sentences. Sentence tokenization is used when we want to count average words per sentence. It performed less well in for electronic health records that includes abbreviations, medical terms measurements and other forms not found in standard written English.	"It is a technique of dividing a paragraph into many sentences". "Sentence tokenization is used when we want to count average words per sentence". "It performed less well in for electronic health records that includes abbreviations, medical terms measurements and other forms not found in standard written English".

3.1.3 Stop Words Removal

In NLP, idle words are considered as stop words. A stop word is a commonly used word that is overlooked by search engines, we would not want these idle words to take a space in our datasets. Most commonly used stop words are "the", "a", "an", "in", "a". we don't need these words during the processing or training because these words are not going to help in building the training model and will cost useless processing/computing power. Processing time and memory is very valuable in case of language processing, so we cannot let this useless data to increase the processing time and taking up extra memory. We can remove stop words easily by storing a list of words which are unnecessary. We have used Python's tool NLTK for the removal of these stop words. Examples of stop words are given in table 3.3

Table 3.3: Stop words removal

Text with stop words	**Text without stop words**
It's difficult for the developers to resolve all bug reports	Difficult, Developers, Resolve, Bug, Reports
That is exactly what we want to learn	Exactly, Want, Learn
I like reading, so I read	Like, Reading, Read

3.1.4 Stemming

Stemming is a technique used for the normalization of words [63]. Basically, it is a process of removing the suffix from a word and reduce to its base form. It is utilized for the reduction of dimensionality of data, that is good for machine learning algorithms. A stemming algorithm can reduce the words to their base forms. Simply we can say that, if there are words like 'work', 'works', 'worked', 'working,' all these words are contextually same but different words. We can remove the suffixes of these words, 'work' will be the stemmed word. We have used porter stemmer [66] library of NLP for the stemming process in our purposed methodology. Stemming example show in table 3.4.

Table 3.4: Stemming of words

Words	Stemmed Words
Sleeping, Slept, Sleeps	Sleep
Reading, Read, Reads	Read
Add, Adding, Added, Adds	Add
Eating, Eats, Eaten	Eat

3.1.5 Machine Learning

ML is a "study of statistical models and algorithms which computer used to operate specific tasks without using explicit instructions". It is an Artificial Intelligence (AI)'s application that allows computers to learn accordingly and improved from the experience without any programming. ML algorithms are usually categorized into two type. One is supervised machine learning and other one is unsupervised machine learning. In first type, class labels of some data are known, this data is called training data. New data is classified depend on the training data; this new data is called testing data.

There are one or many inputs and liked output for each training example. In unsupervised machine learning algorithms, class labels of training data are unknown.[10]. We have chosen Logistic regression(LR), Naïve Bayes (NB), and Decision Tree (DT) algorithms from supervised ML for our approach because they performed very well for the labeling of bug reports.

3.1.5.1 Supervised Machine Learning

In this learning there are some input variables (a) and an output variable (b). The mapping function is learned from input to put by using an algorithm

$b = f(a)$

The objective is to estimate the mapping function so accurate when an unseen input data (a) which can forecast the output variables (b).

In this learning, it is considered that a teacher is supervising the whole learning process of a model from training data. When the technique accomplishes a sufficient level of efficiency learning will be stopped.

Supervised learning further branched into the regression and classification problems.

Classification

In this problem the output variables are categories, like "black" or "white" and "disease" or "not a disease".

Regression

In this problem the output variables are real values, like "dollars" or "weight".

3.1.5.2 Unsupervised Machine Learning

In this type class labels of training data are unknown. These approaches get a set of data which consists inputs only and find the structure of data such as clustering or grouping of the data points. Therefore, these approaches learned from the data which is not labeled, categorized or classified. This learning is further divided into the clustering and association problems.

Clustering

In this problem objective is to disclose the internal groupings presents in the data, like grouping the customers according to their behavior of purchasing. K-means is a popular ML approach for the clustering problem.

Association

In association learning problem objective is to identify the rules which show maximum parts of the data, like people whose buy X also used to buy Y.

3.1.6 Training & Prediction Phases

In training phase ML algorithms are providing with the training data in order to learn from it. The training data will contain the true answers, that are known as "*target*" or "*target attributes*". These learning algorithms identify patterns in the training data which map the input data features to the target (An answer which we want to predict), and ML model is obtained as output which captured these-patterns.

In prediction phase, a prediction or testing is utilized for the evaluation of a machine learning model. In this phase a machine learning model is validated on a testing data, whose target attributes are missing.

ML model is used for the predictions on new data about which you do not know the outputs. In our approach we want to train multiple ML models for the predictions of SBRs and NSBRs. We provided our approach with the training data which contains bug reports for which target is known. In our approach models are trained by this data to predict whether new bug reports are SBRs or NSBRs.

3.2 Purposed Solution

Our objective here is to purpose an approach for the automatic labeling of SBRs. Our purposed approach performed better than previous approaches. This approach builds a classification model from historically labeled bug report for the identification of SBRs. Firstly, preprocessing is performed on bug reports.
Then we performed feature selection using a TF-IDF values. After feature selection, a class imbalance sampling technique is applied on the training data. We have applied three ML algorithms, named Logistic Regression and Decision Tree and Naïve Bayes. At the end a voting technique is applied on the results of these machine learning algorithms to get a better results. Our purposed approach is shown in figure 3.1

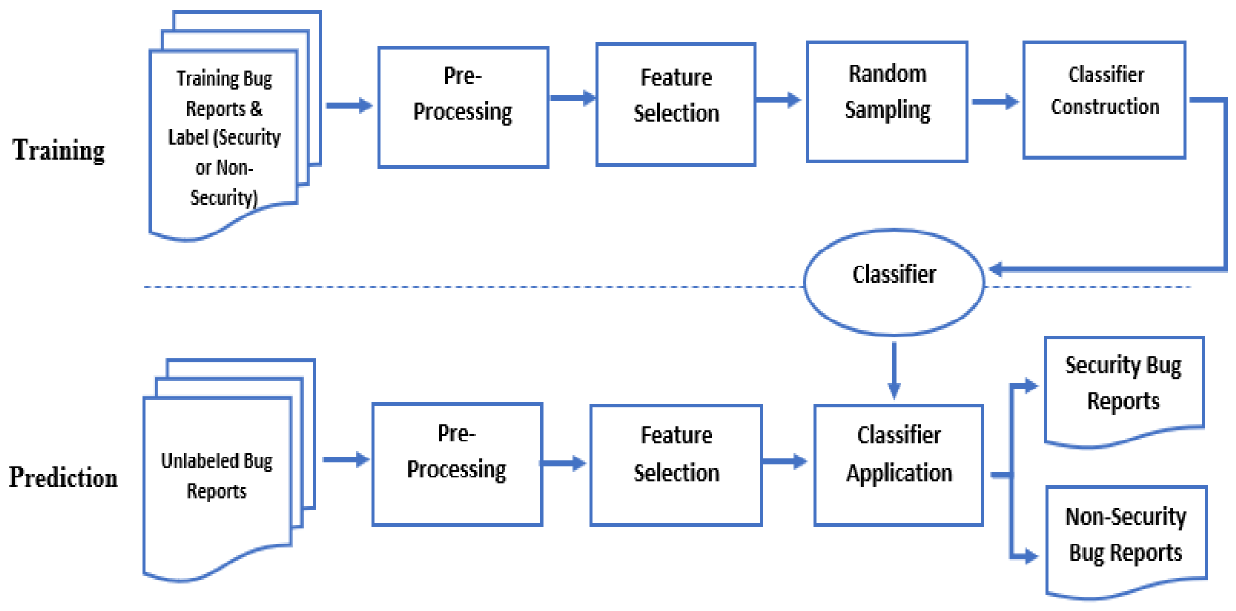


Figure 3.1: Purposed approach

3.2.1 Preprocessing

Preprocessing of bug reports is the first step in our purposed technique. We only considered the description and summary column of bug-reports as a textual data. Summary provide us an overview of bug reports in one sentence and description mostly encompasses more information. These fields are commonly available when a bug report is submitted. Preprocessing is applied on this textual information of both training and testing bug reports. A textual information is transformed into a set of features by using Python's NLTK [70] standard preprocessing techniques.

First step of preprocessing is tokenization of words. Tokenization of description and summary fields of bug reports are performed. As result of tokenization, terms are extracted from bug reports. These extracted terms are converted to lowercase forms. Punctuation removal is also performed. These terms also include stop words. In NLP, idle words are considered as stop words. These are mostly used words that are overlooked by search engines, we would not want these words to take a space in our datasets. Most commonly used stop words are "the", "a", "an", "in", "a". These stop words are removed using NLTK's stop words list. During stop words removal, unwanted terms are also removed.

After the removal of stop words stemming is applied on the terms. Stemming is a process used for the normalization of words [63]. Basically, it is a process of removing the suffix from a word and reduce to its base form. Porter Stemmer is applied for the process of stemming. After the stemming, all terms are return to their base forms. These terms are considered as extracted features. Preprocessing steps are shown in figure 3.2,

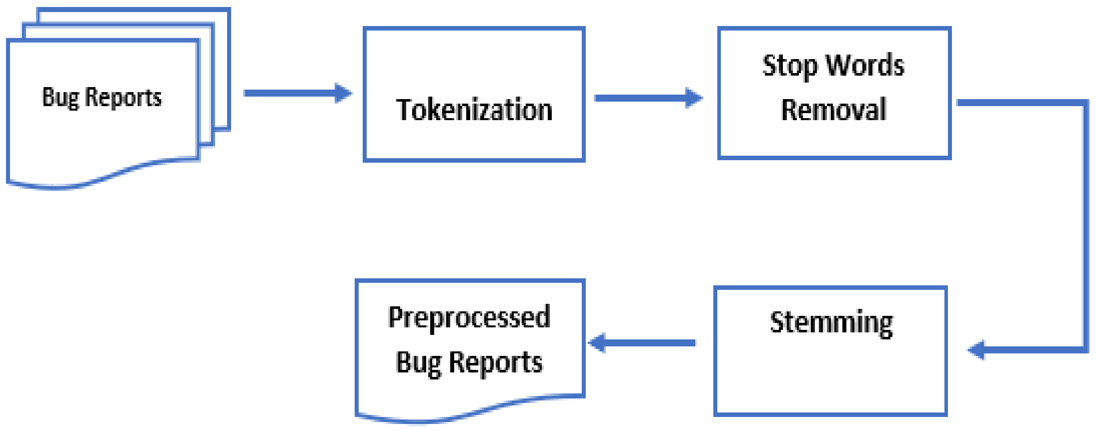


Figure 3.2: Preprocessing of bug reports

3.2.2 Feature Selection

Security bug reports identification is thought as a text categorization problem. After the preprocessing, extracted terms are considered as features. The feature space's dimension is very high for text categorization. The performance of a classification models will degrade because of high this dimension [71]. We have selected those features which are more important than others by applying TF-IDF. In our purposed methodology, we selected the top 50 terms each from both categories of bug reports (SBRs and NSBRs) with the maximum TF-IDF values as our feature set [14]. We contracted the feature set to hundred because hundred features covered the feature families [72].

3.2.2.1 TF-IDF

The abbreviation of TF-IDF is "term frequency–inverse document frequency". TF-IDF is commonly utilized to measure the usefulness of words to a document in a corpus [66]. TF-IDF weight is commonly utilized in text mining.

In this, words which are frequent in every document, such as "it", "who", and "so", rank less even still these are present many times, after all they are not important that document. But, if a word "Bug" presents many times in a document and not presenting many times in other documents, it doubtlessly means that it is most relevant.

By multiplying two different terms TF-IDF for a word in a document is computed:

The Term Frequency (TF)

There are many methods to compute Term frequency. The simplest and easiest method is a rough count of word occurs in a document. There are ways to accommodate the frequency, one is by the raw frequency of the most common word in a document and other is by length of a document.

The Inverse Document Frequency (IDF)

IDF computes that how much a word is common or rare to whole document set. If its value is near to 0, the word will be more common. It is computed by counting the total number of documents and dividing it by the number of documents which contain this word, and at the end calculate the logarithm.

Therefore, this number will be about “0”, if the word is most common and occurs in more documents. Or else, it will be about “1”.

By multiplication of these two numbers, TF-IDF values will obtained. The maximum value, the more relevant this word in that specific document.

TF (term, doc) = count of term in doc / number of words in doc
IDF (term) = log (n/(df + 1))
TF-IDF = (TF * IDF)

“Term” stands for word (term), “doc” stands for document, “n” stands for count of corpus. Corpus are the total documents.

3.2.3 Random Sampling

Class unbalancing is consistently a major issue in ML. It may cause a classifier to perform badly. Imbalanced learning strategies will be utilized for the balancing of the preliminary unbalanced dataset and assisting the trained model to not to be biased to the majority magnificence. Hence, in maximum cases, it is able to improve the efficiency of the classifier [73].
Three famous sampling strategies are under sampling, oversampling and the SMOTE. We utilized random under sampling, as the performance of this method is better in maximum cases [74].

3.2.3.1 Rando Under Sampling (RUS)

RUS involves randomly selecting samples from the maximum class to eliminate these from the training dataset. This has the impact of reducing the quantity of samples in the maximum class in the converted version of the training dataset. This procedure can be replicated until the preferred class distribution is obtained, such as an identical number of samples for each class.

Under sampling reruns, the below mentioned 2 steps until a preferred ratio of maximum samples to all the samples reaches to “r”:

Step1: **Sample Selection**

Step2: Sample Deletion

Because of outstanding performance, we used RUS [74]. We set the value of r as .5 declaring that the variety of samples of both classes is identical within the training data.

3.2.4 Classifier Construction

We have used three machine learning algorithms named Decision Tree, Logistic Regression and Naïve Bayes in our approach. A voting strategy is also used to get the better result. Details of these algorithms are explained below.

3.2.4.1 Naïve Bayes (NB)

Naïve Bayes is one of the most effective and efficient algorithms in machine learning [67]. Study shows that it performed very well as compare to other algorithms in defect prediction [68][69]. The NB is based on the Bayes' law that used independence assumptions between the predictors. A NB model can easily build. Even with its easiness, the NB model frequently does noticeably excellent and is extensively used because it mostly surpasses more known classification techniques. We have used Gaussian NB, which is the variant of NB.

3.2.4.2 Decision Tree (DT)

A decision tree is a supporting tool for making decision that utilizes a graph that is a tree. This tree is like a model of decisions and its potential effects, along with likelihood event results, costs of resources, and the utility. A DT displays an algorithm which only consists statements about conditional controls. A DT is like a flowchart who's each internal node presents a "test" on an attribute (e.g. if a bug report is SBR or NSBR), each branch shows the outcome of a test, and at the end each leaf node indicates a label of the class. The rules of classification are indicated by the roof to leaf paths.

Tree-based algorithms grant easy interpretation and stability with high accuracy to predictive models. They map-nonlinear relationships quite well unlike linear models. These algorithms are flexible at solving all types of problem such as, classification-or-regression. A DT algorithm is also known as CART (Classification and Regression Trees).

3.2.4.3 Logistic-Regression (LR)

Logistic-regression is performs well when the dependent variables are binary. A LR performs predictive analysis like all other regression analysis. It is utilized to define the data and for explanation of the relationship among one dependent binary variable and one or more independent variables.

In the early twentieth century, it was mostly utilized in the biological sciences. It was also utilized in many different social science's applications. LR is suitable when the resultant variable is a categorical.

As an example,

- To forecast if a bug report is SBR (1) or NSBR (0)
- If the email is spam (1) or not (0)

In our case where we want to classify a bug report as SBR or NSBR. Threshold value is necessary for classification in when linear regression is used.

Say if the real class is SBR and 0.4 is the forecasted value and 0.5 is the threshold value, bug report will be categorized as NSBR that can cause a consequence for the software. Linear regression is not appropriative for the classifications problem as seen from this example. LR is un-bounded, and logistic regression comes into picture. Their value are from zero to 1.

3.2.4.4 Voting

A Voting-Classifier which trains-on-an ensemble of many different models and forecasts an output which based on the maximum probability of selected class as an output.

It combines the outcomes of every classifier and processed these into a voting classifier. This classifier will forecast the outcome class which based on the maximum majority voting.

The concept is that rather creating individual devoted models and calculating the efficiency for each model, it creates one model that trains by these devoted models and forecasts the outcomes that based on their joined majorities voting for every output class.

There are two types of voting supported by Voting Classifier.

1. **Hard Voting**
   In hard voting, the forecasted output class is a class with the maximum majority of votes. For example, the class which has the highest probability of being predicted by each of the classifiers. Consider three classifiers forecasted the *output class (X, X, Y)*, so the majority predicted *X* as output class. So, *X* will be the final forecast.
2. **Soft Voting**
   In soft voting, the prediction is based on the probabilities' average given to that a specific class. Consider some input is given to three models, the forecast probability for class *X = (0.40, 0.37, 0.63)* and *Y = (0.30, 0.22, 0.50)*. the average for class *X is 0.4666* and *Y is 0.34*, So the winner is class *X* because it has the maximum probability averaged by each classifier.
   We have used soft voting for our approach.

3.3 Implementation

This section discussed the details of standard data set we used for our research and also elaborates the implementation details of our purposed approach.

3.3.1 Data Set

For the implementation of our purposed approach, we required labeled bug reports, that are labeled as SBRs or NSBRs. We utilized total 5 projects. Among these 5 projects, 4 are from-Ohira et al. [75]. A portion of reports is selected from the project of chromium. The details of these projects are presents in table 3.5. This table shows the name of projects, total number bug reports for every project and the percentage and the number of SBRs for each project. These are arranged in ascending order of security bug reports percentage.

Table 3.5: Details of data sets

Projects	**Total Bug Reports**	**Security Bug Reports**	**Security Bug Reports (%)**
Chromium	41,940	192	0.5
Wicket	1000	10	1.0
Ambari	1000	29	3.0
Camel	1000	32	3.0
Derby	1000	88	9.0

In Ohira's [75] data set, six kinds of bug reports are there. These includes dormant, security, blocking, performance, surprise and breakage. JIRA is used as their bug tracking system for these four projects and the application-domain of every project is different. As Ohira et al. [75] only concentrated on high impact bug reports, they only chose one thousand reports for each project randomly whose label are BUG or IMPROVEMENT. These bug reports are labelled by faculty members and graduate students.

The Chromium data set came from "mining challenge of the mining software repositories conference 2011". When these reports are submitted to the system, (SBRs) are labeled as Bug-Security. As our purposed methodology only focus on SBRs predictions, we consider all other type of bug reports as NSBRs.

The bug reports form Ohira et al. [75] are in comma separated value (CSV) files. Each row of these CSV files indicates a bug report and columns shows the attributes of bug reports. These attributes are issue_id, type, status, description, summary, and date & time of report submission and resolved. CSV file for Ohira project is shown in figure 3.3. For prediction of security bug reports, we only select summary, description and security fields for each project. Security fields with label '0' for NSBRs and '1' for SBRs for each project.

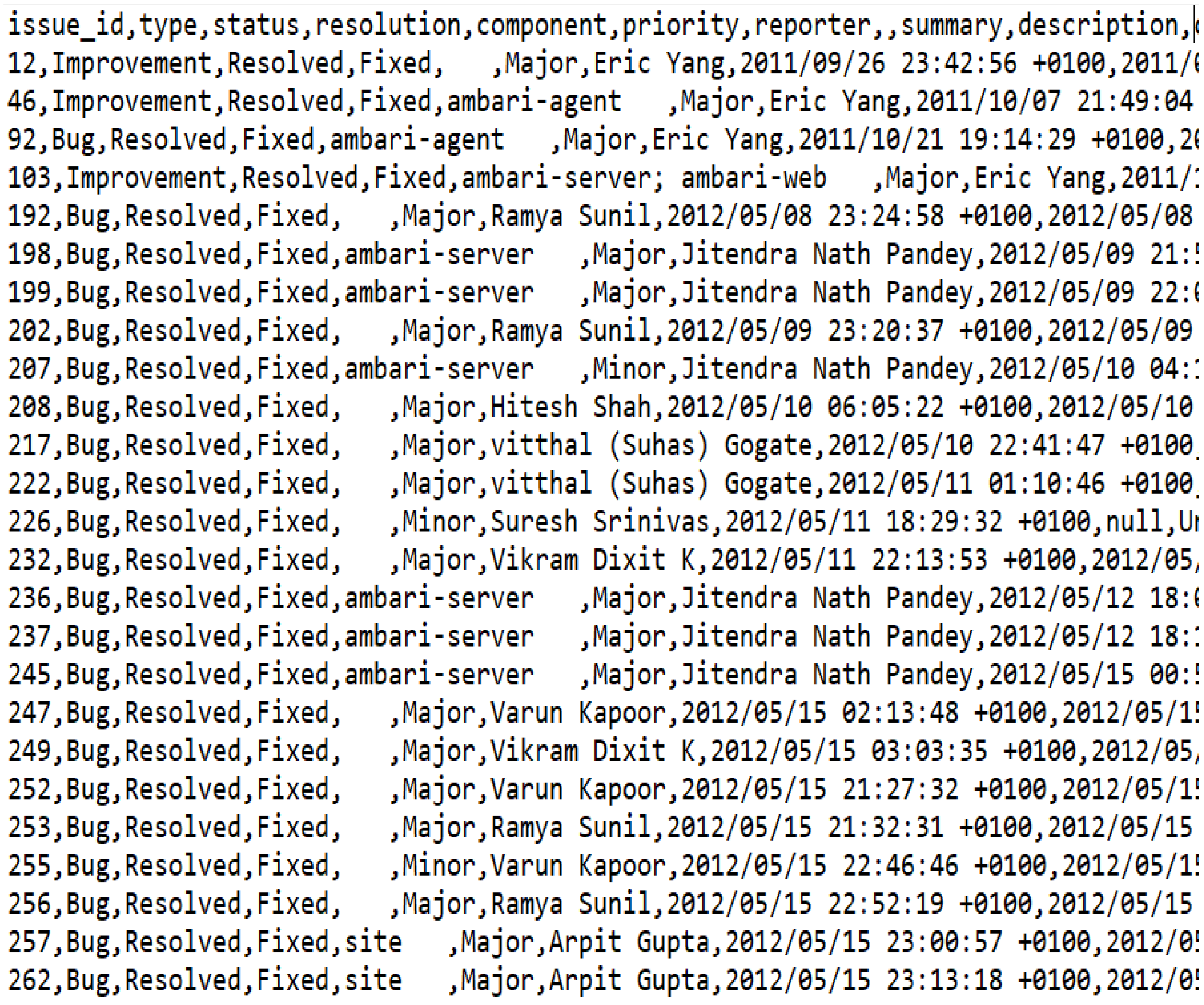
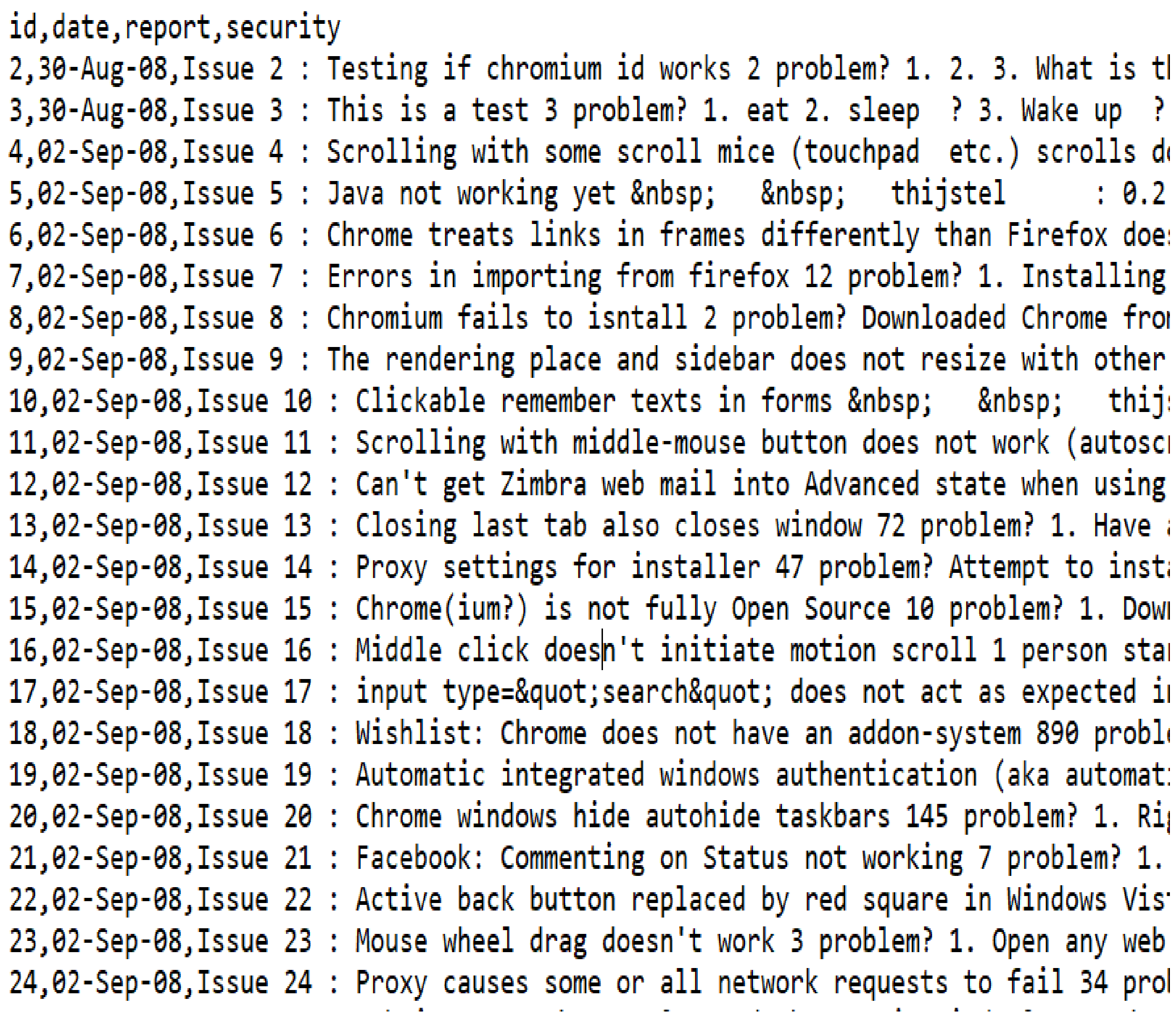

```
issue_id,type,status,resolution,component,priority,reporter,,summary,description,
12,Improvement,Resolved,Fixed,   ,Major,Eric Yang,2011/09/26 23:42:56 +0100,2011/
46,Improvement,Resolved,Fixed,ambari-agent   ,Major,Eric Yang,2011/10/07 21:49:04
92,Bug,Resolved,Fixed,ambari-agent   ,Major,Eric Yang,2011/10/21 19:14:29 +0100,2
103,Improvement,Resolved,Fixed,ambari-server; ambari-web   ,Major,Eric Yang,2011/
192,Bug,Resolved,Fixed,   ,Major,Ramya Sunil,2012/05/08 23:24:58 +0100,2012/05/08
198,Bug,Resolved,Fixed,ambari-server   ,Major,Jitendra Nath Pandey,2012/05/09 21:
199,Bug,Resolved,Fixed,ambari-server   ,Major,Jitendra Nath Pandey,2012/05/09 22:
202,Bug,Resolved,Fixed,   ,Major,Ramya Sunil,2012/05/09 23:20:37 +0100,2012/05/09
207,Bug,Resolved,Fixed,ambari-server   ,Minor,Jitendra Nath Pandey,2012/05/10 04:
208,Bug,Resolved,Fixed,   ,Major,Hitesh Shah,2012/05/10 06:05:22 +0100,2012/05/10
217,Bug,Resolved,Fixed,   ,Major,vitthal (Suhas) Gogate,2012/05/10 22:41:47 +0100
222,Bug,Resolved,Fixed,   ,Major,vitthal (Suhas) Gogate,2012/05/11 01:10:46 +0100
226,Bug,Resolved,Fixed,   ,Minor,Suresh Srinivas,2012/05/11 18:29:32 +0100,null,U
232,Bug,Resolved,Fixed,   ,Major,Vikram Dixit K,2012/05/11 22:13:53 +0100,2012/05,
236,Bug,Resolved,Fixed,ambari-server   ,Major,Jitendra Nath Pandey,2012/05/12 18:
237,Bug,Resolved,Fixed,ambari-server   ,Major,Jitendra Nath Pandey,2012/05/12 18:
245,Bug,Resolved,Fixed,ambari-server   ,Major,Jitendra Nath Pandey,2012/05/15 00:
247,Bug,Resolved,Fixed,   ,Major,Varun Kapoor,2012/05/15 02:13:48 +0100,2012/05/1
249,Bug,Resolved,Fixed,   ,Major,Vikram Dixit K,2012/05/15 03:03:35 +0100,2012/05,
252,Bug,Resolved,Fixed,   ,Major,Varun Kapoor,2012/05/15 21:27:32 +0100,2012/05/1
253,Bug,Resolved,Fixed,   ,Major,Ramya Sunil,2012/05/15 21:32:31 +0100,2012/05/15
255,Bug,Resolved,Fixed,   ,Minor,Varun Kapoor,2012/05/15 22:46:46 +0100,2012/05/1
256,Bug,Resolved,Fixed,   ,Major,Ramya Sunil,2012/05/15 22:52:19 +0100,2012/05/15
257,Bug,Resolved,Fixed,site   ,Major,Arpit Gupta,2012/05/15 23:00:57 +0100,2012/0
262,Bug,Resolved,Fixed,site   ,Major,Arpit Gupta,2012/05/15 23:13:18 +0100,2012/0
```

Figure 3.3: Ohira's data set

Chromium bug reports are also in CSV file whose column name are id, date, reports and security. Each row of CSV files represents a bug report. CSV file for Chromium project is shown in figure 3.4. We only selected summary and description fields from Ohira’s bug reports for prediction of security bug reports. From chromium subset, we selected reports field for prediction. All the preprocessing is applied on these fields of bug reports.



```
id,date,report,security
2,30-Aug-08,Issue 2 : Testing if chromium id works 2 problem? 1. 2. 3. What is t
3,30-Aug-08,Issue 3 : This is a test 3 problem? 1. eat 2. sleep  ? 3. Wake up  ?
4,02-Sep-08,Issue 4 : Scrolling with some scroll mice (touchpad  etc.) scrolls d
5,02-Sep-08,Issue 5 : Java not working yet &nbsp;   &nbsp;   thijstel      : 0.2
6,02-Sep-08,Issue 6 : Chrome treats links in frames differently than Firefox doe
7,02-Sep-08,Issue 7 : Errors in importing from firefox 12 problem? 1. Installing
8,02-Sep-08,Issue 8 : Chromium fails to isntall 2 problem? Downloaded Chrome fro
9,02-Sep-08,Issue 9 : The rendering place and sidebar does not resize with other
10,02-Sep-08,Issue 10 : Clickable remember texts in forms &nbsp;   &nbsp;   thij
11,02-Sep-08,Issue 11 : Scrolling with middle-mouse button does not work (autosc
12,02-Sep-08,Issue 12 : Can't get Zimbra web mail into Advanced state when using
13,02-Sep-08,Issue 13 : Closing last tab also closes window 72 problem? 1. Have
14,02-Sep-08,Issue 14 : Proxy settings for installer 47 problem? Attempt to inst
15,02-Sep-08,Issue 15 : Chrome(ium?) is not fully Open Source 10 problem? 1. Dow
16,02-Sep-08,Issue 16 : Middle click doesn't initiate motion scroll 1 person sta
17,02-Sep-08,Issue 17 : input type=&quot;search&quot; does not act as expected i
18,02-Sep-08,Issue 18 : Wishlist: Chrome does not have an addon-system 890 probl
19,02-Sep-08,Issue 19 : Automatic integrated windows authentication (aka automat
20,02-Sep-08,Issue 20 : Chrome windows hide autohide taskbars 145 problem? 1. Ri
21,02-Sep-08,Issue 21 : Facebook: Commenting on Status not working 7 problem? 1.
22,02-Sep-08,Issue 22 : Active back button replaced by red square in Windows Vis
23,02-Sep-08,Issue 23 : Mouse wheel drag doesn't work 3 problem? 1. Open any web
24,02-Sep-08,Issue 24 : Proxy causes some or all network requests to fail 34 pro
```

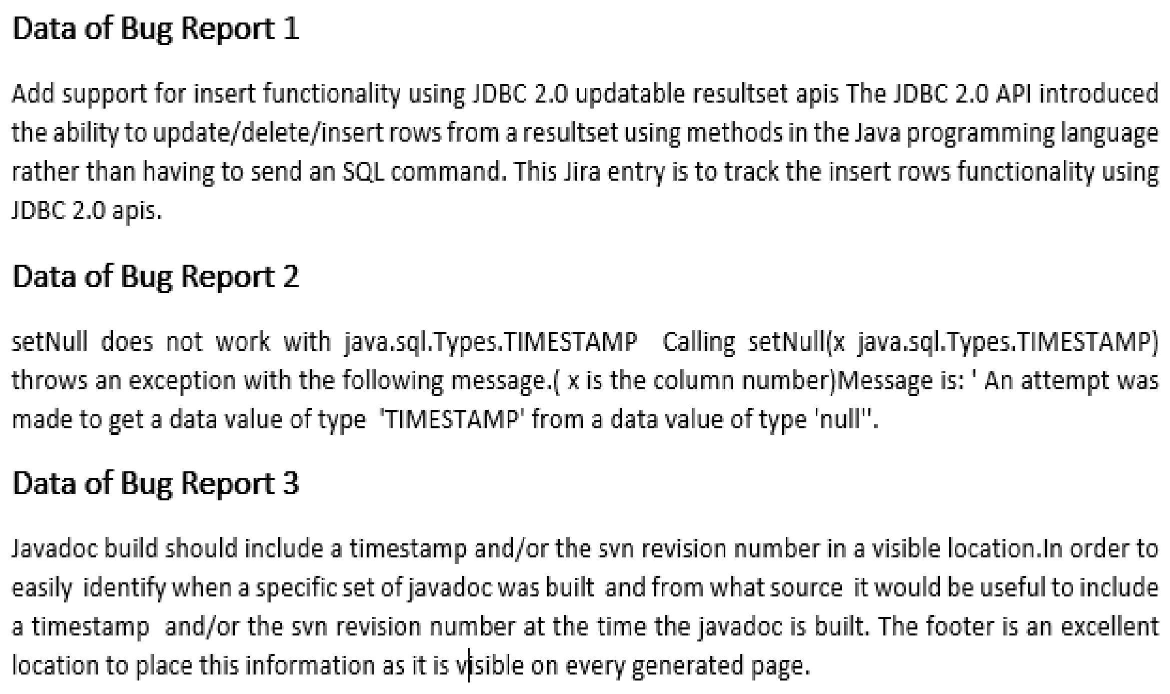
Figure 3.4: Chromium’s subset of data

3.3.2 Experimentation

For the implementation of our purposed approach, we used PyCharm as IDE. Python is used as a programming language for the implantation of our approach. It is simple and its sentences are easy to learn. We have used different libraries of Pythons for the implementation. NLTK is used for the preprocessing of bug reports, Sklearn is utilized for the implementation of ML algorithms and Imblearn is utilized for RUS of bug reports. We have run our approach on Core i5-5200U CPU @ 2.20GHz with RAM of 12GB.

There are two phases in our approach, one is training phase and other one is prediction phase. In training phase, labelled bug reports are utilized for the training of ML classifiers. In prediction phase, new or unlabeled bug reports are given to classifiers, these classifiers will predict these bug reports either SBRs or NSBRs. All the data of bug reports are in CSV files. We select only description and

summary fields for the Ohira's [75] four projects shown in figure 3.5. For chromium projects, only report filed is selected. For training phase, label from bug reports is also selected.



Data of Bug Report 1

Add support for insert functionality using JDBC 2.0 updatable resultset apis The JDBC 2.0 API introduced the ability to update/delete/insert rows from a resultset using methods in the Java programming language rather than having to send an SQL command. This Jira entry is to track the insert rows functionality using JDBC 2.0 apis.

Data of Bug Report 2

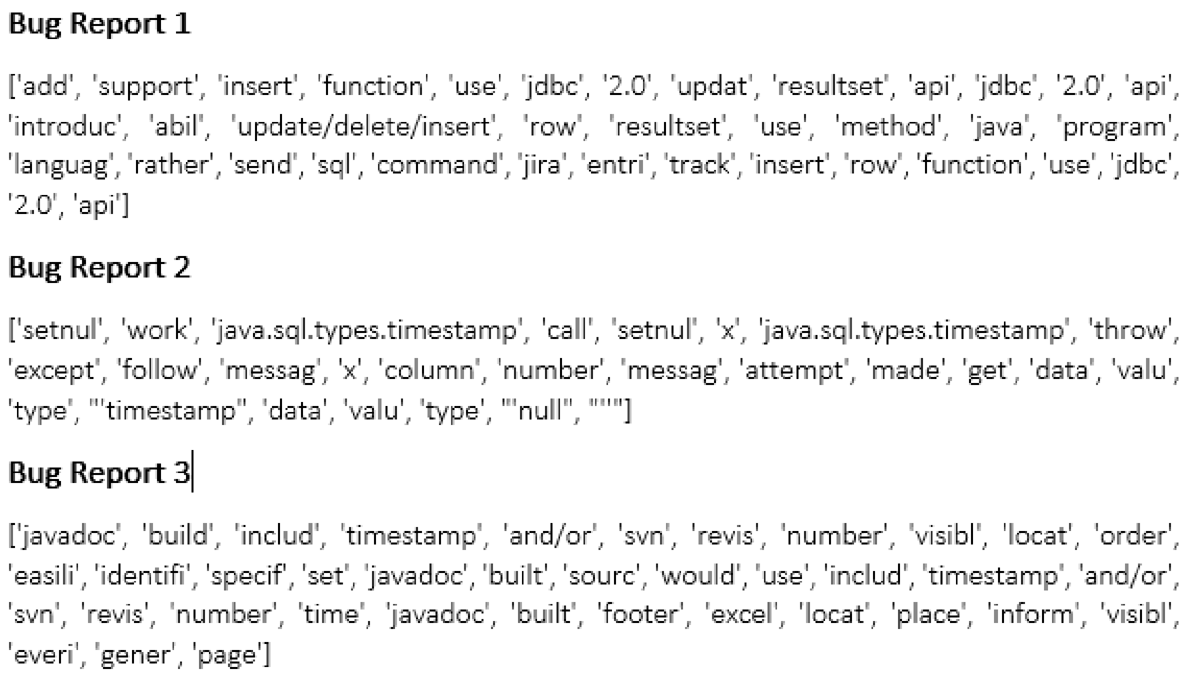
setNull does not work with java.sql.Types.TIMESTAMP Calling setNull(x java.sql.Types.TIMESTAMP) throws an exception with the following message.(x is the column number)Message is: ' An attempt was made to get a data value of type 'TIMESTAMP' from a data value of type 'null''.

Data of Bug Report 3

Javadoc build should include a timestamp and/or the svn revision number in a visible location.In order to easily identify when a specific set of javadoc was built and from what source it would be useful to include a timestamp and/or the svn revision number at the time the javadoc is built. The footer is an excellent location to place this information as it is visible on every generated page.

Figure 3.5: Textual data of bug reports

First step of our approach is preprocessing of bug reports. In preprocessing, we tokenized the summary and description fields of bug reports, convert them into lowercase, then removed stop
words, punctuations, unwanted terms and at the end stemming is applied. As a result, many terms for each report are extracted as displayed in figure 3.6.



Bug Report 1

```
['add', 'support', 'insert', 'function', 'use', 'jdbc', '2.0', 'updat', 'resultset', 'api', 'jdbc', '2.0', 'api',
'introduc', 'abil', 'update/delete/insert', 'row', 'resultset', 'use', 'method', 'java', 'program',
'languag', 'rather', 'send', 'sql', 'command', 'jira', 'entri', 'track', 'insert', 'row', 'function', 'use', 'jdbc',
'2.0', 'api']
```

Bug Report 2

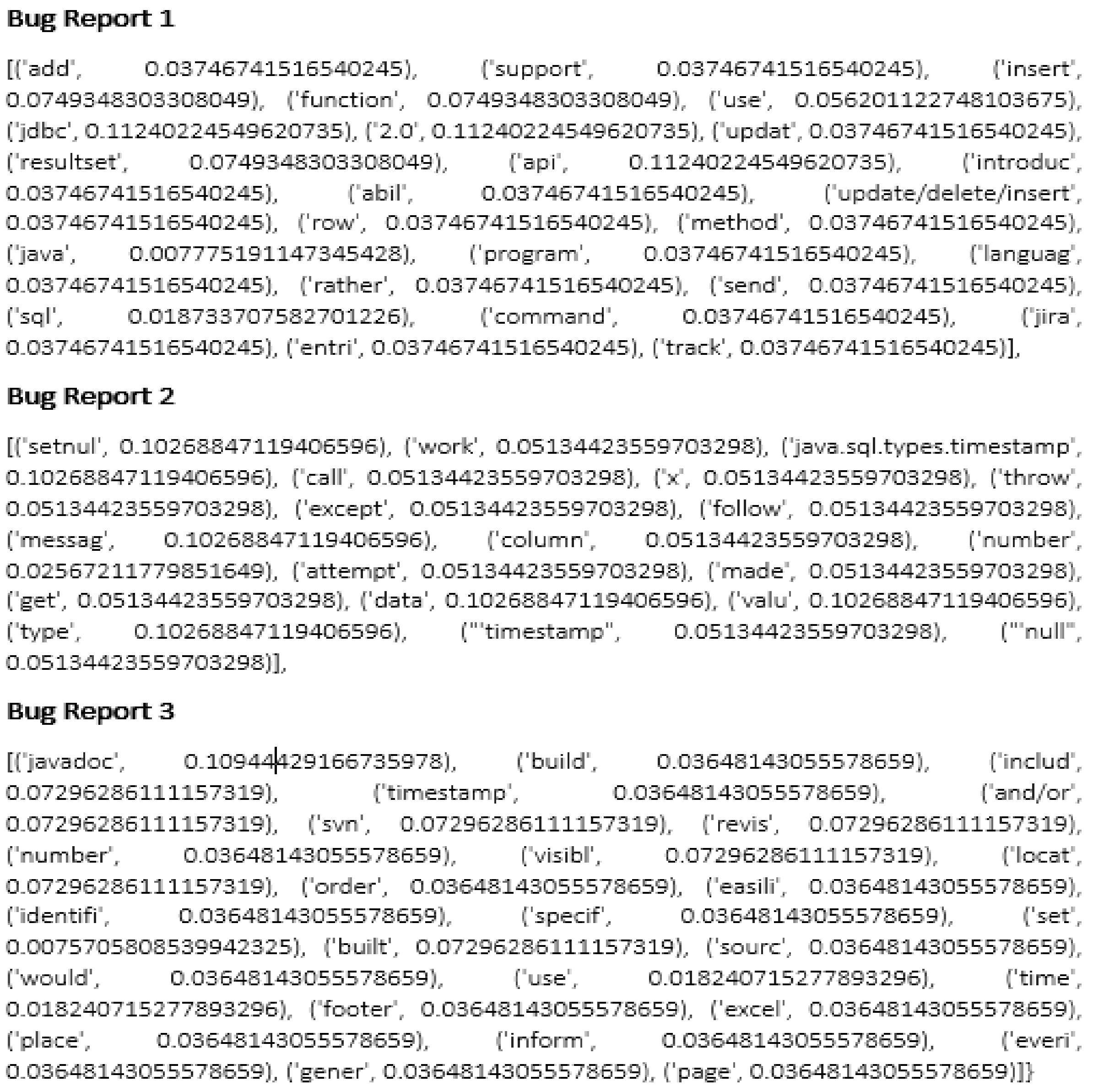
```
['setnul', 'work', 'java.sql.types.timestamp', 'call', 'setnul', 'x', 'java.sql.types.timestamp', 'throw',
'except', 'follow', 'messag', 'x', 'column', 'number', 'messag', 'attempt', 'made', 'get', 'data', 'valu',
'type', "'timestamp'", 'data', 'valu', 'type', "'null'", "''"]
```

Bug Report 3

```
['javadoc', 'build', 'includ', 'timestamp', 'and/or', 'svn', 'revis', 'number', 'visibl', 'locat', 'order',
'easili', 'identifi', 'specif', 'set', 'javadoc', 'built', 'sourc', 'would', 'use', 'includ', 'timestamp', 'and/or',
'svn', 'revis', 'number', 'time', 'javadoc', 'built', 'footer', 'excel', 'locat', 'place', 'inform', 'visibl',
'everi', 'gener', 'page']
```

Figure 3.6: Preprocessed Data

These extracted terms are considered as features. As these features are to many, we reduced it by applying TF-IDF. TF-IDF calculates the value of a term in a document. TF-IDF of terms are shown in figure 3.7.



Bug Report 1

```
[('add', 0.03746741516540245), ('support', 0.03746741516540245), ('insert', 0.0749348303308049), ('function', 0.0749348303308049), ('use', 0.05620112274810367 5), ('jdbc', 0.11240224549620735), ('2.0', 0.11240224549620735), ('updat', 0.03746741516540245), ('resultset', 0.0749348303308049), ('api', 0.11240224549620735), ('introduc', 0.03746741516540245), ('abil', 0.03746741516540245), ('update/delete/insert', 0.03746741516540245), ('row', 0.03746741516540245), ('method', 0.03746741516540245), ('java', 0.007775191147345428), ('program', 0.03746741516540245), ('languag', 0.03746741516540245), ('rather', 0.03746741516540245), ('send', 0.03746741516540245), ('sql', 0.018733707582701226), ('command', 0.03746741516540245), ('jira', 0.03746741516540245), ('entri', 0.03746741516540245), ('track', 0.03746741516540245)],
```

Bug Report 2

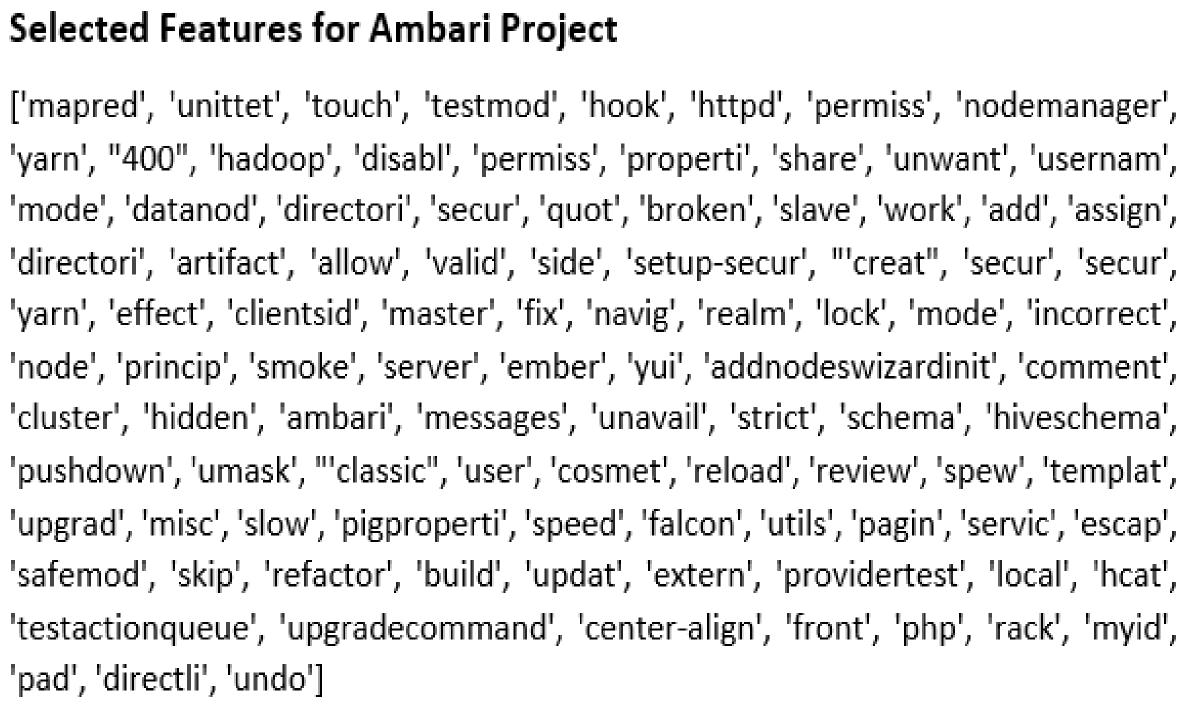
```
[('setnul', 0.10268847119406596), ('work', 0.05134423559703298), ('java.sql.types.timestamp', 0.10268847119406596), ('call', 0.05134423559703298), ('x', 0.05134423559703298), ('throw', 0.05134423559703298), ('except', 0.05134423559703298), ('follow', 0.05134423559703298), ('messag', 0.10268847119406596), ('column', 0.05134423559703298), ('number', 0.025672117798516 49), ('attempt', 0.05134423559703298), ('made', 0.05134423559703298), ('get', 0.05134423559703298), ('data', 0.10268847119406596), ('valu', 0.10268847119406596), ('type', 0.10268847119406596), ("'timestamp'", 0.05134423559703298), ("'null'", 0.05134423559703298)],
```

Bug Report 3

```
[('javadoc', 0.10944429166735978), ('build', 0.03648143055578659), ('includ', 0.07296286111157319), ('timestamp', 0.03648143055578659), ('and/or', 0.07296286111157319), ('svn', 0.07296286111157319), ('revis', 0.07296286111157319), ('number', 0.03648143055578659), ('visibl', 0.07296286111157319), ('locat', 0.07296286111157319), ('order', 0.03648143055578659), ('easili', 0.03648143055578659), ('identifi', 0.03648143055578659), ('specif', 0.03648143055578659), ('set', 0.0075705808539942325), ('built', 0.07296286111157319), ('sourc', 0.03648143055578659), ('would', 0.03648143055578659), ('use', 0.01824071527789329 6), ('time', 0.018240715277893296), ('footer', 0.03648143055578659), ('excel', 0.03648143055578659), ('place', 0.03648143055578659), ('inform', 0.03648143055578659), ('everi', 0.03648143055578659), ('gener', 0.03648143055578659), ('page', 0.03648143055578659)]}
```

Figure 3.7: TF-IDF Values

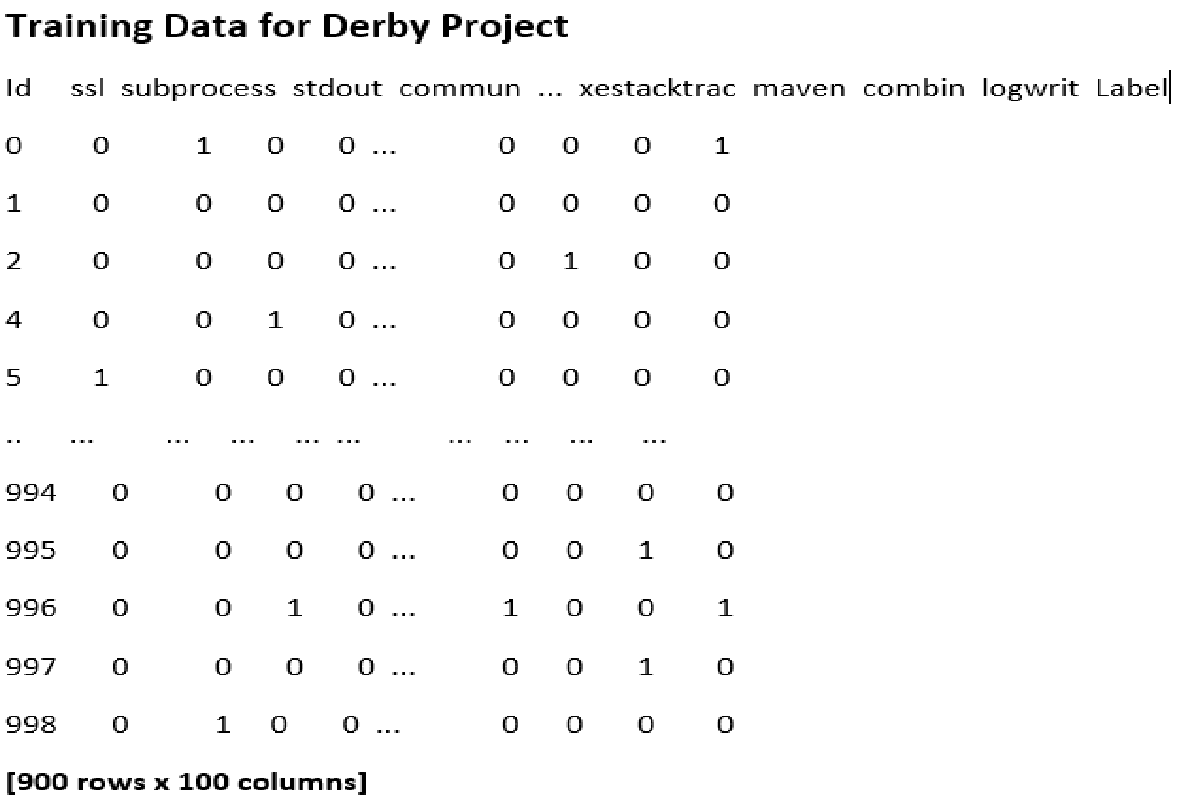
We have selected Top 50 terms according to top TF-IDF from each bug reports category (SBRs and NSBRs) for each project as feature sets. The 100 selected features of Derby project shown in figure 3.8


Selected Features for Ambari Project

['mapred', 'unittet', 'touch', 'testmod', 'hook', 'httpd', 'permiss', 'nodemanager', 'yarn', "400", 'hadoop', 'disabl', 'permiss', 'properti', 'share', 'unwant', 'usernam', 'mode', 'datanod', 'directori', 'secur', 'quot', 'broken', 'slave', 'work', 'add', 'assign', 'directori', 'artifact', 'allow', 'valid', 'side', 'setup-secur', "'creat'", 'secur', 'secur', 'yarn', 'effect', 'clientsid', 'master', 'fix', 'navig', 'realm', 'lock', 'mode', 'incorrect', 'node', 'princip', 'smoke', 'server', 'ember', 'yui', 'addnodeswizardinit', 'comment', 'cluster', 'hidden', 'ambari', 'messages', 'unavail', 'strict', 'schema', 'hiveschema', 'pushdown', 'umask', "'classic'", 'user', 'cosmet', 'reload', 'review', 'spew', 'templat', 'upgrad', 'misc', 'slow', 'pigproperti', 'speed', 'falcon', 'utils', 'pagin', 'servic', 'escap', 'safemod', 'skip', 'refactor', 'build', 'updat', 'extern', 'providertest', 'local', 'hcat', 'testactionqueue', 'upgradecommand', 'center-align', 'front', 'php', 'rack', 'myid', 'pad', 'directli', 'undo']

Figure 3.8: Selected features for Ambari project

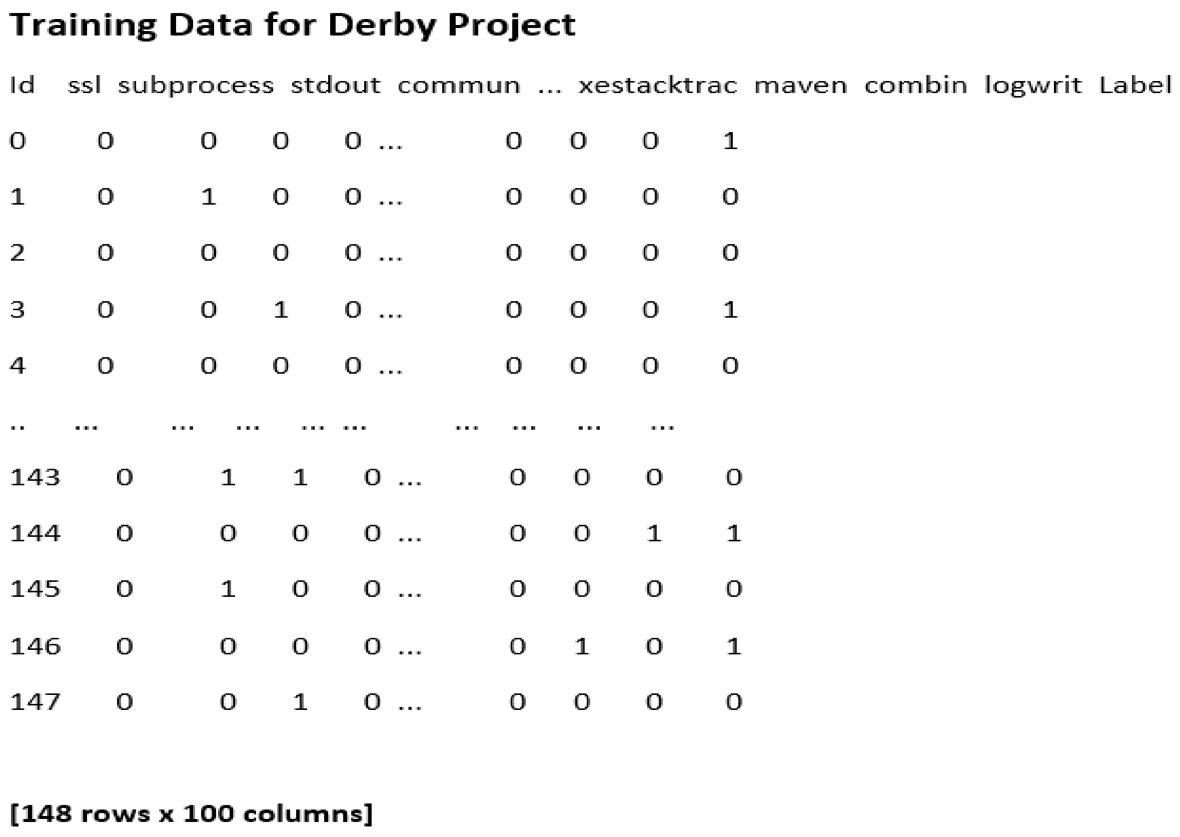
After feature selection, we have applied a sampling technique named random under sampling to reduce the majority class for better performance of our purposed approach. Before random under sampling and after random under sampling training data shown in figure 3.9 and 3.10 respectively. Before random sampling, training data has 900 samples.


Training Data for Derby Project

```
Id   ssl  subprocess  stdout  commun  ...  xestacktrac  maven  combin  logwrit  Label
0     0       1      0     0 ...        0     0     0     1
1     0       0      0     0 ...        0     0     0     0
2     0       0      0     0 ...        0     1     0     0
4     0       0      1     0 ...        0     0     0     0
5     1       0      0     0 ...        0     0     0     0
..   ...     ...    ...   ... ...      ...   ...   ...   ...
994    0       0     0     0 ...       0     0     0     0
995    0       0     0     0 ...       0     0     1     0
996    0       0     1     0 ...       1     0     0     1
997    0       0     0     0 ...       0     0     1     0
998    0       1    0     0 ...        0     0     0     0
[900 rows x 100 columns]
```

Figure 3.9: Samples before Random Under Sampling

After applying random under sampling technique, it reduces to 148 samples as it selects and delete the samples belonging to majority class and equals to minority class.

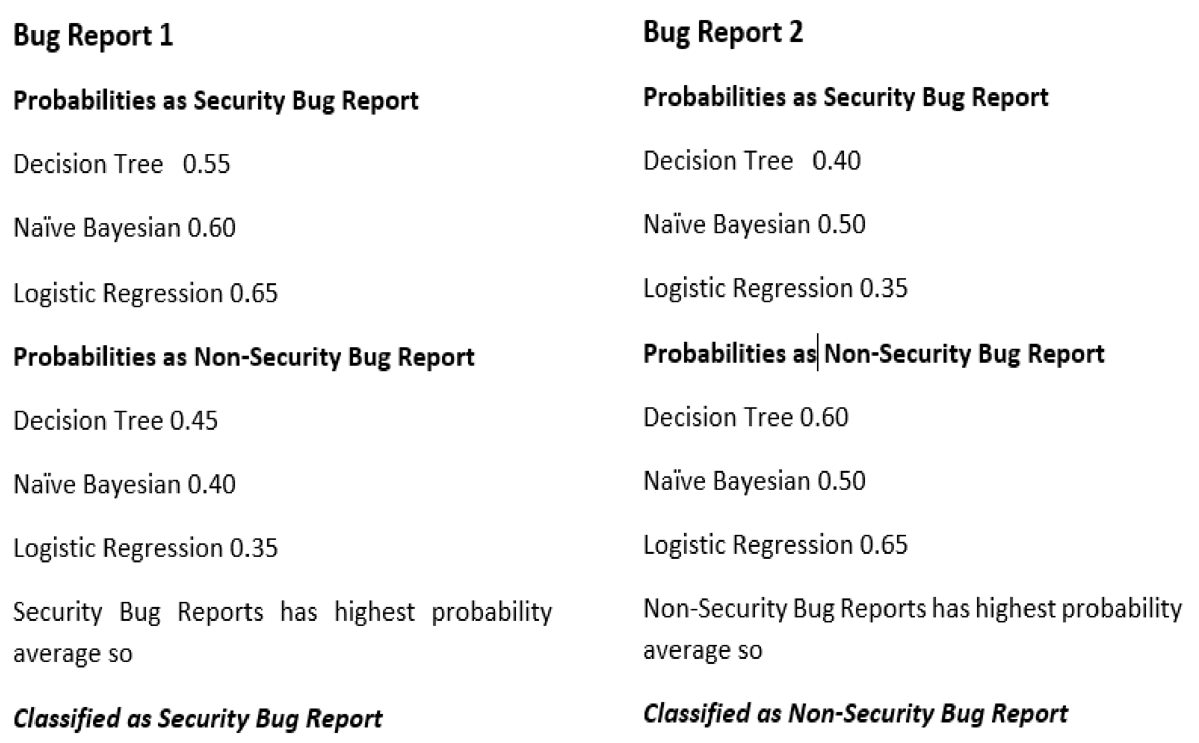


```
Training Data for Derby Project
Id   ssl  subprocess  stdout  commun  ...  xestacktrac  maven  combin  logwrit  Label
0      0         0      0      0 ...          0     0      0      1
1      0         1      0      0 ...          0     0      0      0
2      0         0      0      0 ...          0     0      0      0
3      0         0      1      0 ...          0     0      0      1
4      0         0      0      0 ...          0     0      0      0
..   ...       ...    ...    ... ...     ...   ...    ...    ...
143    0         1      1      0 ...        0     0      0      0
144    0         0      0      0 ...        0     0      1      1
145    0         1      0      0 ...        0     0      0      0
146    0         0      0      0 ...        0     1      0      1
147    0         0      1      0 ...        0     0      0      0

[148 rows x 100 columns]
```

Figure 3.10: Samples after Random Under Sampling

We have used three ML algorithms which are decision Tree, Logistic Regression and Naïve Bayes in our purposed approach. Voting is used as ensemble to get better result. These ML models are trained and tested on data sets for each project. Then voting is applied on the results of these three machine learning algorithms to get better results shown in figure 3.11.


Bug Report 1

Probabilities as Security Bug Report

Decision Tree 0.55

Naïve Bayesian 0.60

Logistic Regression 0.65

Probabilities as Non-Security Bug Report

Decision Tree 0.45

Naïve Bayesian 0.40

Logistic Regression 0.35

Security Bug Reports has highest probability average so

Classified as Security Bug Report

Bug Report 2

Probabilities as Security Bug Report

Decision Tree 0.40

Naïve Bayesian 0.50

Logistic Regression 0.35

Probabilities as Non-Security Bug Report

Decision Tree 0.60

Naïve Bayesian 0.50

Logistic Regression 0.65

Non-Security Bug Reports has highest probability average so

Classified as Non-Security Bug Report

Figure 3.11: Classifier

At the end, for the validation of our approach, K-Fold cross validation is applied. Our approach is evaluated through confusion matrix. Performance measures used for our approach are Precision, PD, PF, F-measure and G-measure.

Chapter 4

Results and Evaluation

CHAPTER 4: RESULTS AND EVALUATION

This chapter deals with the results and evaluation of purposed approach. **Section 5.1** discussed the evaluation metrics to be used for the evaluation of our approach. **Section 5.2** discussed the results and comparison with the previously used techniques.

4.1 Evaluation Metric

For the evaluation of our purposed approach, we used precision, probability of detection or recall, F-measure and G-measure as evaluation matrix's. These are commonly used metrics for performance evaluation which are derived from confusion metric. These concepts are explained below.

4.1.1 Confusion Metrix

Confusion matrix is a table which is commonly utilized to measure the efficiency of a classification model. It is mostly used for measuring precision, recall, F-measure, accuracy. It grants easy recognition of confusion between the classes for example 1 class is mostly misclassified as the other. It's an overview of the predicted outcomes on a problem of classification. The true and false prediction's numbers are overviewed with the numeric values. It shows the means in which a classification model is confused during prediction. A confusion matric displayed in table 4.1.

Table 4.1: Confusion Matric

		Predicted Values	
		SBRs	**Non-SBRs**
Actual Values	**SBRs**	TP	FN
	Non-SBRs	FP	TN

Definition of Terms

- **Positive (P)**: Perception is true (It's a SBR).
- **Negative (N)**: Perception is not true (It's a Non-SBR).
- **True Positive (TP)**: W have predicted it's a SBR and actually it's a SBR
- **False Negative (FN)**: We have predicted it's a SBR and actually it's a NSBR
- **True Negative (TN)**: We have predicted it is a NSBR and actually it is a NSBR
- **False Positive (FP)**: We have predicted it is a NSBR and actually it is a SBR.

4.1.2 Precision

It is defined as out of all the positive classes we forecasted correctly, how many are actually positive. Its value should be high.

Precision = TP/(TP+FP)

4.1.3 Probability of Detection

It is defined as out of all the positive classes, how much we forecasted correctly. Its value should be high as possible. It is also called recall.

PD = TP/(TP+FN)

4.1.4 Probability of False Alarm

It calculates the fraction of the Non-SBRs which are inaccurately forecasted as SBRs.

PF = FP/(FP+TN)

4.1.5 F-measure

It is very problematic to compare two models which have precision's value high and have recall's value low or vice versa. So, in order to make them comparable, we utilize F-measure. F-measure helps to calculate recall and precision at same time. Harmonic Mean is utilized instead of arithmetic mean by grueling the higher values more.

F-measure = (2*Precision*PD)/(Precision + PD)

4.1.6 G-measure

The g-measure is the defined as "the harmonic-mean of-PD and-(100-PF)"[76]. PF is the probability of false alarm.

G-measure = (2*PD*(100-PF))/(PD+(100-PF))

4.1.7 K Fold Cross Validation

Cross-validation is a shuffling process utilized for the evaluation of machine learning models on a data sample. This process has only one parameter which is K that indicates the number of bunches for a given data is divide into. This process is commonly called k-fold-cross-validation.

It is mostly used in ML to measure the accomplishment of a ML model on unseen data. it is very easy understand. It is a popular method because it is less biased model than other methods.

We set the value of K = 10. The idea behind 10-fold cross-validation is that whole data is divided randomly into ten subsets. One subset is used as testing data and nine subsets are used for training of model from these ten subsets of data. These steps are repeated ten times, with all of the subsets are used as testing data for once in order to evaluate the performance of our approach. The end result is the average of all ten iterations.

4.2 Results Evaluation and Comparison

After the implementation of our purposed approach, we evaluated our experimentation by using Precision, PD, PF, F-measure and G-measure. For these evaluation matrixes, we used confusion matrix. TN, TP, FN and FP values are utilized for the evaluation from confusion matrix. For the validation of data, we used K-Fold cross validation method. We set the value of K = 10.

The python code and output for results are given below

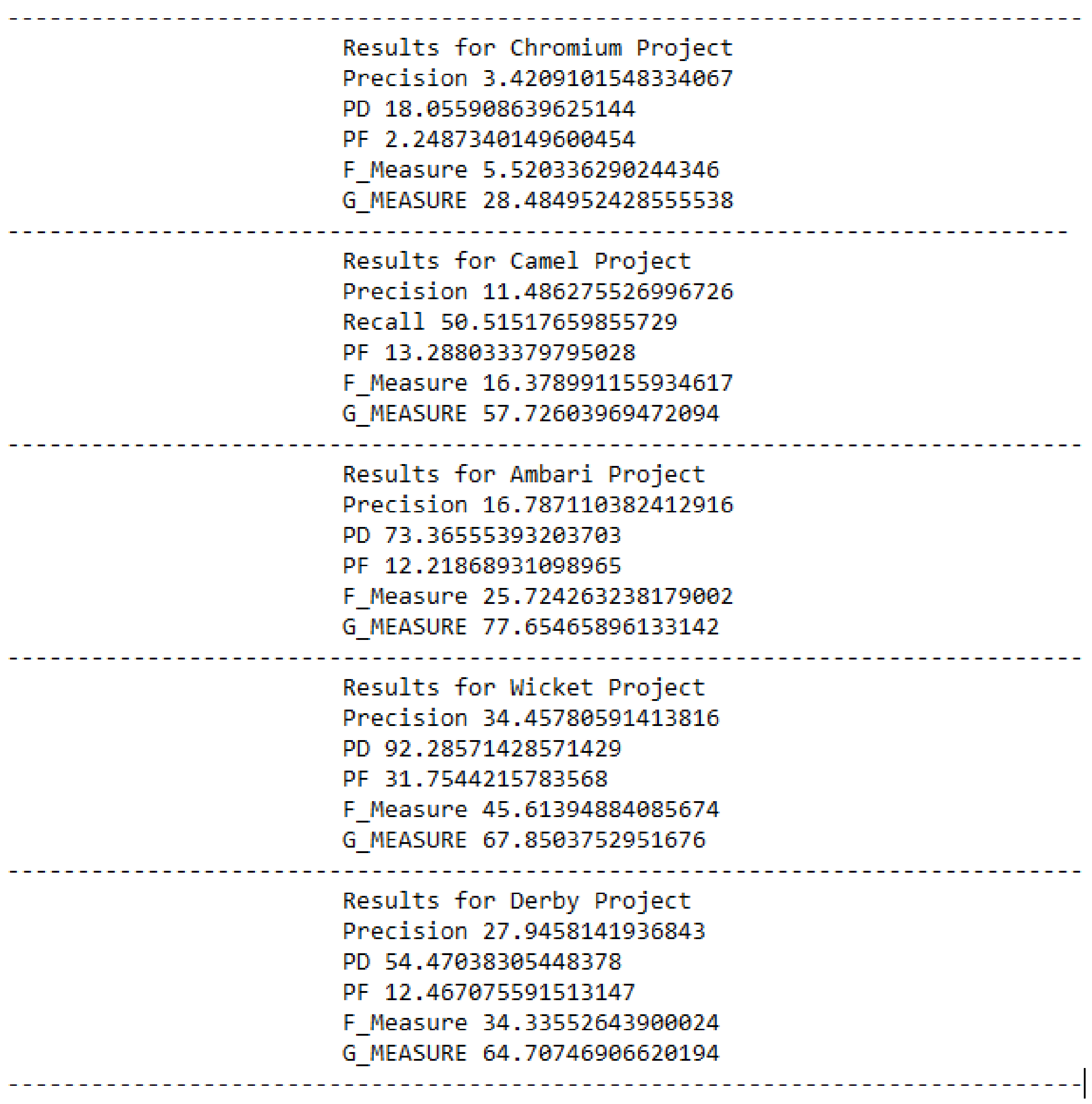
Python Code

```
“Import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from imblearn.under_sampling import RandomUnderSampler
df = pd.read_csv('Features.csv')
target = 'label'
G_measure = 0
Recall = 0
Precision = 0
F_measure = 0
X = df.loc[:, df.columns!=target]
Y = df.loc[:, df.columns==target]
skf = KFold(n_splits=10, random_state=0,shuffle=True)
skf.get_n_splits(X,Y)
nr = RandomUnderSampler(random_state=0)
G_measure = 0
Recall = 0
Precision = 0
F_measure = 0
PFF = 0
for train_index, test_index in skf.split(X, Y):
   #print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X.loc[train_index], X.loc[test_index]
```

```
    y_train, y_test = Y.loc[train_index], Y.loc[test_index]
    X_train_S, y_train_S = nr.fit_sample(X_train, y_train)
   clf1 = LogisticRegression()
    clf2 = DecisionTreeClassifier()
    clf3 = GaussianNB()
    evc = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2), ('nb', clf3)], voting='soft')
    result = evc.fit(X_train_S, np.ravel(y_train_S))
    Y_Test_Pred = result.predict(X_test)
    tn, fp, fn, tp = confusion_matrix(y_test, Y_Test_Pred).ravel()

    PD = (tp / (tp + fn)) * 100
    Recall += PD
    PF = (fp / (fp + tn)) * 100
    PFF += PF
    PREC = (tp / (tp + fp)) * 100
    Precision += PREC
    f_measure = (2 * PD * PREC) / (PD + PREC)
    F_measure += f_measure
    g_measure = (2 * PD * (100 - PF)) / (PD + (100 - PF))
    G_measure += g_measure
print('Precision', Precision/10)
print('PD', Recall/10)
print('PF', PFF/10)
print('F_Measure', F_measure/10)
print('G_MEASURE', G_measure/10)”
```

Results are shown in figure 4.1



```
---------------------------------------------------------------------------
                        Results for Chromium Project
                        Precision 3.4209101548334067
                        PD 18.055908639625144
                        PF 2.2487340149600454
                        F_Measure 5.520336290244346
                        G_MEASURE 28.484952428555538
--------------------------------------------------------------------------
                        Results for Camel Project
                        Precision 11.486275526996726
                        Recall 50.51517659855729
                        PF 13.288033379795028
                        F_Measure 16.378991155934617
                        G_MEASURE 57.72603969472094
---------------------------------------------------------------------------
                        Results for Ambari Project
                        Precision 16.787110382412916
                        PD 73.36555393203703
                        PF 12.21868931098965
                        F_Measure 25.724263238179002
                        G_MEASURE 77.65465896133142
---------------------------------------------------------------------------
                        Results for Wicket Project
                        Precision 34.45780591413816
                        PD 92.28571428571429
                        PF 31.7544215783568
                        F_Measure 45.61394884085674
                        G_MEASURE 67.8503752951676
---------------------------------------------------------------------------
                        Results for Derby Project
                        Precision 27.9458141936843
                        PD 54.47038305448378
                        PF 12.467075591513147
                        F_Measure 34.33552643900024
                        G_MEASURE 64.70746906620194
---------------------------------------------------------------------------
```

Figure 4.1: Results for each project

The average values of Precision, PD, PF, F-measure and G measure for each project are given in table 4.2.

Table 4.2: Results for each project

Project	Precision	PD	PF	F-measure	G-measure
Camel	11.54 %	50.51 %	13.28 %	16.42 %	57.71 %
Ambari	16.0 %	73.36 %	12.72 %	25.48 %	77.31 %
Wicket	34.45 %	92.28 %	31.75 %	45.61 %	67.85 %
Derby	27.90 %	54.47 %	12.47 %	34.34 %	64.70 %
Chromium	3.42 %	18.055 %	2.24 %	5.52 %	28.48 %

These results are also shown in figure 4.2

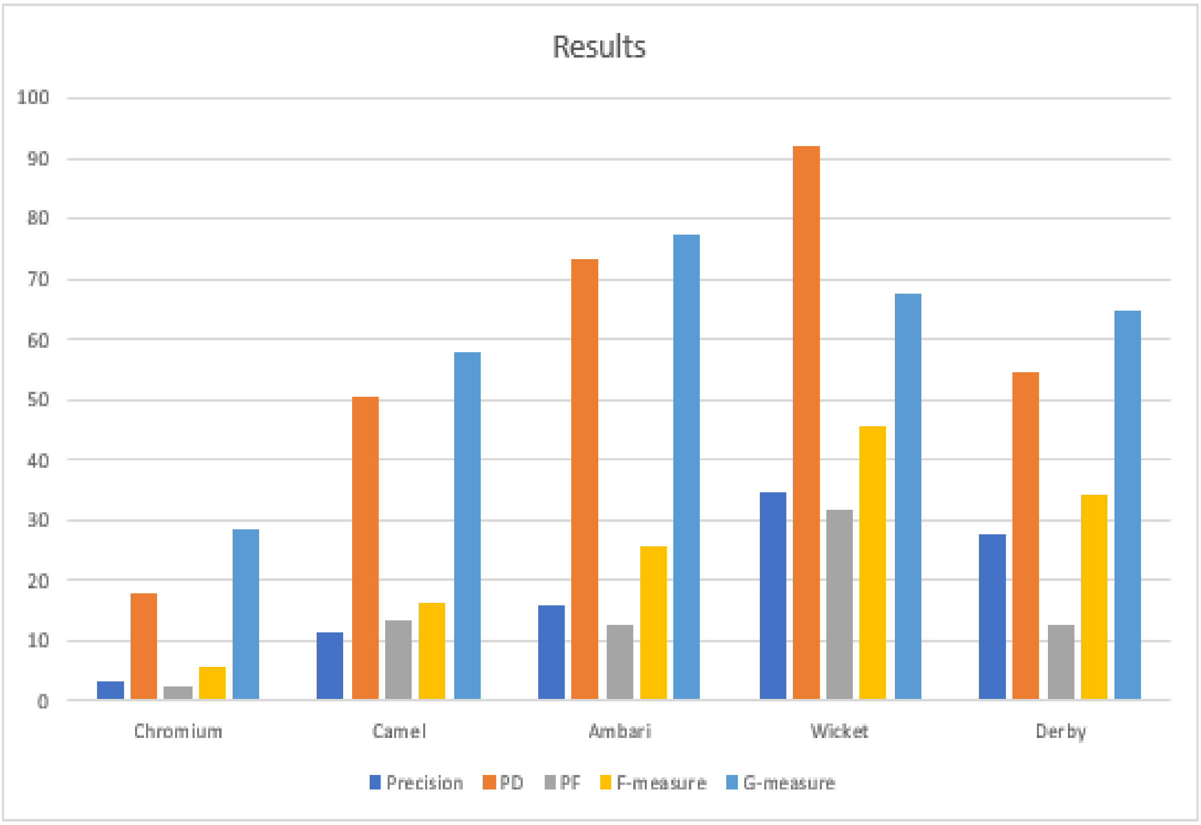


Figure 4.2: Chart for results

We have performed comparison of our approach with the FARSEC [9] approach. They [9] purposed a filtering and ranking method for prediction of security bug reports [FARSEC]. They also used two filters, one is FARSECSQ in which applying support function to the word's frequency found in SBRs and in FARSECTWO multiplying the frequency by two. We compared results of our purposed approach with FARSEC, FARSECSQ and FARSECTWO's results. Our purposed approached better than these three approaches according to G-measure in all five projects. Comparison is shown in table 4.3.

Table 4.3: Comparison of results

Project	**Approach**	**Precision**	**PD**	**PF**	**F-measure**	**G-measure**
Camel	**ALSBR**	**11.54 %**	**50.51 %**	**13.28 %**	**16.42 %**	**57.71 %**
	FARSEC	8.3 %	16.7 %	6.9 %	11.1 %	28.3 %
	FARSECSQ	5.2 %	16.7 %	11.4 %	7.9 %	28.1 %
	FARSECTWO	**4.3 %**	**50 %**	**41.8 %**	**7.9 %**	**53.8 %**
Ambari	**ALSBR**	**16.0 %**	**73.36 %**	**12.72**	**25.48 %**	**77.31 %**
	FARSEC	4.0 %	14.3 %	4.9 %	6.3 %	24.8 %
	FARSECSQ	4.1 %	42.9 %	14.4%	7.4 %	57.1 %
	FARSECTWO	**21.1 %**	**57.1 %**	**3.0 %**	**30.8 %**	**71.9 %**
Wicket	**ALSBR**	**34.45 %**	**92.28 %**	**31.75 %**	**45.61 %**	**67.85 %**
	FARSEC	4.8 %	33.3 %	8.1 %	8.3 %	48.9 %
	FARSECSQ	2.1 %	66.7 %	38.3 %	4.0 %	64.1 %
	FARSECTWO	**2.2 %**	**66.7 %**	**36.6 %**	**4.2 %**	**65.0 %**
Derby	**ALSBR**	**27.90 %**	**54.47 %**	**12.47 %**	**34.34 %**	**64.70 %**
	FARSEC	35.6 %	38.1 %	6.3 %	36.8 %	54.2 %
	FARSECSQ	14.4 %	54.8 %	29.9 %	22.8 %	61.5 %
	FARSECTWO	**26.0 %**	**47.6 %**	**12.4 %**	**33.6 %**	**61.7 %**
Chromium	**ALSBR**	**3.42 %**	**18.055%**	**2.24 %**	**5.52 %**	**28.48 %**
	FARSEC	**31.0 %**	**15.7 %**	**0.2 %**	**20.8 %**	**27.1 %**
	FARSECSQ	23.9 %	14.8 %	0.3 %	18.3 %	25.7 %
	FARSECTWO	31.0 %	15.7 %	0.2 %	20.8 %	27.1 %

Comparison of ALSBR, FARSEC, FARSECSQ and FARSECTWO with respect to G-measure shown graphically in figure 4.3 and 4.4.

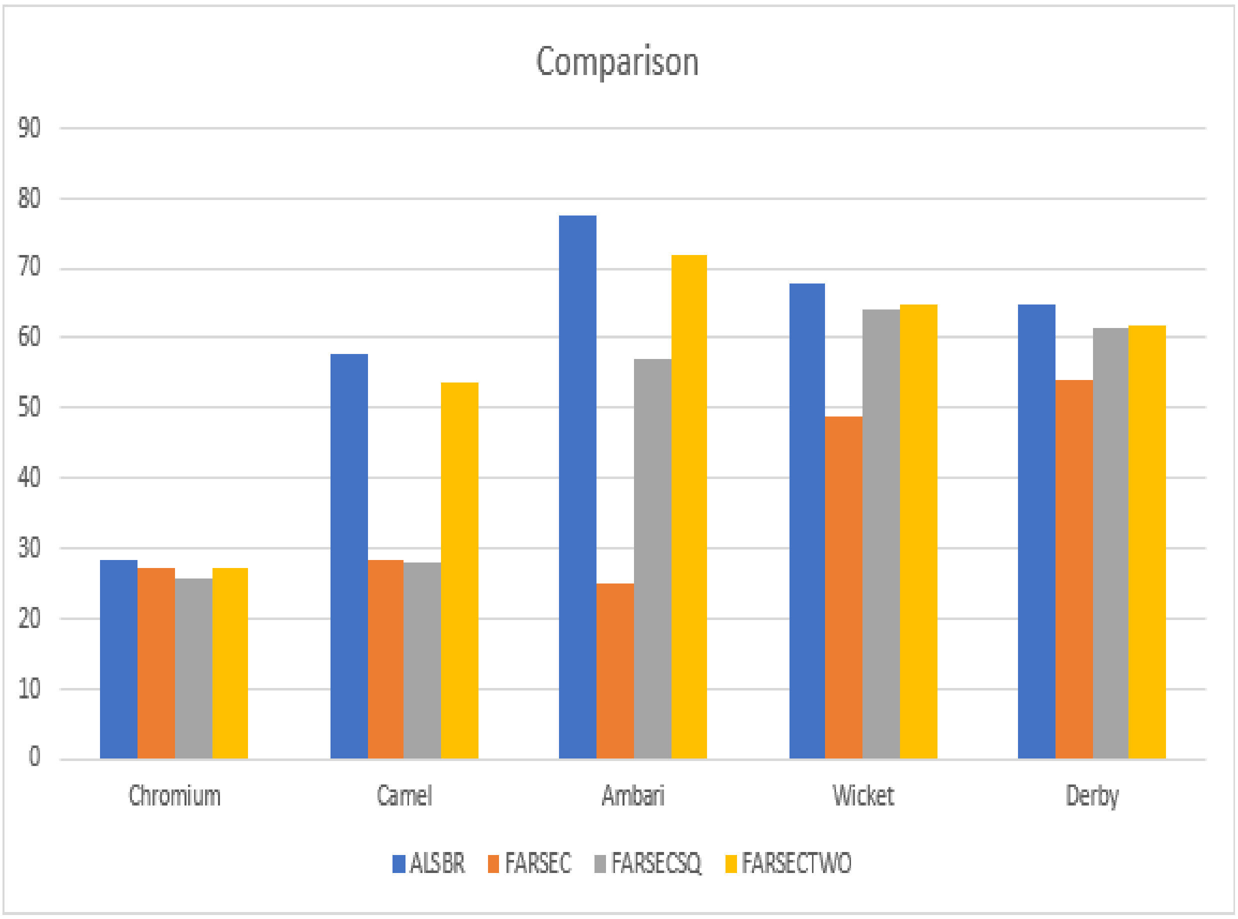


Figure 4.3: Comparison 1 of results with respect to G-measure

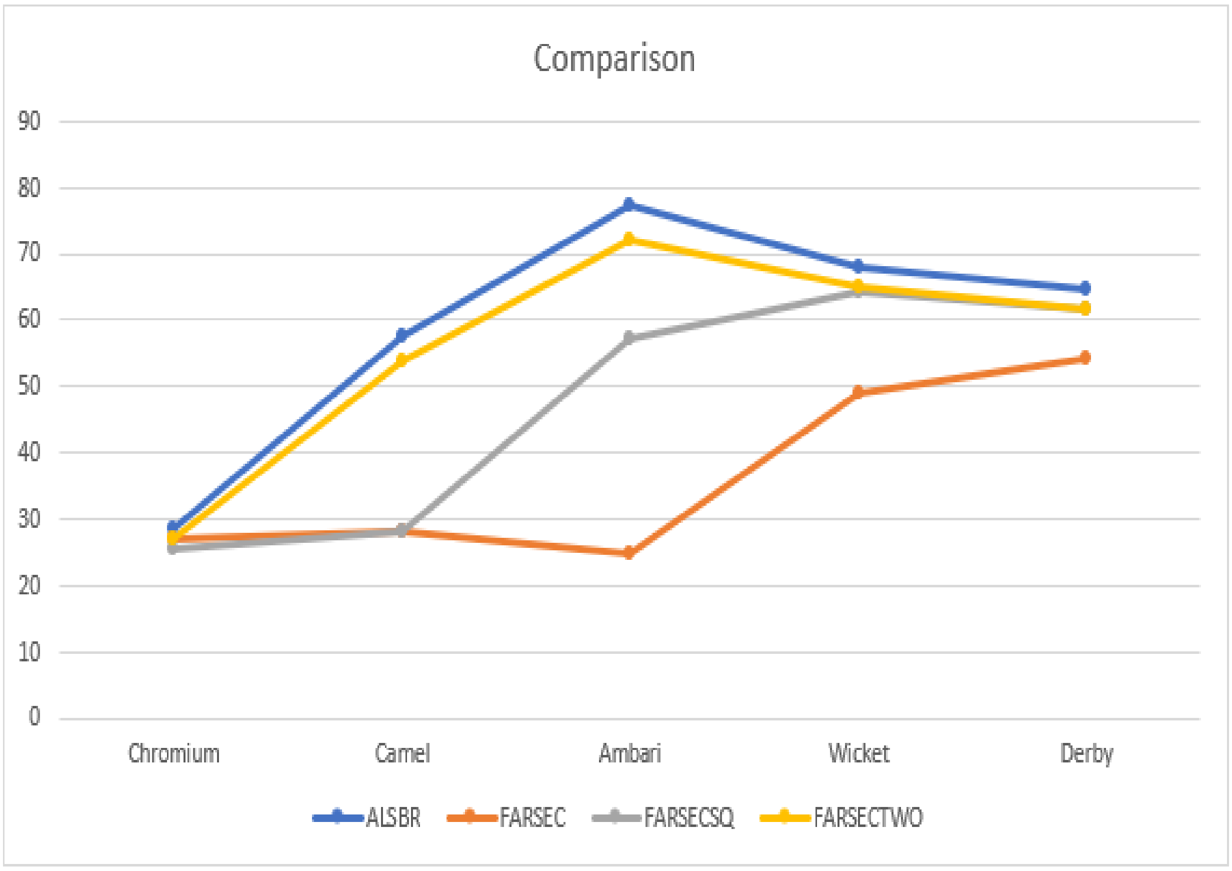


Figure 4.4: Comparison 2 of results with respect to G-measure

Chapter 5

Discussion & Limitations

CHAPTER 5: DISCUSSION & LIMITATIONS

This chapter presents discussion in **Section 6.1.** Limitations to our work is shown in section **6.2.**

5.1 Discussion

Automation has changed the today processes by introducing a great accuracy and reducing the time delays. NLP proved helpful in software development processes and it helped the software developers in many fields. On such field is the resolving the bugs found in many different software. In all kinds of bugs, security bugs are most important because these bugs are great threat to the privacy and security of end users. Therefore, these bugs are needed to be fixed as early as possible.

Therefore, we purposed an approach for the automatic identifications of security bug reports. We compared purposed approach with FARSEC model that is the state of the art technique and achieved improved results in terms of Precision, PD, PF, F-measure and G-measure.

5.2 Limitations

As our technique is a unique step towards the automatic labeling of security bug reports but few limitations are also present in our work. The accuracy of Python's libraries such as NLTK, Sklearn and Imblearn is questionable as it is not 100%.

Chapter 6

Conclusion and Future Work

CHAPTER 6: CONCLUSION AND FUTURE WORK

This chapter includes research conclusion in **Section 7.1** and future work is described in **Section 7.2.**

6.1 Conclusion

This research shows a unique technique for the automatic labelling of security bug reports by using NLP's techniques and ML algorithms. The following milestones are achieved from this research.

At first, a detailed SLR of most related research is conducted. For SLR, first we defined some research questions mentioned in chapter 2. Then done a detailed SLR of 54 literatures which were selected conferring selection and rejection criterion. All the chosen studies were categorized into five categories conferring to the approaches used. Then we have found thirteen (13) tools that are purposed or developed by different researchers. We have also found different NLP techniques used by different researchers and categorized all selected studies according to these NLP techniques. We have also identified the twenty-four libraries/technologies used for the labeling/classifications of bug reports by different literatures. Then we have performed a comparison of identified or developed tools. After performing this analysis, we have answered all five questions.

After conducting SLR, we purposed a novel approach for the automatic labelling of security bug reports by using NLP's techniques and ML algorithms mentioned in chapter 3. Our approach named ALSBR is implemented in Python using Natural language Toolkit (NLTK), Sklearn and Imblearn.

In our purposed methodology, first of all preprocessing of bug reports is performed. After the preprocessing, features are selected by TF-IDF values. Top 100 terms according to TF-IDF values are selected as features. After feature selections, a random under sampling technique is applied to balance the majority and minority classes. Three ML algorithms named Logistic Regression, Decision Tree and Naïve Bayes is used as classification model. A voting strategy is also applied to get the more accuracy. For the validation of our approach, 10-Fold cross validation is applied.

For the validation of our purposed technique we used bug reports of five projects. Among these 5 projects, four are from Ohira et al. [75] and a part of bug reports is selected from Chromium project. At the end, we compared our approach with state-of-the-art approach named FARSEC model and achieved improved results in terms of Precision, PD, PF, F-measure and G-measure.

6.2 Future Work

Future work includes improving and extending this approach in order to support the bug resolution in better way. It includes followings

- In future we can add more information presents in bug reports like who were assigned reports and time duration presenting how much time is spent on fixing a bug.
- Feedback and comments on bug reports can also include on future.
- A proper GUI tool can be generated and provided for the public access to use this tool.
- In future, our work can be extend for the labelling of other types of bugs such as performance, usability etc.

References

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

[2] Bettenburg, Nicolas, et al. "What makes a good bug report?." *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008.

[3] Xuan, Jifeng, et al. "Towards effective bug triage with software data reduction techniques." *IEEE transactions on knowledge and data engineering* 27.1 (2014): 264-280.

[4] Pressman, Roger S. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[5] Gegick, Michael, Pete Rotella, and Tao Xie. "Identifying security bug reports via text mining: An industrial case study." *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010.

[6] Kitchenham, Barbara, et al. "Systematic literature reviews in software engineering–a systematic literature review." *Information and software technology* 51.1 (2009): 7-15.

[7] Behl, Diksha, Sahil Handa, and Anuja Arora. "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf." *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014.

[8] Das, Dipok Chandra, and Md Rayhanur Rahman. "Security and Performance Bug Reports Identification with Class-Imbalance Sampling and Feature Selection." *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2018.

[9] Peters, Fayola, et al. "Text filtering and ranking for security bug report prediction." *IEEE Transactions on Software Engineering* (2017).

[10] Pereira, Mayana, Alok Kumar, and Scott Cristiansen. "Identifying Security Bug Reports Based Solely on Report Titles and Noisy Data." *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019.

[11] Goseva-Popstojanova, Katerina, and Jacob Tyo. "Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification." *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018.

[12] Peeples, Cody R., Pete Rotella, and Mark-David McLaughlin. "Textual analysis of security bug reports." *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 2017.

[13] Du, Tie, et al. "Automatic Security Bug Classification: A Compile-Time Approach." *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016.

[14] Chawla, Indu, and Sandeep K. Singh. "Automatic bug labeling using semantic information from LSI." *2014 Seventh International Conference on Contemporary Computing (IC3)*. IEEE, 2014.

[15] Rawal, Bharat S., and Anthony K. Tsetse. "Analysis of bugs in Google security research project database." *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. IEEE, 2015.

[16] Alves, Henrique, Baldoino Fonseca, and Nuno Antunes. "Software metrics and security vulnerabilities: Dataset and exploratory study." *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016.

[17] Catolino, Gemma, et al. "Not all bugs are the same: Understanding, characterizing, and classifying bug types." *Journal of Systems and Software* 152 (2019): 165-181.

[18] Kukkar, Ashima, and Rajni Mohana. "A Supervised Bug Report Classification with Incorporate and Textual field Knowledge." *Procedia computer science* 132 (2018): 352-361.

[19] Zhou, Yaqin, and Asankhaya Sharma. "Automated identification of security issues from commit messages and bug reports." *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017.

[20] Zou, Deqing, et al. "Automatically Identifying Security Bug Reports via Multitype Features Analysis." *Australasian Conference on Information Security and Privacy*. Springer, Cham, 2018.

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

[22] Xia, Xin, et al. "Automatic defect categorization based on fault triggering conditions." *2014 19th International Conference on Engineering of Complex Computer Systems*. IEEE, 2014.

[23] Guo, Shikai, et al. "Ensemble data reduction techniques and multi-RSMOTE via fuzzy integral for bug report classification." *IEEE Access* 6 (2018): 45934-45950.

[24] Jain, Deepak Kumar, et al. "A particle swarm optimized learning model of fault classification in Web-Apps." *IEEE Access* 7 (2019): 18480-18489.

[25] Wijayasekara, Dumidu, Milos Manic, and Miles McQueen. "Vulnerability identification and classification via text mining bug databases." *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2014.

[26] Zibran, Minhaz F. "On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization." *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016.

[27] Chen, Dingshan, et al. "Automatically Identifying Bug Entities and Relations for Bug Analysis." *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*. IEEE, 2019.

[28] Karim, Md Rejaul, et al. "Understanding key features of high-impact bug reports." *2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 2017.

[29] Terdchanakul, Pannavat, et al. "Bug or not? bug report classification using n-gram idf." *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017.

[30] Wright, Jason L., Jason W. Larsen, and Miles McQueen. "Estimating software vulnerabilities: A case study based on the misclassification of bugs in MySQL server." *2013 International Conference on Availability, Reliability and Security*. IEEE, 2013.

[31] Lal, Sangeeta, and Ashish Sureka. "Comparison of seven bug report types: A case-study of google chrome browser project." *2012 19th Asia-Pacific Software Engineering Conference*. Vol. 1. IEEE, 2012.

[32] Ohira, Masao, et al. "A dataset of high impact bugs: Manually-classified issue reports." *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015.

[33] Jonsson, Leif, et al. "Automatic localization of bugs to faulty components in large scale software systems using Bayesian classification." *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016.

[34] Xia, Xin, et al. "Automated bug report field reassignment and refinement prediction." *IEEE Transactions on Reliability* 65.3 (2015): 1094-1113.

[35] Bhattacharya, Pamela, et al. "An empirical analysis of bug reports and bug fixing in open source android apps." *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013.

[36] Alenezi, Mamdouh, and Shadi Banitaan. "Bug reports prioritization: Which features and classifier to use?." *2013 12th International Conference on Machine Learning and Applications*. Vol. 2. IEEE, 2013.

[37] Umer, Qasim, Hui Liu, and Yasir Sultan. "Emotion based automated priority prediction for bug reports." *IEEE Access* 6 (2018): 35743-35752.

[38] Li, Xiaodan, et al. "A novel approach for software vulnerability classification." *2017 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2017.

[39] Jimenez, Matthieu, Mike Papadakis, and Yves Le Traon. "Vulnerability prediction models: A case study on the linux kernel." *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2016.

[40] Smith, Justin, et al. "How developers diagnose potential security vulnerabilities with a static analysis tool." *IEEE Transactions on Software Engineering* (2018).

[41] Somasundaram, Kalyanasundaram, and Gail C. Murphy. "Automatic categorization of bug reports using latent dirichlet allocation." *Proceedings of the 5th India software engineering conference*. ACM, 2012.

[42] Chawla, Indu, and Sandeep K. Singh. "An automated approach for bug categorization using fuzzy logic." *Proceedings of the 8th India Software Engineering Conference*. ACM, 2015.

[43] Chen, Rong, et al. "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution." *IEEE Transactions on Fuzzy Systems* (2019).

[44] Limsettho, Nachai, et al. "Automatic unsupervised bug report categorization." *2014 6th International Workshop on Empirical Software Engineering in Practice*. IEEE, 2014.

[45] Herzig, Kim, Sascha Just, and Andreas Zeller. "It's not a bug, it's a feature: how misclassification impacts bug prediction." *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013.

[46] Pingclasai, Natthakul, Hideaki Hata, and Ken-ichi Matsumoto. "Classifying bug reports to bugs and other requests using topic modeling." *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. Vol. 2. IEEE, 2013.

[47] Neysiani, Behzad Soleimani, and Seyed Morteza Babamir. "New labeled dataset of interconnected lexical typos for automatic correction in the bug reports." *SN Applied Sciences* 1.11 (2019): 1385.

[48] Tian, Yuan, et al. "Automated prediction of bug report priority using multi-factor analysis." *Empirical Software Engineering* 20.5 (2015): 1354-1383.

[49] Yang, Xin-Li, et al. "High-impact bug report identification with imbalanced learning strategies." *Journal of Computer Science and Technology* 32.1 (2017): 181-198.

[50] Tran, Ha Manh, et al. "An Analysis of Software Bug Reports Using Machine Learning Techniques." *SN Computer Science* 1.1 (2020): 4.

[51] Kanwal, Jaweria, and Onaiza Maqbool. "Bug prioritization to facilitate bug report triage." *Journal of Computer Science and Technology* 27.2 (2012): 397-412.

[52] Pandey, Nitish, et al. "Automated classification of software issue reports using machine learning techniques: an empirical study." *Innovations in Systems and Software Engineering* 13.4 (2017): 279-297.

[53] Huang, LiGuo, et al. "AutoODC: Automated generation of orthogonal defect classifications." *Automated Software Engineering* 22.1 (2015): 3-46.

[54] Thung, Ferdian, David Lo, and Lingxiao Jiang. "Automatic defect categorization." *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012.

[55] Luaphol, Bancha, et al. "Feature Comparison for Automatic Bug Report Classification." *International Conference on Computing and Information Technology*. Springer, Cham, 2019.

[56] Wu, Xiaoxue, et al. "CVE-assisted large-scale security bug report dataset construction method." *Journal of Systems and Software* 160 (2020): 110456.

[57] Ebrahimi, Neda, et al. "An HMM-based approach for automatic detection and classification of duplicate bug reports." *Information and Software Technology* 113 (2019): 98-109.

[58] Otoom, Ahmed Fawzi, Sara Al-jdaeh, and Maen Hammad. "Automated Classification of Software Bug Reports." *Proceedings of the 9th International Conference on Information Communication and Management*. 2019.

[59] Qin, Hanmin, and Xin Sun. "Classifying bug reports into bugs and non-bugs using LSTM." *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. ACM, 2018.

[60] Zhang, Tian-Lun, et al. "An uncertainty based incremental learning for identifying the severity of bug report." *International Journal of Machine Learning and Cybernetics* (2019): 1-14.

[61] Feldman, Ronen, and James Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

[62] Jurafsky andJamesH, Daniel. "Martin. Speech and Language Processing." (2009).

[63] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." *Journal of machine learning research* 12.Aug (2011): 2493-2537.

[64] Feldman, Ronen, and James Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

[65] Barcala, Francisco-Mario, et al. "Tokenization and proper noun recognition for information retrieval." *Proceedings. 13th International Workshop on Database and Expert Systems Applications*. IEEE, 2002.

[66] Willett, Peter. "The Porter stemming algorithm: then and now." *Program* (2006).

[67] Zhang, Harry. "The optimality of naive Bayes." *AA* 1.2 (2004): 3.

[68] Menzies, Tim, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors." *IEEE transactions on software engineering* 33.1 (2006): 2-13.

[69] Lessmann, Stefan, et al. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." *IEEE Transactions on Software Engineering* 34.4 (2008): 485-496.

[70] Loper, Edward, and Steven Bird. "NLTK: the natural language toolkit." *arXiv preprint cs/0205028* (2002).

[71] Yang, Yiming, and Jan O. Pedersen. "A comparative study on feature selection in text categorization." *Icml*. Vol. 97. No. 412-420. 1997.

[72] Bozorgi, Mehran, et al. "Beyond heuristics: learning to classify vulnerabilities and predict exploits." *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2010.

[73] Wang, Shuo, and Xin Yao. "Using class imbalance learning for software defect prediction." *IEEE Transactions on Reliability* 62.2 (2013): 434-443.

[74] Yang, Xinli, et al. "Automated identification of high impact bug reports leveraging imbalanced learning strategies." *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE, 2016.

[75] Ohira, Masao, et al. "A dataset of high impact bugs: Manually-classified issue reports." *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015.
[76] Jiang, Yue, Bojan Cukic, and Yan Ma. "Techniques for evaluating fault prediction models." *Empirical Software Engineering* 13.5 (2008): 561-595.