Bug Summarization, Severity Classification and Assignment for Automated Bug Resolution Process

Thesis submitted in fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

By

ASHIMA

Department of Computer Science Engineering and Information Technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

Waknaghat, Solan-173234, Himachal Pradesh, INDIA

December, 2019

TABLE OF CONTENTS

SUPERVISOR'S CERTIFICATE

This is to certify that the work in the thesis entitled **"Bug Summarization, Severity Classification and Assignment for Automated Bug Resolution Process"** submitted by **Ashima** is a record of an original research work carried out by her under our supervision and guidance in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science and Engineering in the Department of Computer Science and Engineering, **Jaypee University of Information Technology, Waknghat, INDIA.** Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Date:

Dr. Yugal Kumar
Assistant Professor (Senior Grade)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology,
Waknghat -173234, INDIA,

Dr. Rajni Mohana
Associate Professor
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology,
Waknghat -173234, INDIA,

DECLARATION OF SCHOLAR

I hereby declare that the work reported in the Ph.D. thesis entitled **"Bug Summarization, Severity Classification and Assignment for Automated Bug Resolution Process"** submitted at **Jaypee University of Information Technology, Waknghat, INDIA.** is an authentic record of my work carried out under the supervision of **Dr. Yugal Kumar and Dr. Rajni Mohana.** I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the contents of my Ph.D Theses.

Ashima

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology,

Waknghat -173234, INDIA,

Date:

ACKNOWLEDGEMENTS

First of all, I would like to express my deep sense of respect and gratitude towards my guide Dr. Yugal Kumar and co-guide Dr. Rajni Mohana in the Department of Computer Science and Engineering and Information Technology, for their support throughout this research work. I want to thank them for introducing me to the field of Software Engineering and giving me the opportunity to work under them. Without their invaluable advice and assistance, it would not have been possible for me to complete this thesis. I am greatly indebted to them for their constant encouragement and invaluable advice in every aspect of my academic life. I consider it my good fortune to have got an opportunity to work with such wonderful persons.

I would like to express my gratitude to our Honorable Vice Chancellor Prof. (Dr.) Vinod Kumar and Director & Academic Head Prof. (Dr.) Samir Dev Gupta to promote the research and facilitate resources in the institution. I would also like to thank Prof. Dr. Satya Prakash Ghrera, Head Department of CSE/IT for constant guidance, research facilities and resources to carry out my research work. I would also like to thank my doctoral committee members Dr. Pradeep Kumar Gupta, Dr. Kapil Sharma and Prof. Dr. Sunil Kumar Khah for their valuable feedback and critical reviews during presentations and time to time help.

I am also grateful for the support received from the JUIT, Waknaghat. In particular, I thank, Ravi Raina, Sanjeev Kumar and all staff at the Department of Computer Science and Engineering and Information Technology, JUIT, Waknaghat who has been extremely helpful at numerous occasions. I thank my fellow PhD friends for their consistent help and valuable discussion. I also thank my PG mates, for all the fun we have had in the last three years. Last but not the least, I would like to thank my family my parents and brother, for giving birth to me at the first place and supporting me spiritually throughout my life.

Ashima

ABSTRACT

Bug resolution process is an important aspect of the software development life cycle(SDLC). The aim of bug resolution process (BRP) is to determine the bugs in software and fix it. These software bugs are introduced in software's during the different phases of SDLC process. The different strategies and mechanisms are considered during the evolution of software to overcome the propagation of bugs. A significant amount of time, cost and effort is put on the identification of bugs. It is observed that some bugs are not identified during the software evolution. These bugs can lead to failure of software's and unexpected behavior. So, prior to delivery of software's, every company ensure that the software is bug free and meets it expectation. Hence, to address and manage the bugs, bug tracking or reporting system are designed such as Mozilla, Eclipse etc. These bug tracking systems store the information related to bugs, called bug repositories. The valuable information regarding for fixing the bugs are described in repositories. This information can be used to automate the BRP and also help developers in terms of reduced time and effort. The BRP is described in terms of report summarization, severity classification and assignment. This thesis addresses the issues associated with bug report summarization, bug severity classification and bug assignment. Bug report summarization is the process of generating the short description of lengthy bugs. To resolve the bug, initially developer analyses and understands the content of bug report and make a summary set. This process is time consuming and tedious as large number of bugs are deposited in repositories per day. In turn, bug fixing time can be increased. Therefore, to automate and improve the accuracy rate of summarization task, a new summary subset selection technique is proposed to determine optimal summary subsets. This proposed technique is based particle swarm optimization (PSO) and ant colony optimization (ACO). The aim of proposed technique is to address data redundancy and sparsity issues of bug reports. Further, the semantic relationship between sentences is also measured using informativeness and phareseness scores. This thesis also provides a solution for severity of bugs. It is noticed that all bugs are not equally important. Some, bugs require immediate attention of developer, whereas others are not. In this thesis, a new classifier is proposed to determine the severity of bugs. Prior to severity prediction, a deep learning based feature selection technique is also adopted to determine relevant features. Further, random forest with boosting method is applied for predicting the severity of bugs. The

performance of proposed classifier is tested over well bugs dataset. In this thesis, a developer recommendation (DevRE) system is also proposed to allocate the bugs with appropriate developers. The proposed system integrates the ACO based feature weighting technique to determine relevant feature from bug reports. Further, three metrics i.e. capability ranking, severity level and average bug fixing time are developed to rank the developers. The NB and SVM techniques are used to measure the severity levels of bugs. The results of proposed DevRE system is compared with existing recommendation systems. It is observed that proposed DevRE system provides more accurate results.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACO	Ant Colony Optimization
AP	Activity Profile
ADtree	Alternating Decision Tree
BTS	Bug Tracking System
BRP	Bug Resolution Process
BRC	Bug Report Classifier
BRS	Bug Report Summarization
BC	Binary Classification
BN	Bayesian Net
BFS	Bug Feature Selection
BRA	Bug Report Analysis
BSCM	Bug Severity Classification Model
CP	Concept Profile
CNN	Condensed Nearest Neighbor
CSTF	Categorical Summary and Long Description Features
CSF	Categorical Summary Features
CF	Categorical Features
CFS	Candidate Feature Selection
CH	Change History
DT	Decision Tree
DREX	Developer Recommendation and Expertise Ranking
DRETOM	Developer Recommendation based on Topic Models
DBA	Developer Based Analysis
DRS	Developer Recommendation System
DevRE	Developer Recommendation
EC	Email Classifier

EMC	Email Meeting Classifier
EM	Expectation Maximization
EB	Expertise
FFS	Final Feature Selection
FL	Fault Localization
IBRS	Intention based Report Summarization
ICF	Iterative Case Filter
KNN	K-nearest Neighbor
KLD	Kullback- Leibler Divergence
KLI	Kullback–Leibler Divergence Informativeness
KLP	Kullback–Leibler Divergence Phraseness
KSAP	K-Nearest-Neighbor Search Algorithm and Heterogeneous Proximity
LRCA	Logistic Regression with Crowdsourced Attributes
LDA	Linear Discriminant Analysis
LR	Logic Regression
LSI	Latent Semantic Indexing
MC	Multi-class Classification
MMLR	Multi-Nominal Multivariate Logistic Regression
MLP	Multi-Layer Perception
MI	Mutual Information
MLR	Multinomial Logistic Regression
ML	Machine Learning
MTG	Multi-Feature Tossing Graph
NB	Naive Bayes
NBM	Naive Bayes Multinomial
NFN	Nearest False Negatives
NFP	Nearest False Positives
NE	N-gram Extraction

PRST	PageRank based Summarization Technique
PSO	Particle Swarm Optimization
PO	Pareto Optimality
RIPPER	Repeated incremental pruning to produce error reduction
RFE	Request for Enchantment
RF	Random Forest
ROC	Receiver Operating Characteristics
RNG	Relative Neighbor Graph
RC	Refined Classification
RE	Root Extraction
RFB	Random Forest with Boosting
SDLC	Software Development Life Cycle
SBLC	Software Bug Life Cycle
SVM	Support Vector Machine
SBR	Security Related Bug Reports
SN	Social Network
STNT	Stanford Topic Modeling Toolbox
SWR	Stop Word Removal
TF-IDF	Term Frequency –Inverse Document Frequency
TDM	Term Document Matrix
TG	Tossing Graph

CHAPTER 1
INTRODUCTION

1 Introduction

Software maintenance phase is an important aspect of SDLC. The purpose of software maintenance phase is to fix the software bugs and improves the quality of software. A software bug can be described as fault, mistake, and error. Basically, it is human error that can feed unintentionally during the execution of the software especially either in source code or design. If, these bugs are not detected in earlier phase of SDLC, then it can lead to failure of software's and sometime unexpected behavior of software [1]. In turn, development cost of software's is increased and further also affected the delivery of software's. It is noticed that more than 90% of software development cost is spent on software evolution and maintenance activities [2]. In past few years, software maintenance can get wide attention from the research community. The main reasons are security risk, user inconvenience, inappropriate functionality, and large number of bugs. To handle aforementioned issues, bug tracking or reporting system are designed such as Mozilla, Eclipse, OpenFoam, JIRA, Trac, BugZilla, Redmine, OTRS, Mantis, FogBugz, BugNet etc. These systems detect the bugs in earlier phase of SDLC, prioritize the bugs and also ensure the timely delivery of softwares. The core component of bug tracking system (BTS) is bug repository. It contains various artifacts related to software's such as change history of status, source code and bug reports etc. The artifacts are used in software maintenance phase to resolve the bugs and also the key points for developers [3]. To remove the bugs from software is known as bug resolution process (BRP) and it is shown in Figure 1.1. This process is handled through triager or developer. In BRP, bug reports have significant impact. It contains diverse information like freeform text, attachment and predefined fields as shown in Figure 1.2. Freeform text contains the description, summary and comments. Attachments consist of non-textual information such as test cases, patches, etc. The predefined fields contain the various metadata, like status, product, importance, component, and assignee. Metadata is used to determine relevant features from bug report [4]. Further, the status in the predefined fields indicates the software bug life cycle (SBLC) [5]. The different status of SBLC are demonstrated in Figure 1.3.

- UNSOLVED: Initially, status of all bug reports is UNSOLVED.

- NEW: When, triager searches the appropriate developer for assigning the bugs, then status of bugs is changed, called NEW.

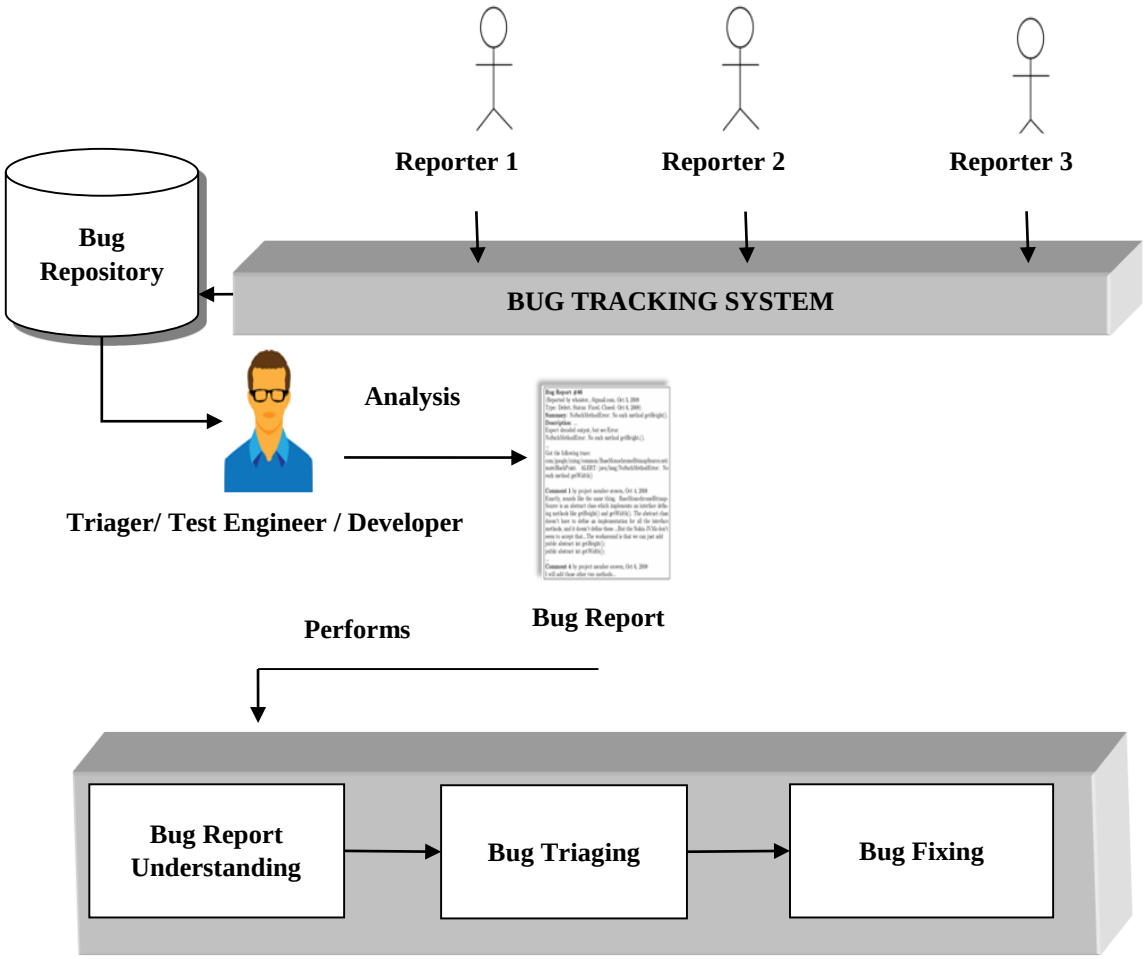


Figure 1.1: Bug Resolution Process

- ASSIGNED: Once, bugs are assigned to a relevant developer, its status is ASSIGNED.
- RESOLVED: When, bugs are resolved from the source code, its status is RESOLVED.
- REOPEN: If, the tester is not satisfied with the bug solution, then status of bug is REOPEN.
- VERIFIED: It corresponds to the approval of bug solution.
- CLOSED: It indicates the removal of bug.

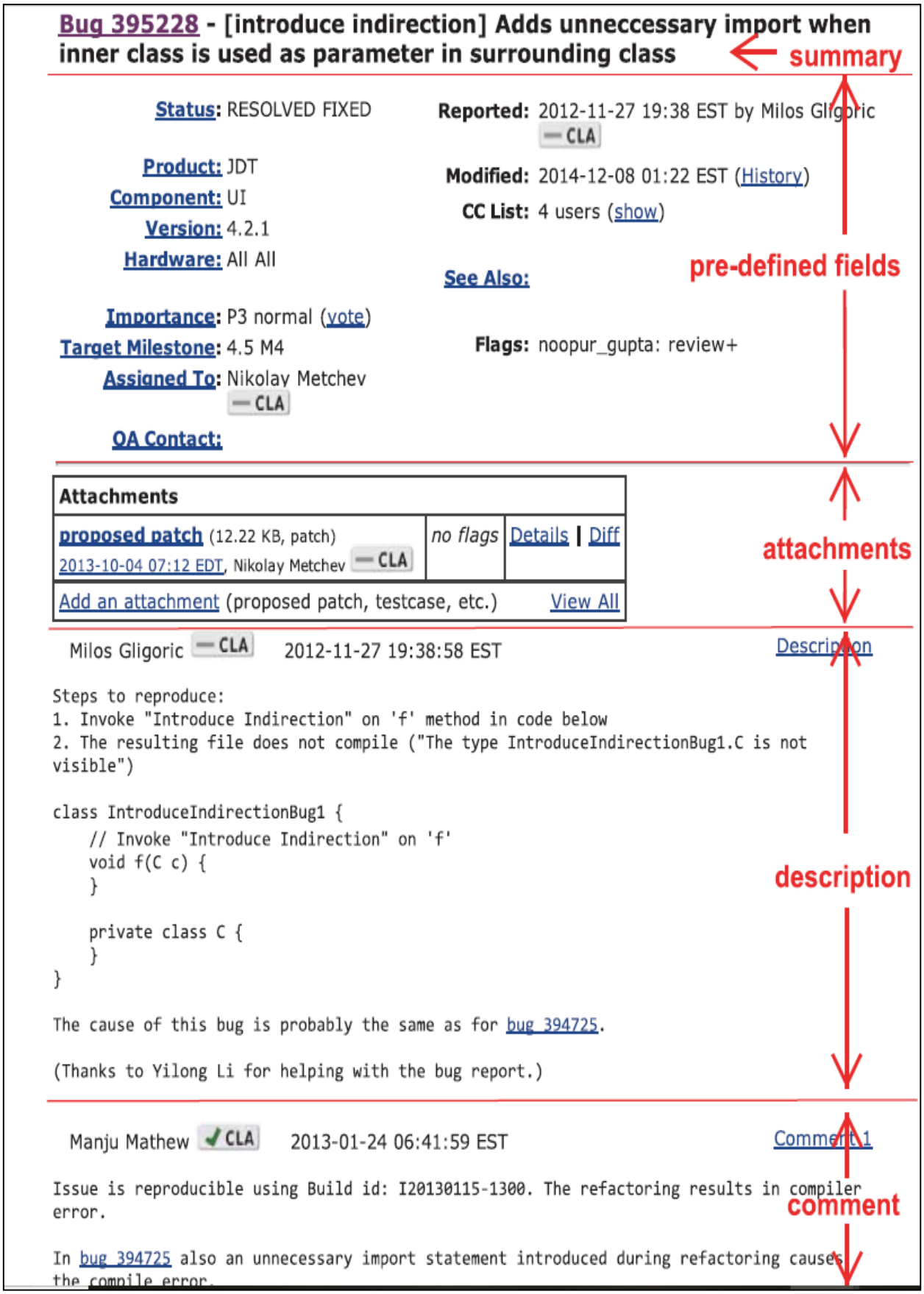


Figure 1.2: Eclipse Bug Report Example [6]

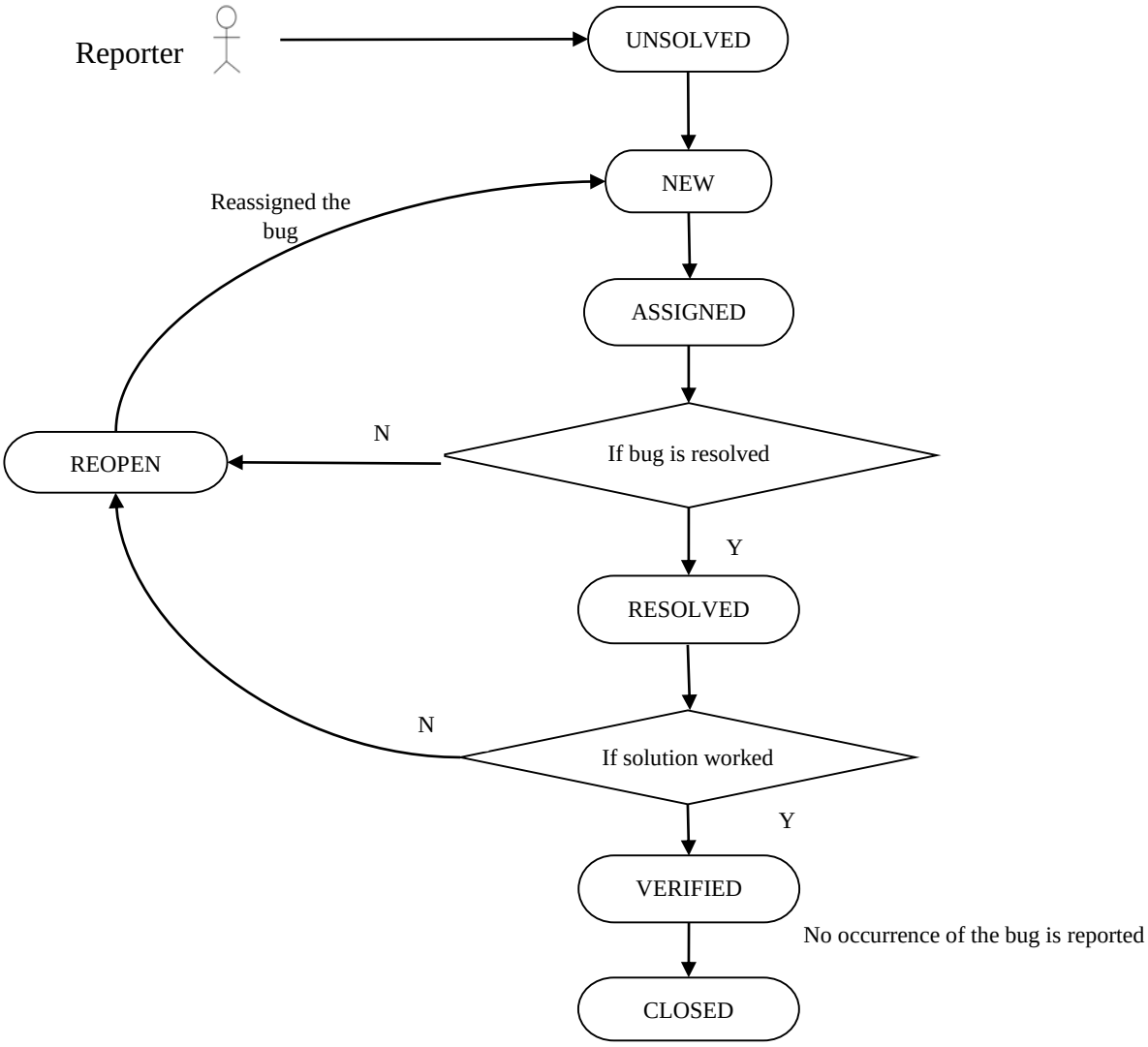


Figure 1.3: Bug Life Cycle Process

BRP is iterative process and consists of three phases. These phases are bug report understanding, bug triage and bug fixing.

- Bug Report Understanding Phase: In this phase, the status of bug report is changed from NEW to ASSIGNED. Triagers explore the contents of given bug reports for summarization, filteration of the duplicate bugs and features prediction (e.g., priority, severity and reopened/blocking). Large number of bug reports is generated every day and to summarize these bug reports manually is one of challenging task [7,8]. This issue attracts various researcher to work in this direction i.e. developed automated tool for bug report understanding.

- Bug Triaging Phase: This phase responsible to identify the appropriate developer to assign bugs [10-12]. This can be done with the help of triagers. Without prior information on bug report, triagers cannot determine relevant developer to fix the bug. Hence, bug assignment is also an important task to improve the bug fixing rate. To address the bug assignment issue, several bug triage approaches have been developed.
- Bug Fixing Phase: In this phase, bugs are fixed with the help of developers. For a given bug, developer analyzes the source code files, identifies the source of bugs and also updates the patch code as a bug-fixing process. But, manual bug localization and patch generation can make the bug fixing process difficult. Several researchers have also developed automatic tools to address bug fixing issues [13-15].

Each phase of bug life cycle associates with different challenges. The BTS contains the large amount of bug reports and it is the responsibility of triager to handle the bug reports. During the execution of Eclipse open source project, thirty new bug reports are stored in BTS daily from October 2001 to December 2010 and overall 333,371 bug reports are submitted to Eclipse bug repository [5]. The manual inspection of large scale bug reports is a tedious, and time consuming. Further, this process also puts heavy burden on triager and in turn low success rate for bug fixing. Many researches are worked in this direction, but the optimal solution is not obtained [25-33, 59-67, 92-97].

1.2 Motivation

Over the past few years, it is noted that bug resolution problem is popular research filed among researchers [13]. Research community can provide both theoretical and experimental solution for bug resolution problem. BRP is divided into bug report summarization, severity classification and assignment. The report summarization can be described as bug report understanding. While, Bug severity classification and assignment is the part of bug triaging. The bug resolution techniques are adopted to determine optimal solution for BRP. These techniques are based on textual content of description and summary fields. It is observed that lot of work is reported on automation of BRP in literature [3,13,11]; still, the reliability and accuracy are active area of research. It is noticed that several shortcomings are associated with bug resolution techniques. These are highlighted below:

1. Scope of improving the methods to handle bug report summarization, bug fixing etc.
2. Scope of improving the accuracy rate of bug severity classification.

3. Identification of appropriate developer for bug resolution.
4. Handle manual patch generation and bug localization as number of bugs are increased.

1.3 Objectives

The aim of this work is to address the summarization issues of bug reports such as data dimensionality, sparse representation and estimation issues of data and also improve the solution quality. Hence, the objective of this work based on the motivation section are given as

- To implement domain specific Bug Report Summarization technique
- To improve the accuracy of bug severity classification model
- To develop a developer recommendation system for bug assignment.

1.4 Structure of the Thesis

The entire work is organized in the following chapters.

Chapter 1: This chapter contains the information on bug resolution process, motivation and objective of the work.

Chapter 2: In this chapter, state of the art approaches for bug resolution process are presented. The literature review is divided into three sections i.e. Bug Report Summarization, Bug Severity Classification and Bug assignment.

Chapter 3: This chapter address the first objective of our work. In this chapter, a new bug report summarization model is proposed for summarization task. Further this model consists of PSO-ACO based summary subset selection algorithm. The proposed technique is the combination of PSO and ACO techniques. It is stated that proposed model provides state of art results for summarization problems.

Chapter 4: This chapter address the second objective of our work. In this chapter, a new bug severity classification model is proposed, called BSCM. The proposed model is based on deep learning and random forest with boosting method. The aim of proposed model is to improve the accuracy rate of severity classification task. It is stated that proposed model achieves higher accuracy rate than Zhou et. al.

Chapter 5: This chapter addresses the third objective of the work. In this chapter, a new developer recommendation (DevRE) system is proposed to assign the bugs for relevant

developers. The proposed model consists of ACO based feature weighting method for feature selection. Further, the developer is recommended on the basis of three metrics. The results of the proposed recommendation system are compared with existing recommendation system. It is observed that proposed recommendation system provided better results.

Chapter 6: This chapter concludes the entire work. Future research direction is also discussed.

CHAPTER 2
REVIEW OF LIERATURE

2.1 Introduction

This chapter presents the systematic literature review on bug resolution process. It is divided into three sections. These sections are

- Bug Report Summarization
- Bug Severity Classification
- BugAssignment

2.2 Bug Report Summarization

Bug report summarization is the process of generating the short description of lengthy bug report [16]. To know the root cause of the bug, large amount of conversation happens between the developer and reporter through the bug reports. This recorded conversation contains multiple comments from multiple peoples. The comments can be described either in terms of lines or passages and contains useful information for developer to resolve the bugs [17]. These resolved bugs are used as future reference [18,19]. After the submission of new bug report, developer first check the historical bug reports to see whether the similar bug was resolved previously. The similar bug helps the developer in two ways. The first one is understanding the current problem in better way. The second one is reusing the recorded solution of previous bug for new bug [20]. However, triager or developer reads the entire bug report to find the solution of previous bug report or to know the problem of new bug report. This activity consumes a lot of time, manual effort and frustrating due to the dozens of comments in bug reports [21]. Automatic bug report summarization is the only way to help the developers by reducing the size of bug reports. The basic approach of generating the bug report summary is to select the subset of existing sentences. The main issue of this process is the selection of summary subset [22,23]. In literature, large number of supervised and unsupervised approaches are adopted to determine the appropriate summaries [24-36]. Table 1 demonstrates the works reported on bug summarization problems.

To address the difficulties of manual bug report summarization, **Rastkar et al.** [24] tried to automate the bug report summarization. The aim of this work is to reduce the lengthy bug reports

into smaller ones. It should be beneficial for developer by two reasons. First, the developer consults with small bug reports. Second, it also reduces the time of developer. Hence, in this work, three supervised classifiers are used to automate the bug report summarization process. These classifiers are email classifier (EC), bug report classifier (BRC), email meeting classifier (EMC). The main work of these classifiers is to extract the top ranked sentences from the bug reports. The email thread dataset is used to train the EMC and EC. Whereas, BRC is trained using bug report dataset. All these classifiers are tested on 36 bug reports. These bug reports are taken from KDE, eclipse, gnome and Mozilla open source projects. The performance of these classifiers are evaluated using precision and F-value parameters. The results showed that BRC outperforms than EC and EMC. It is noticed that BRC achieves 63% precision rate and 40% F-value rate.

In continuation of their work, **Rastkar et al.** [25] explored the automated bug report summaries to determine duplicate bug report. The author used same classifiers to generate the summaries. These classifiers are trained on the set of 24 features. Further, the feature value of sentences is utilized to compute the probability of sentences. The sentences with higher probability are considered as a part of summary. Further, these 24 features are divided into four groups such as lexical, participant, structural and length features. The task-based validation was performed on 36 bug reports. The results showed that the generated summaries reduce the developer time to detect the duplicates bugs without affect the accuracy.

To determine the more relevant sentences for summarization task, **Jiang et al.** [26] conducted a survey to explore the existing techniques for attribute construction. In this work, a new model is developed, called Crowd–Attribute to determine the new effective attributes from crowd data. Further, eleven new attributes are constructed with Crowd – Attribute for developing a supervised summarization classifier called logistic regression with crowdsourced attributes (LRCA). The LRCA classifier is tested on 105,177 bug reports and SDS dataset. The effectiveness of proposed classifier is evaluated using precision, recall and F-score parameters. It is seen that LRCA classifier provides better results than the BRC technique [24]. It is noticed that LRCA improves the simulation results for Precision, Recall and F-score by 1.33%, 10.11%, and 8.94% respectively.

To find the multiple intentions of sentence in bug report, **Huai et al.** [27] implemented a supervised intention- based summarization model (IBRS). The IBRS model is divided into three steps such as feature extraction, intention classifier and summarization model. In first step, the features are extracted to generate the taxonomy for intentions in bug report. In second step, intention classifier is applied to get intentions from the sentences. In third step, intention taxonomy is leveraged to enhance the results for summarization task. The performance of IBRS is tested on 36 bug reports and evaluated using F-score, precision and recall parameters. The experimental results showed that IBRS outperforms than BRC [24].

To address the multidimensional and heterogeneous nature of bug reports, Ponzanelli et al. [28] developed an extension of LexRank approach [29]. LexRank approach basically consists of PageRank algorithm. In this work, PageRank algorithm is combined with custom similarity function for heterogeneous entities like configuration files and code samples. The proposed approach is tested on the stack overflow dataset. The simulation results are evaluated using precision parameter and compared with the text based approach i.e. classical LexRank. It is observed that proposed approach provides better summarization results than existing approach.

To reduce the developer time and effort, **Yeasmin et al.** [30] designed a prototype to visualize the extractive summary of bug reports. The working of proposed prototype is divided into two parts. Firstly, features are extracted using LDA modeling. While, in second part, bug report summaries are computed based on the cosine (lexical) similarity of each sentence with respect to other sentences. Moreover, API4 technique is adopted to determine the sentiment of each sentence and the relationship score of each sentence is also computed. Finally, the sentences are combined on the basis of relationship score and ranked. The sentences with higher score are selected to generate the summaries. Further, in this work, different colors and its combinations are used to visualize the summaries of bug reports. Bugzilla dataset is considered to assess the performance the proposed prototype. Precision, f-measure and recall are taken as performance parameters to evaluate the simulation results of proposed prototype. The simulation results of proposed prototype are compared with time sensitive keyword based extraction approach. Authors claimed that proposed prototype performs better than existing approach and archives higher f-measure, recall and precision rates.

Mani et al. [31] applied unsupervised approaches to automate the bug summarization process. The aim of this work is to explore the capabilities of unsupervised approaches in the field of automated bug summarization approaches. The various raking algorithms like centroid, grasshopper, diverse rank, and maximum-marginal relevance are applied to determine the relevant sentences for summarization task. The performances of aforementioned approaches are tested on 55 bug reports from SDS and DB2 repository. F-score parameter is used to evaluate the results of unsupervised approaches consider in this study. F-score of the centroid, grasshopper, diverse rank, and maximum-marginal relevance approaches are 43%, 50%, 46% and 48% respectively. Simulation results of unsupervised approaches are also compared with some supervised approaches. It is noticed that grasshopper and diverse rank approaches are performed better than supervised approaches [24]. It is also observed that unsupervised approaches significantly reduce the noise effect and also avoids human annotated summaries.

Lotufo et al. [32] presented an unsupervised method to automate the bug report summarization. Prior to generate the effective summaries, authors are designed three hypotheses to make a sentence relevant using Markov chain. These hypotheses are frequently discussed topics, evaluation and assessment of sentences and bug reports title and description. Further, the PageRank algorithm is implemented to determine the relevant summaries for bug report. The performance of PageRank approach is measured using debian, launchpad, mozilla and chrome datasets. Authors claimed that the proposed hypotheses are valid, and improves the simulation results up to 12% as compared to existing approach [24]. Moreover, the system-generated summaries are passed to developers to assess the quality and usefulness. Developers stated that system generated summaries are more useful, efficient and effective. But, it is observed that hypotheses can increases the cost of both approaches. Because, additional time is required to test the hypotheses.

To reduce developer time and effort for bug report summarization**, Nithya et al.** [33] developed an unsupervised extractive summarization approach. The proposed approach works in two modules. In first module corresponds to the noise reduction. In this module, bug report sentences are classified as code fragment, questions, investigative etc. The aim of this module is to remove unwanted sentences form bug reports. Further, centroid computation summarizer method is applied to compute the value of features in each sentence. In second module, an extractive summarization algorithm is to choose the optimum subset of sentences. This

algorithm determines the sentences with higher feature score and puts into summary subset. The similarity between the bug reports summaries are also determined to identify the duplicate bug reports. It is observed that the proposed unsupervised approach increases the accuracy of summarization and also capable to detect the duplicate bug reports.

To investigate the pitfalls of duplicate bug reports in summarization task, **Jiang et al.** [34] developed a page rank based summarization technique, called PRST. The aim of proposed technique is to summarize the textual content of bug reports and detect the duplicate bug reports. In PRST, three metrics are utilized with PageRank algorithm to measure the textual similarities between sentences These metrics are WordNet, vector space model and Jaccard. Further, a regression model is used to predict the probability of sentences. Moreover, it is observed that sentences are ranked on the basis of PageRank and regression model scores. Top ranked sentences are selected to generate the summary. The performance of PRST technique is assessed over two corpora i.e. MBRC and OSCAR. Simulation results stated that proposed PRST technique outperforms than BRC in term of F-score, recall and precision. It is also observed that combination of WordNet metric and PRST provides better computational results than PRST+VSM and PRST+Jaccard.

Ferreira et al. [35] applied cosine similarity, PageRank, Euclidean distance and Louvain community techniques to determine relevant sentences for summarization. All these techniques are implemented in unsupervised manner. The performance of these algorithms is tested on fifty bug report taken from the bootstrap, jquery and angular datasets. The simulation results of these techniques are evaluated using average precision score. The precision score of Cosine Similarity, PageRank, Euclidean Distance and Louvain techniques are 44.6%, 29.3%, 47.3% and 38% respectively. Further, two statistical tests i.e. Spearman's rank correlation coefficient and oracle ranking is adopted to determine the correlation between the rankings of each technique. It is noticed that Cosine Similarity and PageRank techniques are capable to produce average summary of bug reports than Euclidean Distance and Louvain techniques. The effort put on the analysis of bug reports is also reduced using Cosine Similarity and PageRank techniques.

Li et al. [36] explored the capability of deep learning network for bug report summarization. In this work, unsupervised deep sum approach is used to select relevant sentences for

summarization task. Further, stepped auto-encoder network with evaluation enhancement and predefined fields are utilized to integrate the bug report characteristics into a deep neural network. It is revealed that the proposed deep sum approach considerably reduces the eff ort put on the labeling huge training sets. The performance of proposed deep sum approach is evaluated using F-score and Rouge–n measures. It is seen that proposed approach state of art results than other compared approaches [24, 31,35]and it is one of viable and effective method for summarization problems.

Table 2.1: Demonstrates the works reported on bug summarization problems

Authors	Approach/ Method	Adoption Criteria	Dataset	Performance parameter	Advantage	Disadvantage
Rastkar et al. [24]	Email Classifier, Bug Report classifier, Email Meeting classifier	To extract the top ranked sentences from the bug reports	KDE Eclipse Gnome Mozilla	F-measure Precision Rouge score	BRC outperforms than EC and EMC	Generated summary is sensitive to the training dataset.
Rastkar et al., [25]	Email Classifier, Bug Report classifier, Email Meeting classifier	To determine duplicate bug reports using bug report summary	KDE Eclipse Gnome Mozilla	F-measure Precision Rouge score	Reduce the developer time to detect the duplicates bugs without affect the accuracy	Need of human annotated summaries
Jiang et al. [26]	Logistic regression with crowdsourced attributes (LRCA) classifier	To determine the relevant sentences for summarization	SDS	Precision Recall F-score	LRCA classifier provides better results than the Bug Report classifier	Need to generate the manual summaries for training the classifier
Huai et al. [27]	Intention- based summarization model (IBRS)	To find the multiple intentions of sentence in bug report	KDE Eclipse Gnome Mozilla	Precision Recall F-score	IBRS outperforms than Bug Report classifier.	Less summarization accuracy

Ponzanelli et al. [28]	PageRank and custom similarity function	To address the multidimensional and heterogeneous nature of bug reports	Stack Overflow	Precision	Proposed approach enhance the precision of summarization than classical LexRank	Raised the problem of convergence
Yeasmin et al. [30]	Prototype based on LDA modeling, Cosine similarity function and API4 technique	To extract the interactively visualizing insightful information	Bugzilla	Precision F-measure Recall	Proposed prototype performs better than time sensitive keyword based extraction approach	Need to include more interesting and useful visualization features
Mani et al., [31]	Centroid, Grasshopper, Diverse Rank, and Maximal Marginal Relevance	To increase the accuracy of supervised summarization approaches	SDS DB2	Precision F-measure Recall	Grasshopper and Diverse rank provides better results than supervised approaches	Ignore semantic relation between the sentences
Lotufo et al., [32]	Three hypotheses using Markov chain method	To explore the capabilities of unsupervised approaches in the field of automated bug summarization approaches.	Debian Launchpad Mozilla Chrome	Precision F-measure Recall	Proposed approach provides better results than Bug Report classifier	Require additional cost to test the hypotheses that decided the rank of each candidate sentence
Nithya et al. [33]	Centroid similarity function	To choose the optimum subset of sentences	Mozilla	Precision	Proposed summaries are capable to detect the duplicate bug reports	Less accuracy of summarization

Jiang et al. [34]	Page rank based summarization technique, called PRST	To investigate the pitfalls of duplicate bug reports in summarization task	MBRC OSCAR	Precision F-measure Recall	Proposed approach provides better results than Bug Report classifier	Dependent bugs consider for summarizing the bug reports
Ferreira et al., [35]	Cosine Similarity, PageRank, Euclidean Distance, and Louvain community detection	To determine relevant sentences for summarization	Bootstrap, Jquery and Angular	Precision	Cosine Similarity and PageRank produce better summary of bug reports than Euclidean Distance and Louvain techniques	Sparsity issues.
Li et al., [36]	Developed deep sum approach	To enhance the accuracy of summarization approaches	SDS	F-measure, Precision and Rouge score	Improved results than other approaches	Take more training time

2.3 Bug Severity Classification

Bug severity classification is a process of classifying the bug under appropriate severity similarly as the text classification [37]. The severity is based on the criticality and complexity of the bugs. It would also provide the complete detail of the impact of a bug within software operation [38]. Further, it also provides the advantage to industry for determining the earnestness of bug resolving in respect of resources [39-41]. Different bugs have different impact upon the quality, based on the functionality of software. The bug reports assigned with wrong severity goes through the reassignment process [42, 43]. The reassignment severity bugs take longer resolving time and influence the efficiency of developers [44]. There are various parameters set by the developer (Quality Assurance Engineer) according to different organizations [45]. It is essential for the developer to assign the correct severity due to the three

main reasons. The first one is to keep way from the confusion with the development group. The second one is to ensure the reliability in large- scale software and avoid the reassignment process. The third one is to resolve the critical bugs earlier than normal bugs because the critical bugs can trigger some security issues [46-48]. The manual severity classification is an error prone, tedious and time-consuming task due to large number of bug reports. Moreover, the precision of the classification depends on the developer's experience and knowledge who investigated the bugs. Therefore, there is a need to automate the bug severity classification process. In literature, large number of techniques has been presented for severity classification [49-71]. These techniques classified the bugs into binary (BC) and multiple severity classes (MC). Table 2 presents the works reported in literature on bug severity classification.

To predict the severity of bug reports, **Lamkanfi et al.** [49] developed a binary severity classification approach based on the textual description of bug reports. The severity of bug reports is divided into two classes i.e. server or non-server. The proposed approach contains five steps for bug severity prediction. These steps are described as extract and organize bug reports, preprocessing, training and evaluation set, train the classifier and apply the classifier on the evaluation set. The first step is to select the bug report from bug repository. The next step is the preprocessing step. In this step, feature vectors are extracted from the text of bug reports. Further, Naive Bayes classifier is applied to predict the severity of bug reports. The GNOME, Eclipse and Mozilla datasets are considered to evaluate the performance of NB classifier. Further, precision, recall and F-measure parameters are applied to validate the performance of classifier. The results demonstrated that NB classifier achieves higher precision, recall and f-measure rates as compared to other algorithms.

In the continuation of their work, **Lamkanfi et al.** [50] explored the efficiency of four machine learning classifiers to classify the bug into server or non-server classes. In this work, authors apply term frequency –inverse document frequency (TF-IDF) method to determine the probability of each feature. Further, four machine learning algorithms such as Naive Bayes (NB), Naive Bayes Multinomial (NBM), K-nearest neighbor (KNN) and Support vector machine (SVM) are applied to predict the server and non-server classes. The performance of these is tested on Mozilla, Eclipse and GNOME datasets. Accuracy parameter is used to evaluate the experimental results. It is revealed that NBM classifier obtains better results than other

algorithms. It is also stated that the average accuracy of all four machine learning algorithms are in between 73% to 85.7% using all three datasets.

To differentiate security bugs from normal bugs and also address the delay to identify and fix the security bugs, **Gegick et al.** [51] presented an automatic severity classification approach. This approach classifies the bugs into two classes i.e. security related bug reports (SBR) or non-SBR. In this work, text mining techniques are also implemented on the textual description of bugs to determine the feature vectors. Several feature selection algorithms like chi-square, correlation coefficient and info gain are applied to determine relevant set of features. Further, NBM classifier is applied on the reduced feature set to classify the bugs into SBR and Non-SBR classes. The performance of proposed NBM is assessed over Cisco, Mozilla and GNOME datasets. The experimental results indicate that feature selection algorithms improve the performance of NBM classifier significantly. It is observed that 77% of security bugs are classified as Non-SBR through reports, but the proposed NBM classifier corrects the same and classifies the security bugs as SBR.

Chaturvedi et al. [52] applied several machine learning techniques for bug severity prediction and described the severity classes on the scale of 1 to 5. In this work, info gain method is used to determine the appropriate terms from the text of bug reports. A total one twenty-five relevant terms are determined for severity prediction. Further, NB, NBM, SVM, KNN, C4.5, and RIPPER techniques are considered to predict the severity classes. The NASA dataset is selected to assess the performance of above mentioned machine learning technique. Further, the simulation results are validated using 5-fold cross validation method. It is also seen that bug severity can be described through different levels i.e. level 1 to level 5. In this work, using The results demonstrated that machine algorithms are suitable to determine different bug severity levels. Moreover, it is also noticed that severity level 2, 3 and 4 achieve higher F-measure rate than rest of levels using most of techniques.

Thung et al. [53] designed a new bug severity classification model based on the text and source code features of bug reports. In this study, JIRA dataset is considered for experimental work. Initially, relevant features are extracted using linear discriminant analysis (LDA) technique for classification task. Further, the extracted features are used to construct the new dataset with reduced number of features. The reduced dataset contains different severity classes

such as control, dataflow, structure, non-functional. Finally, the bug severity classification model is fed with the newly constructed dataset. The working of this model is divided into two phases- training and deployment phases. The various machine learning classifiers are integrated in proposed model. These are SVM, Decision Tree (DT) and NB. The performance of proposed model is evaluated using accuracy parameter. The experimental results showed that the proposed model achieves average accuracy rate of 77.8% using SVM classifier.

To determine the relevant features for automatic bug severity classification, **Yang et al.** [54] presented three feature selection methods. These feature selection methods are Information Gain, Chi-Square and Correlation Coefficient. The aforementioned methods are used to determine relevant severe and non-severe features for severity classification. Further, NBM classifier is applied to classify the data into different severity classes. The simulation results are evaluated using precision, recall and F-measure parameters. The Mozilla, Eclipse and GNOME datasets are considered for experimental work. It is observed that the combination of feature selection methods with MNB classifier improves the results of MNB classifier. Moreover, it is stated that combination of Information Gain and MNB classifier gives better results than others.

To handle the uneven categorization of the component of bug reports, **Murphy et al.** [55] presented three different variants SVM classifier based on TF-IDF, LDA, Kullback–Leibler divergence (KLD). These variants are SVM-TF-IDF, SVM-LDA and LDA-KLD. In SVM-TF-IDF variant, TF-IDF approach is used for extracting the features and SVM classifier is utilized to predict the severity of bugs. In SVM-LDA variant, LDA feature extraction approach is combined with SVM classifier. In LDA-KLD model, LDA is used to extract important features and KLD is utilized as classifier. The performance of these variants are tested over Bugzilla dataset and predicted the bug severity into high, medium, low and very low classes. The simulation results are evaluated using recall parameter. The experimental results showed that the proposed approach achieved the accuracy up to 75%. Further, it is observed that SVM-LDA obtains higher recall rate as compared to SVM-LDA and LDA-KLD. It is also observed that LDA- KLD provides more consistent than SVM-LDA and SVM-TF-IDF variants.

To understand the software system evolution and identify defected bugs, **Bhattacharya et al.** [56] developed graph based approach. The proposed approach also addresses the effort prioritization and bug severity estimation. In this work, authors design a set of metric based on

the graph theory concept. These metrics are based on the different properties of graph and can be described as modularity ratio, edit distance, assortativity, graph diameter, noderank, clustering coefficient and average degree metrics. The work of modularity Ratio metric is to predict modules with high maintenance effort. NodeRank metric measures the critical modules and functions with high severity indication. Edit distance metric analysis the developer collaboration graph with failure prone bugs. Eclipse and Mozilla datasets are considered to evaluate the performance of proposed graph based approach. It is stated that proposed approach obtains higher accuracy rate in comparison to other existing approaches. It is also observed that proposed approach efficiently works on functional and module levels.

Pingclasai et al. [57] presented a new model to automate the bug severity classification. The proposed model divides the bugs into binary severity classes i.e. bug or non-bug. This model consists of two phases. These phases are topic modeling and classification phase. The work of the topic modeling phase is to extract the appropriate features for classification phase. The relevant features are extracted using Latent Dirichlet allocation (LDA) algorithm. The aim of the classification phase is to predict the class labels. So, in classification phase, logic regression (LR), NB, ADTree techniques are considered. In this work, HTTP Client, jackrabbit and lucene datasets are taken into consideration. The simulation results are evaluated using accuracy parameter. The experimental results showed that F-measure rate of LR, NB and ADTree techniques varies in between 66% to 76% for HTTP Client dataset, 65% to 77% for Jackrabbit dataset and 71% to 82% for Lucene dataset. Authors claimed that NB classifier based model provides good results than LR and ADTree based models.

To improve the accuracy rate of bug severity classification model, **Nagwani et al.** [58] applied **s**imple random sampling and LDA algorithm based classification model for severity prediction. to select the taxonomic terms for classification. The working of the proposed model is divided into six steps. These steps are retrieve software bugs, text pre-processing, clusters using textual similarity, classification phase using random sampling and LDA, topics identification and filtration of topics. The performance of the proposed approach is assessed over Mozilla, MySQL, JBoss-seam and Android datasets. The experimental results revealed that proposed approach improves the accuracy rate considerably. It is noticed that the proposed approach more than seventy percent accuracy rate for all datasets.

Kanti et al. [59] developed a new approach to maintain the semantic relationship between the text of bug reports. In this work, several feature extraction methods are also considered to determine the relevant features from the bug reports. These techniques are Chi-square, Unigram and Bi-grams. The aim of these techniques is to compute the frequency of each feature. Features with higher frequency are selected for severity classification. The NB classifier is applied to predict the severity of bugs. The well-known Mozilla and Eclipse datasets are used in this work for severity prediction. Authors claimed that combination of unigram and bigram improves the accuracy rate of the NB classifiers. The accuracy rate achieved for Mozilla dataset is 73.5% to 85.5%, whereas accuracy rate of Eclipse dataset is 71.1% to 76.4%.

To explore the capabilities of unsupervised learning for severity prediction, **Limsettho et al.** [60] applied various clustering methods to classify the bugs into different classes. The classes of bugs are given as bug request for enchantment (RFE), improvement, task and test. In this work, two clustering algorithms i.e. Expectation Maximization (EM), and X-means are utilized for bug severity prediction. These methods group the similar bug reports on the basis of textual similarity. The NLP chucking is also used for labelling the clusters. The performance of the unsupervised methods is tested on three well known datasets and evaluated using accuracy parameter. From simulation results, it is noticed that both of EM and X-means clustering algorithms obtain similar results for Lucene and JCR datasets. But, EM algorithm obtains better accuracy rate than X-means for HTTPClient dataset. Moreover, the simulation results of unsupervised methods are also compared with supervised methods i.e. logistic regression and J48 classifiers. It is observed that the performance of supervised methods is better than unsupervised methods, especially logistic regression classifier provides optimum results than other methods.

Chawla et al. [61] developed a new automatic severity classification model to classify the bugs into two classes i.e. bug and other request. Authors adopt term frequency (TF-IDF) and latent semantic indexing (LSI) algorithms for selecting relevant features. Further, fuzzy logic based classifier is used for classification task. Three well known datasets are considered to validate the performance of fuzzy classifier. The simulation results are assessed using accuracy performance metric. The simulation results of fuzzy classifier are compared with LR, NB and ADTree classifiers. The experimental results indicated that fuzzy classifier obtains higher

accuracy rate than LR, NB and ADTree classifiers and average accuracy ranges in between 82-84% for all three datasets.

Zhang et al. [62] developed a concept profile based bug severity classification model. The proposed concept profile method is based on the analysis of previous bug reports. The working of proposed model is divided into three steps. In first step, bug reports are collected from various bug repositories and text of the bug reports are pre-processed to reduce the noise effect. In second step, concept profile method is introduced. The aim of this method is determine the term with higher frequency, called concept profile. In third step, the similarity between CP and query is calculated and further KLD technique is applied to classify the severity of bugs. The bugs are classified into blocker, trivial, critical, minor and major severity classes. The performance of proposed model is tested on Eclipse, Mozilla datasets and simulation results are compared with KNN, NB and NBM classifiers. It is stated that proposed model performs better than existing classifiers and produces quality of results. The f-value rate of proposed model varies in between 70%-96% for Eclipse dataset, and 78.57%-93.74% for Mozilla.

To enhance the performance of bug severity classification approach, **Gujral et al.** [63] developed a classification model based on the text mining technique and NBM classifier. The working of proposed model is divided into three steps. In first step, the text of Eclipse bug reports is pre-processed to identify the critical terms and these critical terms are stored in a dictionary. The task of critical terms is to decide the severity of bugs. The second step computes the occurrence of each feature to design a dataset. In this step, TF-IDF is applied to compute the frequency of features. The third step consists of the NBM classifier. The work of NBM classifier is to classify the bugs into server or non-server classes. The simulation results are evaluated using precision and accuracy parameters. It is seen that the proposed model achieves higher precision and accuracy rate. The precision rate of proposed model is 69% whereas, the accuracy rate is 72%.

In the continuation of their work, **Gujral et al.** [64] applied two feature selection methods to measure the different level of severities. These methods are Info gain and Chi square and aim of these methods is to determine critical terms for severity classification. All the critical terms are stored in the form of dictionary. Further, two machine-learning algorithms i.e. NBM and KNN is adopted to predict the severity of bugs. In this work, four models are developed for

severity prediction on the basis of feature selection methods and machine learning algorithms. These models are Info gain + NBM, Info gain + KNN, Chi- square + KNN and Chi- square + NBM. The performance of aforementioned models is evaluated using precision and accuracy parameters. The experimental results revealed that all models achieve higher accuracy rate using UI component. Moreover, it is also noticed that Chi- square + KNN model provides better severity prediction as compared to rest of models.

Zhou et al. [65] developed a multi-stage approach for improving the prediction rate of severity classification. The proposed approach is the combination of the data mining and text mining techniques. The proposed approach consists of three stages. In first stage, data mining approaches are applied to extract the relevant features from the summary of bug reports. Further, NBM classifier is trained on the extracted features and predicted the severity classes. The second stage consists of a set of structural features that are determined in stage 1. Further this set of features is used to train the Bayesian Net (BN) classifier for predicting the severity classes i.e. bug and non-bug. The third stage consists of data drafting approach and the work of this approach is to make the bridge between previous stages. This approach combines the output of the first stage with the selected features of the same bug reports. The proposed multi stage approach is tested on OpenFOAM, Jboss, Mozilla, Eclipse and Firefox datasets. The performance of proposed approach is compared with existing techniques using precision, recall and F-measure parameters. It is observed that f-measure rate is significantly improved using the proposed approach. The f-measure rates for OpenFOAM, JBoss, Mozilla, Eclipse and Firefox datasets are 85.9%, 93.7%, 81.7%, 80.2% and 79.5%.

To improve the classification rate, **Jin et al.** [66] presented a new method based on NBM classifier for automatic bug severity classification. The proposed method considers the product, component, reporter and severity as input along with the description and summary of bug reports. Further, the preprocessing phase is used to process the attributes of bug reports and output this phase is relevant attributes for classification task. These attributes are passed to the NBM classifier for classification purpose. The work of NBM classifier is to predict the label of bugs either severe or non-sever. The Eclipse and Mozilla datasets are considered for the experimental work and performance of proposed method is evaluated using F-measure parameter. The simulation results of proposed method are compared with Lamkanfi method

[49]. The simulation results demonstrated that proposed method is capable to separate the bugs into severe and non-severe label. It is also observed that proposed method obtains 80% and 83% F-measure rate for Eclipse and Mozilla datasets.

Pandey et al. [67] investigated the performance of different machine learning algorithms for bug severity classification. In this work, SVM with different kernel functions, NB, LDA, KNN, DT and random forest (RF) are considered for severity classification. The task of these algorithms is to predict the class labels of bugs i.e. server or non-server. Apart from these, Bag-Of-Words technique is applied to determine the relevant features for classification task. In this work, HTTP Client, Lucene and jackrabbit bug reports are taken for experimental work. The performance of above mentioned algorithms are evaluated in terms of average F-measure and accuracy parameters. The experimental results showed that RF gives better performance among all machine learning algorithms. It is also seen that SVM with sigmoid kernel function obtains higher F-measure and accuracy rates than rest of kernel functions.

Jindal et al. [68] presented a bug severity model to classify the bugs into high, medium, low and very low severity classes. In this work, the four variants of PITS datasets are used to validate the proposed model. These variants are PITS-A, PITS-C, PITS-D and PITS-E. The proposed model also considers the text mining methodology for extracting the relevant features from bug reports. The text mining methodology consists of three steps – preprocessing, feature selection and weighting. In first step, irrelevant words form the PITS dataset are removed. In second step, relevant attributes are identified using Info gain method. These features are the input of third step. In third step, the weights are given to each relevant feature using TF-IDF method. The output of this step is features with respective weight scores and higher weight score features are selected for classification task. In this study, three classifiers are used for predicting the severity classes. These classifiers are Multi-Nominal Multivariate Logistic Regression (MMLR), DT and Multi-Layer Perception (MLP). The performance of these classifiers are evaluated using receiver operating characteristics (ROC) parameter. It is stated that DT classifier provides better classification rate than MLP and MMLR classifiers.

To automate the bug severity classification process, **Mishra et al.** [69] developed a bug severity approach using cross datasets. The basic aim to consider the cross datasets is to identify the best training set using bug report summary of different datasets. Further, Bagging and Vote

techniques are used to reduce the imbalance issue of bug reports and also to extract the relevant features. In this study, K-NN, SVM and NB classifiers are applied to predict the bugs classes i.e. blocker, critical, major, minor and trivial. The performance of theses classifiers is evaluated using recall, precision and F-measure parameters. The experimental results showed the performance of K-NN classifier is better than SVM and NB classifiers. While, NB classifier exhibits worst performance among all classifiers. It is stated due to the cross training set, performance of K-NN classifier is enhanced especially with Eclipse dataset. Authors claimed that Eclipse dataset can be used to develop the classification model for Mozilla dataset and vice versa using K-NN and SVM classifiers and provide good results for cross datasets.

Sharmin et al. [70] designed a bug feature selection (BFS) method to identify the relevant features for severity classification. The working of BFS method is divided into four steps. In first step, the text of bug reports is preprocessed and Term Document Matrix (TDM) of each feature is built using TF-IDF method. In second step, mutual information (MI) technique is used to select the relevant features. The second step is further divided into two sub steps- candidate feature selection (CFS) and final feature selection (FFS). In CFS sub step, MI and chi-square methods are used to measure the statistical dependency between features and classes. The features with higher MI values are selected for next step. In third step, Pareto Optimality (PO) method is applied features to obtain complementary information regarding the classes and only those features can be selected for next steps that provide the complementary information. In fourth step, two classifiers are trained using selected features and classified the bugs into different severity classes i.e. blocker, trivial, critical, minor, major. These classifiers are DT and SVM. The performance of proposed method is compared with already existing approach using F-measure parameter [65]. Three public datasets are used to validate the existence of the proposed method. These public datasets are Eclipse, GCC and Mozilla. The experimental results revealed that proposed method provides higher F-measure value than other existing methods using all datasets.

To increase of bug severity classification rate, **Kumari et al.** [71] developed entropy based approach for severity prediction. The working of proposed approach is divided into two phases. In first phase, features are extracted using Info Gain method. In second step, extracted features are used to train the different classifiers to predict the bugs into different classes like blocker, trivial, critical, minor, and major. In this study, NB, KNN, J48, RF, Relative Neighbor Graph

(RNG), Condensed Nearest Neighbor (CNN) and Multinomial Logistic Regression (MLR) classifiers are used for severity prediction. Further, the Shannon entropy is calculated for severity classification to address uncertainty and irregular fluctuations in the BTS. The performance of these classifiers are evaluated using precision, recall, F-measure and accuracy parameters. Eclipse, PITS and Mozilla datasets are considered for experimental work. The results demonstrated that entropy function enhances the accuracies of all afore mentioned classifiers. Further, the simulation results of entropy based approach is also compared with Menzies et al. [72] and Zhang et al. works [62]. It is observed that proposed approach improves the F-measure rate in comparison to Menzies et al. and Zhang et al. approaches.

Table 2.2: Depicts the works presented on bug severity classification

Authors	Approach/ Method	Adoption Criteria	Dataset	Performance parameter	Advantage	Disadvantage
Lamkanfi et al., [49]	Naive Bayes classifier	To increase the accuracy of severity classification	GNOME Eclipse Mozilla	Precision F-measure Recall	Proposed approach achieves higher results as compared to other algorithms	Only trusted on the presence of a spontaneous relationship between the text and class
Lamkanfi et al., [50]	Naive Bayes, Naive Bayes Multinomial, K-nearest neighbor and Support vector machine	To explore the efficiency of machine learning classifiers to classify the severity of bug	Mozilla Eclipse GNOME	Precision F-measure Recall	Naive Bayes Multinomial classifier obtains better results than other classifiers	Need to extract more features from other attributes of bug report
Gegicket al., [51]	Chi-square, Correlation Coefficient and In-foGain with Naive Bayes Multinomial classifier.	To overcome the delay process of security bugs identification	Cisco Mozilla Eclipse	Precision F-measure Recall	Feature selection algorithms improve the performance of Naive Bayes Multinomial classifier significantly	Only considered the security related bugs

Chaturvedi et al., [52]	NB, MNB, SVM, k-NN, J48, RIPPER classifiers	To explore the performance of different classifiers on bug severity classification	NASA	Precision F-measure Recall	K-NN performed better than other classifiers	Required to perform on more projects/components of software projects to validate the applicability of machine learning techniques.
Thung et al., [53]	LDA with SVM, DT and NB classifier	To enhance the accuracy of bug report classifier	JIRA	F-measure Accuracy	LDA+SVM performed better than others.	Required to evaluate the performance of approach on more datasets
Yang et al., [54]	Information Gain, Chi-Square and Correlation Coefficient with NBM classifier	To determine the relevant features for automatic bug severity classification	Mozilla Eclipse GNOME	Precision F-measure Recall	Information Gain + NBM classifier gives better results than others	Ignore semantic features
Murphy et al., [55]	TF-IDF and LDA with Kullback–Leibler divergence and SVM	To increase the performance of SVM classifier	Bugzilla	Recall	SVM-LDA obtains higher recall rate as compared to SVM-LDA and LDA-KLD	Require more dataset and components
Bhattacharya et al. [56]	Graph-based analysis	To understand the software system evolution and	Eclipse Mozilla	Accuracy	Proposed approach obtains higher accuracy rate in	Required additional cost for generating the graphs

		identify defected bugs			comparison to other existing approaches	
Pingclasai et al., [57]	Topic-based model, LDA with Logic Regression , NB , ADTree classifiers	To automate the bug severity classification	HTTPClient Jackrabbit and Lucene	Accuracy	Topic based model +NB achieves higher accuracy	Ignore the issue of chronological change of bug occurrence
Nagwani et al., [58]	Simple Random Sampling and LDA	To improve the accuracy rate of bug severity classification model	Mozilla, Mysql, JBoss-Seam and Android	Accuracy	Generation of taxonomic terms increase the accuracy of severity classification	Need to extract the more features
Kanti et al., [59]	Chi-square and Bi-grams methods with Naive Bayes classifier	To maintain the semantic relationship between the text of bug reports	Mozilla Eclipse	Accuracy	Bi-gram slightly enhance the performance of the classifier.	Ignored the information dropping issues
Limsettho et al., [60]	Top modeling-Heuristic Dirichlet Allocation (HDA), Expectation Maximization (EM) and X-means algorithms	To explore the capabilities of unsupervised learning for severity prediction	Lucene	Accuracy	Performance of unsupervised algorithms was nearby supervised classifier	Require to improve the classification Model increasing its performance
Chawla et al., [61]	Term frequency Latent semantic	To explore the performance of different classifiers on	Google Chrome	Accuracy	Fuzzy classifier obtains higher accuracy rate than LR, NB and	Need to extract other attributes like description, comments

	indexing with Fuzzy logic	bug severity classification			ADTree classifiers	
Zhang et al.,[62]	Concept profile approach based on KL-divergence	To increase the accuracy of severity classification	Eclipse Mozilla	Precision F-measure Recall	Proposed model performs better than existing classifiers	Required a large amount of historical data
Gujral et al., [63]	TF-IDF with NBM	To enhance the performance of bug severity classification approach	Eclipse	Precision Accuracy	Proposed model achieves higher precision and accuracy rate	Required to increase the accuracy rate of classification
Gujral et al., [64]	In-foGain and Chi-square with K-NN and NBM classifier.	To increase the accuracy rate of classification	Eclipse	Precision Accuracy	Chi- square + KNN model provides better severity prediction as compared to Info gain + NBM, Info gain + KNN, and Chi-square + NBM	Ignore semantic relations of text
Zhou et al., [65]	BN and NB classifier	To improve the prediction rate of severity classification	OpenFOAM, Jboss Mozilla Eclipse Firefox	Precision F-measure Recall	Proposed approach achieves higher F-measure rate than existing approaches	Only considered misclassified bug reports.
Jin et al., [66]	NBM classifier	To improve the classification rate	Eclipse Mozilla	F-measure	Meta -fields of the normal bug severity report are helpful in increasing the accuracy of classification	Ignore the semantic features in meta-fields of bug report

Pandey et al., [67]	Bag-Of-Words with NB, LDA, KNN, DT and random forest (RF) classifier	To investigate the performance of different machine learning algorithms for bug severity classification	HttpClient Lucene Jackrabbit	F-measure Accuracy	RF gives better performance among all classifiers	Considered the sentences as the stack of words
Jindal et al. [68]	Info-Gain and TF-IDF with Multi-nominal Multivariate Logical Regression (MMLR), Multilayer Perception (MLP) , (DT) classifiers	To increase the accuracy of bug severity prediction	PITS A-E	ROC	DT classifier provides better classification rate than MLP and MMLR classifiers	Constrained generalization capability
Mishra et al., [69]	Bagging, Vote technique, In-fogain, TF-IDF with NB , SVM and K-NN classifier	To investigate the influence of cross projects on classifiers for bug severity classifier	Eclipse	Precision F-measure Recall	Performance of K-NN classifier is better than SVM and NB classifiers due to the cross training set	Unordered sequence of words.
Sharmin et al., [70]	TF-IDF, Mutual information (MI) and Pareto optimality	To identify the relevant features for severity classification.	Eclipse GCC Mozilla	F-measure	Selects fewer number of terms and requires less computation than the existing approaches	Proposed approach might require further improvement to get satisfactory result
Kumari et al., [71]	TF-IDF and Info-Gain	To increase of bug severity	Eclipse PITS	Precision Recall,	Entropy function	Ignore the problem of

	with NB, KNN, J48, RF, Relative Neighbor Graph (RNG), Condensed Nearest Neighbor (CNN) and Multinomial Logistic Regression (MLR) classifiers	classification rate	Mozilla	F-measure Accuracy	enhances the accuracies of all classifiers	imbalance dataset

2.4 Bug Assignment Approaches

Bug assignment is the process of assigning the relevant developer based on the severity for resolving the bug [73]. Traditionally, the new bug reports are manually assigned by a human expert called triager. The manual bug assignment process is very time consuming, tedious and prone to error. The main reason is lack of information about the developer who has experienced of resolving similar types of bugs [74-76]. Further it also increases the probability of reassigning the bug. This bug reassignment procedure increases the general bug fixing time [77]. Therefore, there is need to automate the bug assignment process. The aim of bug assignment is to reduce the probability of reassigning the bugs [78]. It also reduces the triages time and effort to assign relevant bug fixer or developer based on the severity [79,80]. However, to determine relevant developer and rank is one of challenging task. In literature, previous researches implemented machine learning (ML), expertise (EB), social network (SN) and tossing graph (TG) based models to determine the relevant developer for resolving the bug [81-98]. Table 3 illustrates the works reported on bug assignment and developer identification approaches in literature.

Xuan et al. [81] developed semi-supervised classifier to address the deficiency associated with labeled bug reports in existing supervised approaches. The working of proposed classifier is divided into two phases. In first phase, NB classifier is trained with labelled bug reports. In second phase, the expectation maximization (EM) algorithm is applied on the combination of

labeled and unlabeled bug reports. The expectation part associates the appropriate label to unlabeled bug reports. While, maximization part rebuilds a new classifier with the help of labeled bug reports. Further, a weighted recommendation function is integrated with EM algorithm for assigning the weights to relevant developers during the training process. The performance of proposed semi-supervised classifier is evaluated using accuracy parameter. Eclipse dataset is considered for experimental work. The simulation results of proposed semi supervised classifier are compared with NB and NB with EM classifiers. It is observed that the accuracy rate of proposed semi-supervised classifier is higher than NB and NB with EM classifiers. It is revealed that semi supervised classifier improves the classification accuracy up to 6% in comparison to supervised classifiers.

Bhattacharya et al. [82] presented a new algorithm based on ML and TG to improve the accuracy rate of bug assignment task and reduce the length of tossing path. In this work, accuracy issue is addressed through refined classification (RC) approach based on additional features and intra-fold updates during training phase. Whereas, a ranking function is developed to recommend potential tosses in TG and multi-feature TG. The working of proposed algorithm is divided into four stages. In first stage, NB and BN classifiers are applied for training purpose and built the TG. In second stage, the potential developer is predicted using the TG and classifiers i.e. NB and BN. The third stage corresponds to the performance evaluation of training set. In this stage, accuracy of classifiers is evaluated for bug assignment. In fourth stage, developer class of test set is predicted using NB, BN and TG. TG is also updated in this stage. The proposed algorithm is validated using Mozilla and Eclipse datasets and also compared with ML based algorithm. The experimental results demonstrated that proposed algorithm reduces the tossing path length and also obtains higher accuracy rate. The accuracy rate of proposed algorithm for Mozilla and Eclipse datasets is 84% and 82.59% respectively.

To help the traiger and determine the relevant developer, **Anvik et al.** [83] examined the capabilities of different machine learning techniques such as NB, SVM, EM and C4.5. Three different developer recommendation system is developed using the aforementioned techniques. The first model is developer recommender system and aim of this model is to identify the developer for fixing the bugs. The second model is component recommender system and it can determine the product component of bug reports. The third model is interest recommender

system that can determine the interest of a developer for a particular project. The performance of these recommendation system is tested on Bugzilla, GCC, Eclipse, Mylyn and Firefox datasets. The results demonstrated that SVM technique based recommendation system performs better than others techniques based recommendation systems. It is also observed that component based recommendation system improves the precision rate considerably. The precision rate achieved for Eclipse, Bugzilla, Mylyn and Firefox datasets is 76%, 96%, 98% and 73% respectively. A field study with four tragers is also reported in this study. The traigers stated that proposed recommended systems reduce the bug fixing time of developers.

A ML based training set reduction approach is presented by **Zou et al.** [84]. The proposed approach integrates the feature selection algorithm with instance algorithm. The aim of this integration is to reduce the redundant and noisy information from datasets as well to improve the accuracy of bug assignment. The proposed approach is the combination Chi square and instance selection algorithm. Chi-square method is used to find the dependency between features and developers. Whereas, instance selection algorithm especially iterative case filter (ICF) is used to filter the noise and instance reduction. It is also seen that different variants are developed on the basis of Chi- square and ICF. These combinations are Chi-square, ICF, Chi-square+ ICF and ICF +chi-square. Further, KNN method is utilized to predict the class label of instances. In this study, Eclipse dataset is considered for experimental work. The performance of proposed variants is evaluated using precision, recall, F-measure and accuracy parameters. The experimental result demonstrated that Chi-square+ ICF and ICF +chi-square variants significantly reduce size of the training data and also obtains improved simulation results than traditional Chi-square and ICF.

To reduce the bug fixing time and identify relevant developer for bug fixing, **Tamrawi et al.** [85] developed EB model, called Bugzie. The proposed model consists of fuzzy set based approach to determine the correlation between the multiple technical terms and identify the experience developer. The correlation information is used to recommend the relevant developer. Further the Bugzie model also computes the membership scores of each developer and it is used to describe correlation between the fixers and terms. The performance of proposed model is tested on Firefox, Eclipse, Apache, NetBeans, FreeDesktop, Gcc and Jazz datasets and compared with SVM and NB models using accuracy parameter. The results showed that Bugzie

model gives better accuracy rate than others models. The highest accuracy rate of proposed Bugzie model is 72% and it is also noticed that developers take less time for bug fixing.

Wu et al. [86] developed a SN based recommendation model for recommending the developer for bug fixing and called developer recommendation with KNN and expertise ranking (DREX). The proposed model consists of two components. The first component contains the KNN method and it is used to compute the similarity between historical and new bug report. In second component, simple-frequency method and six social network metrics such as PageRank, in-degree, betweenness, closeness and outdegree centrality are used to select relevant developer for bug fixing. The performance of proposed model is evaluated using precision, recall and F-score parameters. The experimental result showed that Outdegree and simple frequency metrics performs better than other metrics and attains more than 0.6 recall rate for Mozilla and Firefox datasets

Servant et al. [87] presented an EB based developer recommender system to determine the relevant developer for fixing the bugs. This system consists of three components. These components are fault localization (FL), change history (CH) and expertise mapping (EM). The aim of FL component is to extract the information about failure locations. Whereas, CH component is used to collect the information regarding code editing task performed by developer. The work of EM component is to search the compatible developer. The proposed system is implemented using WHOSEFAULT tool and simulation results are evaluated using accuracy parameter using AspectJ dataset. It is revealed that that proposed system recommend the same developers upto 81% and improves the results of baseline technique. Moreover, it is also seen that simulation results of expertize technique is compared with existing expertise assessment techniques. It is found that proposed expertize technique achieves greater accuracy rate.

To address the developer assignment and assignee task**, Xuan et al.** [88] presented SN based approach for improving the performance of bug assignment. In the proposed SN based approach, social network analysis is carried out to rank the developers who participate in the commenting process of bug fixing task. Further, NB and SVM algorithms are used to determine the appropriate developer. It is observed that authors also consider the developer assignment to handle three issues – ranking of developers, evolution and tolerance. The performance of proposed approach is investigated on Mozilla and Eclipse datasets using accuracy parameter.

The results indicate that the social network analysis improves the average accuracy of NB and SVM by 2% and 10%, respectively. It is also seen that developer assignment can have significant impact on bug resolution tasks.

To assign the relevant developer for bug fixing, **Xie et al.** [89] developed TG based model, called developer recommendation based on topic models (DRETOM). The working of DRETOM is divided into three steps. In first step, Stanford Topic Modeling Toolbox (STNT) is used for grouping the bug reports according to topics. In second step, associations between developer and topic is described through probability function. This function formulates the relation between developer expertise and interest on bug fixing. Third step ranks the developers on the probability function. The performance of DRETOM is evaluated using Eclipse JDT and Mozilla datasets. The simulation results of DRETOM is compared with Bugzie [85] and DREX [86] model using recall parameter. Experimental results showed that DRETOM outperforms than Bugzie and DREX model and achieves 82 % recall rate or Eclipse JDT and 50% for Mozilla.

In continuation of their work, **Bhattacharya et al.** [90] adopted ML and TG based approach for bug assignment or fixing. In this work, authors consider product and component of the bugs to determine the relevant features for bug dataset. These features are used to train the various ML based classifiers like NB, SVM, C4.5 and BN along with bug tossing graph. The aim of bug tossing graph is to reduce the tossing length. The performance of these classifiers are tested on Mozilla and Eclipse datasets and evaluated using accuracy parameter. It is seen that NB and BN classifiers performs better than SVM and C4.5 classifiers. Further, the NB classifier also outperforms than BN classifier by reducing the tossing path length for Mozilla and Eclipse up to 86% and 83% respectively. The NB classifier obtains 77.87% (for Mozilla) and 77.43% (for Eclipse) accuracy rate. The author also implements the ablative analysis to measure the importance of software attributes for improving the accuracy rate. It is revealed that aforementioned techniques improve the accuracy rate with reduced training and prediction time.

Xia et al. [91] presented EB based approach to designed a developer recommender system, called DevRec. The primary task of this system is to perform bug report analysis (BRA) and developer based analysis (DBA) for determining the relevant developer. In BRA, the similarity information between new bug reports and historical bug reports is computed. This information can be used to find the list of developers that have some experience for solving similar bug in

past. In DBA, the affinity of each developer is calculated based on the characteristics of bug report. This information is used to find the set of relevant developers. The performance of DevRec model is evaluated using Mozilla, NetBeans and Eclipse datasets. The simulation results of DevRec model is compared with DREX [86] and Bugzie [85] using recall parameter. The simulation results showed that DREX model improves the recall rate than DREX and Bugzie models. It is also observed that DREX model exhibits the worst performance among all three models.

Naguib et al. [92] developed EB based approach based on the developer activities for bug assignment. The working of proposed approach is divided into two steps. In first step, LDA algorithm is adopted for clustering the bug reports into different clusters. In second step, activity profile (AP) of each developer is generated using topic model and history log. AP indicates the developer role, involvement and expertise for resolving the bugs. All these activities are helped to identify and rank the relevant developers. Hit score parameter is used to evaluate the performance of proposed approach. Further, ATLAS reconstruction, UNICASE, Eclipse and BIRT datasets are considered in this study. The simulation results of proposed approach are compared with LDA-SVM based model. The experimental results revealed that proposed approach performs better than LDA-SVM based model and achieves more than 88% hit ratio for all datasets.

To improve the accuracy of bug assignment, **Zhang et al.** [93] developed TG based approach based on the topic model and relations of developers. The aim of proposed approach is to rank the developers on the basis of developer experience and interest for resolving the bugs. The proposed approach implements LDA for topic extraction from the previous bug reports. Further, the relationship of developers and topics are analyzed using social network. The performance of proposed approach is validated using Eclipse, Mozilla Firefox, and Netbeans datasets. The simulation results are evaluated using recall, precision and F-measure parameters. It is observed that proposed approach provides higher F-measure rate than DRETOM [89] and AP [92].

Yang et al. [94] developed TG based recommendation system for bug fixing and identification of relevant developer. The proposed system consists of different methods such as topic model method, core NLP and Kullback-Leiber. Topic model and core NLP methods are used for pre-processing, while Kullback-Leiber method is used to extract features of developers

such as same topic, priority, product, component and severity of bug reports. Moreover, the social network technique is utilized to capture the commenting activities of developer. The proposed system is tested on Eclipse, Mozilla and NetBeans datasets. The experimental results showed that proposed system obtains better recommendation results than Out-Degree [86], AP-based recommender [92] and DRETOM [89], using all datasets.

In the continuation of their previous work, **Zhang et al.** [95] integrated the K-nearest-neighbor search algorithm and heterogeneous proximity (KSAP) to improve the accuracy of bug assignment. The proposed KSAP works with heterogeneous network and historical bug reports and consists of two phases. In the first phase, KNN is implemented to determine the similarity between historical bug reports and new bug reports. In the second phase, the developers are ranked on the basis of its capabilities to resolve similar kind of bugs using heterogeneous proximity. The efficiency of proposed KSAP is investigated over Eclipse, Mozilla, Apache and Tomcat datasets and simulation results are evaluated using recall, precision and F-measure parameters. The simulation results of KSAP approach is compared with existing ML-KNN, DREX, DRETOM, Bugzie, DevRec and DP models. It is seen that KSAP provides improved results in comparison to existing models.

Lui et al. [96] develop a developer recommendation system based on multiple sources to avoid cold start problem in bug assignment, called DRS. In this work, authors consider two optimizing objectives i.e. time spent for bug fixing (time cost) and accuracy to determine the appropriate bug fixer. Further, modern portfolio theory is used to maintain the balance between time cost and accuracy. In DRS, LDA is used to search similar topics. Further, it computes the multiple scores for each developer based on time, cost and modern portfolio theory. The score is used to rank the developers. The effectiveness of the proposed DRS is validated using Bugzilla dataset and its performance is evaluated using average time to fix the bugs. It is observed that DRS can reduce the cold start problem of bug assignment considerably through assign the relevant developers.

Yadav et al. [97] adopted EB based approach to determine the appropriate developer for bug fixing. The proposed approach works on the developer expertise scores. The developers scores are computed using Jaccard and cosine–similarity methods. These methods measure the similarity between fixing time, priority and versatility of bug reports. Further, the developers

are ranked on the basis of developers score. The proposed approach is tested on FreeDesktop, Eclipse, Firefox, Mozilla and NetBeans datasets. The simulation results of proposed approach are also compared with existing ML approaches such as NB, SVM and C4.5 in term of precision, recall and F-measure parameters. The results demonstrated the proposed approach provides significant results than existing ML approaches and also reduced the tossing length.

To handles the bugs in effective manner, **Kanwal et al.** [98] designed a machine learning based bug assignment model. The proposed model is classified into four phases. In first phase, the text of bug reports is pre-processed to reduce the noise effect in text. In second phase, the text and five categorical features are extracted. These features are categorical, summary and long description (CSTF), summary (SF), summary and long description (TF), categorical and summary (CSF) and categorical (CF). In third phase, NB and SVM classifiers are trained with different combinations of features for classifying the bugs into different priority classes. In fourth phase, a set of developer is assigned according to predicted priority. The performance of NB and SVM classifiers are evaluated using accuracy, precision, recall, nearest false negatives (NFN) and nearest false positives (NFP) parameters. The aim of NFN and NFP parameter is to determine optimum combination of features for measuring the priority of bugs. It is observed that SVM classifier performs better than NB classifier for text features. While, NB classifier obtains better results than SVM classifier for categorical features. It is also observed that SVM achieves higher accuracy with the combination of text and categorical features.

Table 2.3: Illustrate the works reported on bug assignment and developer identification approaches

Paper	Approach/Method	Adoption Criteria	Dataset	Performance parameter	Advantage	Disadvantage
Xuan et al., [81]	Expectation maximization (EM) with Naive Bayes classifier.	To address the deficiency associated with labeled bug reports in existing supervised approaches	Eclipse	Accuracy	Accuracy rate of proposed semi-supervised classifier is higher than NB classifier	Required additional cost for probabilistically labeling the unlabeled bug reports.

Bhattacharya et al., [82]	Refined classification, ranking function and multi-feature tossing graph (MTG)	To improve the accuracy rate of bug assignment task and reduce the length of tossing path	Mozilla Eclipse	Accuracy	Proposed algorithm reduces the tossing path length and also obtains higher accuracy rate	Required more features to generate the multi-feature tossing graph
Anvik et al. ,[83]	Component-based technique with Naive Bayes, SVM, EM and C4.5, classifier.	To examine the capabilities of different machine learning techniques	Bugzilla GCC Eclipse Mylyn Firefox	Precision	SVM classifier based recommendation system performs better than others techniques based recommendation systems	Needs to extract the various feature 'component'
Zou et al., [84]	Chi square and instance selection with KNN classifier	To reduce the training dataset	Eclipse	Precision Recall, F-measure Accuracy	Chi-square+ instance selection variants significantly reduce size of the training data and also obtains improved simulation results than traditional Chi-square	Requires more features

					and instance selection	
Tamrawi et al., [85]	Bugzie based on fuzzy set	To reduce the bug fixing time and identify relevant developer for bug fixing	Firefox, Eclipse, Apache, NetBeans, FreeDesktop, Gcc Jazz	Accuracy	Bugzie model gives better accuracy rate than others models	Required additional cost to generate the technical terms list that are extracted from the software systems.
Wu et al., [86]	Developer recommendation with KNN and expertise ranking (DREX)	To enhance the accuracy of bug assignment model	Mozilla Firefox	Precision Recall, F-measure	Outdegree and simple frequency metrics performs better than other metrics	Need extra cost to adjust the parameters of the algorithm.
Servant et al., [87]	Location information (LI), Change history (CH) and Expertise mapping.	To improve the accuracy of bug assignment	AspectJ	Accuracy	Proposed expertize technique achieves greater accuracy rate	Additional cost to locate the given bugs
Xuan et al. [88]	NB , SVM and social network analysis	To address the developer assignment and assignee task	Mozilla Eclipse	Accuracy	Social network analysis improves the average accuracy of NB and SVM	Needs extra cost for analyzing the developer relationships
Xie et al., [89]	DRETOM based on topic model	To improve the results of existing assignment models	Eclipse JDT Mozilla Firefox	Recall	DRETOM outperforms than Bugzie and DREX model	Required to set the parameters of topic based model.

Bhattacharya et al., [90]	NB, SVM, C4.5 and BN classifiers	To reduce the tossing length	Mozilla Eclipse	Accuracy	NB classifier outperformed the other classifiers	Needs to extract the various feature 'component' and 'Product"
Xia et al., [91]	DevRec based on bug report-based analysis (BRA) and develop-based analysis (DBA)	To determine the relevant developers	Mozilla, NetBeans Eclipse	Recall	DREX model improves the recall rate than DREX and Bugzie models	Required more features for bug report characterization
Naguib et al., [92]	LDA, topic model algorithms	To assign the appropriate developers	ATLAS Reconstructi on UNICASE Eclipse BIRT	Hit score	Proposed approach performs better than LDA-SVM based model	Need extra cost to adjust the parameters of the LDA algorithm.
Zhang et al., [93]	LDA, social network.	To improve the accuracy of bug assignment	Eclipse	Precision Recall, F-measure	Proposed approach provides higher F-measure rate than DRETOM and AP	Additional cost is required for adjusting the parameters and analyzing the social network
Yang et al., [94]	Used topic model approach, core NLP for preprocessing and Kullback-Leiber for finding the similar bug	To identify the relevant developer	Eclipse Mozilla NetBeans	Precision Recall, F-measure	Proposed system obtains better recommendati on results than Out-Degree , AP-based	In the case of incorrect word frequency matching, this approach was unsuccessful in finding the

	reports, after that used multi-factors and social network techniques for finding the relevant developer.				recommender and DRETOM	similar bug topics.
Zhang et al., [95]	KSAP based on heterogeneous bug repository network and K-NN search	To improve the accuracy of bug assignment	Eclipse Mozilla Apache Tomcat6	Precision Recall, F-measure	KSAP approach is compared with existing ML-KNN, DREX, DRETOM, Bugzie, DevRec and DP models	Needs to extract the various feature 'component' 'severity' and 'Product"
Lui et al., [96]	DRS based on modern portfolio and LDA	To avoid cold start problem in bug assignment	Bugzilla	Average fix time	DRS can reduce the cold start problem of bug assignment considerably through assign the relevant developers	Only calculate the average time to fix the bugs.
Yadav et al., [97]	Worked on developer expertise scores. Used Jaccard, cosine –similarity to measure the similarity between fixing time, priority and versatility, based	To determine the appropriate developer for bug fixing	Freedesktop Eclipse Firefox Mozilla Netbeans	Precision Recall, F-measure	Proposed approach provides significant results than existing machine learning approaches	Ignore the severity of bug reports

	on extracted features the developers are ranked					
Kanwal et al. [98]	NB and SVM classifiers	To resolve the important bugs	Eclipse	Nearest False Negatives, Precision Recall , Accuracy and Nearest False Positives	SVM achieved the highest accuracy with both text and categorical features as compared to NB classifier	Ignore semantic relations of text

CHAPTER 3

PSO- ACO BASED BUG REPORT SUMMARIZATION MODEL

3.1 INTRODUCTION

This chapter addresses the summarization issue of bug reports. The aim of bug report summarization is to generate short summary of lengthy bug reports [21]. In literature, both of supervised and unsupervised approaches have been presented for summarizing the bug reports [24,25,31,32]. In supervised approaches, manual summaries of bug reports are prepared and relevant sentences are extracted from these summaries. Further, a model is designed to predict the relevant sentences using extracted sentences. Whenever, a new bug report is reported, the relevant sentences are extracted using trained model. The shortcomings associated with supervised learning approaches is to require large amount of training data and partial towards the specific bug reports [29]. Furthermore, manual summaries generation is time consuming, costly and require a lot of human efforts. Whereas, unsupervised approaches determine the optimal sentences from summary based on diversity and centrality measures. These approaches are more suitable for different types of bug reports without any complex changes [32]. It is also noticed that, these approaches are not required manual summaries for summarization task. In turn, cost and human effort is reduced. For summarization task, summaries are divided into two categories- extractive and abstractive [99]. The extractive summary refers to the identification of relevant sentences from the bug report to produce the final summary. While, in abstractive summary, the appropriate sentences are selected and rephrased to generate a new short sentences. These sentences contain critical information from the original sentences. It is observed that the unsupervised approaches provide better results than supervised approaches for bug report summarization [31,32,35,36]. It is also noticed that most of automatic summarization techniques are extractive in nature. It is observed that main issue with bug report summarization is volume of data associated with bug reports. The data is presented in the form of comments and large search space. In turn, it is difficult to determine the relevant sentences and compute the sentence score [32]. Several other problems are also associated with bug reports such as sparsity issue in high dimensional matrix, time cost, accuracy and overfitting issue [35,36]. Hence, the main objective of this chapter is to address the following issues.

- To develop a generalized bug report summarization technique for summarization task

- To determine relevant summary subset of bug reports
- To address the data sparsity of sentences.

This chapter presents an unsupervised approach based on particle swarm optimization (PSO) and ant colony optimization (ACO) approaches to select the optimal summary subset [100-104]. Initially, PSO is applied to determine the relevant summary subset. Further, ACO algorithm is applied to refine the output of PSO algorithm. The semantic relationship of sentences is also computed to handle data sparsity issue. Moreover, two research questions are also designed to show the effectiveness of the proposed PSO-ACO model.

RQ1: Does the performance of the proposed PSO-ACO model is better than PSO model?

RQ2: Are the unsupervised techniques more feasible than supervised techniques for bug report summarization?

3.2 Formulation of Bug Report Summarization as Optimization Problem

The optimization of summarization process is a difficult task due to low order value optimization. It influences the speed and accuracy of summarization process, but large search space is explored for better summarization [101,105-107]. Bug reports consist of large number of comments and descriptions. These comments and descriptions contains valuable information for summarization task. To extract the relevant sentences for short summaries, a PSO-ACO based summarization technique is proposed. Further, the semantic relationship is also determined for extractive bug report summarization (BRS). When, bug reports are processed, a weight function is assigned to each normalized text. The subset of normalized texts is generated according the user preferences. A user preference summarization model contains a large number of subsets. Suppose, the number of lines present in a Bug report is x. The number of possible subsets with the combination of 1, 2 and n sentences of a bug report is r. So, the number of summary subsets with different number of sentences is computed using equation 3.1.

$$A = \sum_{r=1}^{n} C_r^x \tag{3.1}$$

Assume, summary percentage mentioned by user is z%. Hence, the number of sentences in subset based on user percentage is computed through equation 3.2.

$$N = x \times \frac{z}{100} \quad (3.2)$$

$$A \cap N = B \quad (3.3)$$

In equation 3.3, A represents the number of subset with different number of sentences, N denotes number of sentences in subset based on user percentage and B contains the common summary subsets presented in both of A and N. The optimization process is implemented on the summary subsets presented in set B and the aim is to choose minimum summary subsets that can cover entire summary. Hence, the bug report summarization problem is formulated as subset selection problem.

3.3 Proposed PSO-ACO Based Summarization Model

This section presents a PSO-ACO based summarization model to determine the optimal subset of summaries. The working of proposed model is demonstrated using Figure 3.1. The proposed model consists of four phases. These phases are preprocessing, sentence scoring, summary subset selection and performance evaluation.

3.3.1 Preprocessing Phase

This phase consists of bug report descriptions, titles and summary percentage in the form of text. The text of bug reports (D) is divided into set of passages P_i, $D = (P_1, P_2 \ldots P_N)$. These passages are further divided into many sentences, $P_i = S_i$, and sentences are further segmented into words W_i, $S_i = (W_1, W_2 \ldots\ldots W_N)$ and this process is known as tokenization. The outcome of tokenization is isolated words, called token. Moreover, stop word removal (SWR) and rooting extraction (RE) techniques are also applied on the output of tokenization process. SWR technique removes the unwanted and insignificant words. Whereas, RE technique is applied for linguistic normalization. It can be described as change of inflectional and derived words into familiar base words [108-110]. The output of this phase is normalized text and can be acted as the input of second phase.

3.3.2 Sentence Scoring Phase

The output of preprocessing phase is tokens i.e. words and features. The score of each words are computed using extended frequency criteria. These criteria are illustrated in Figure 3.2.

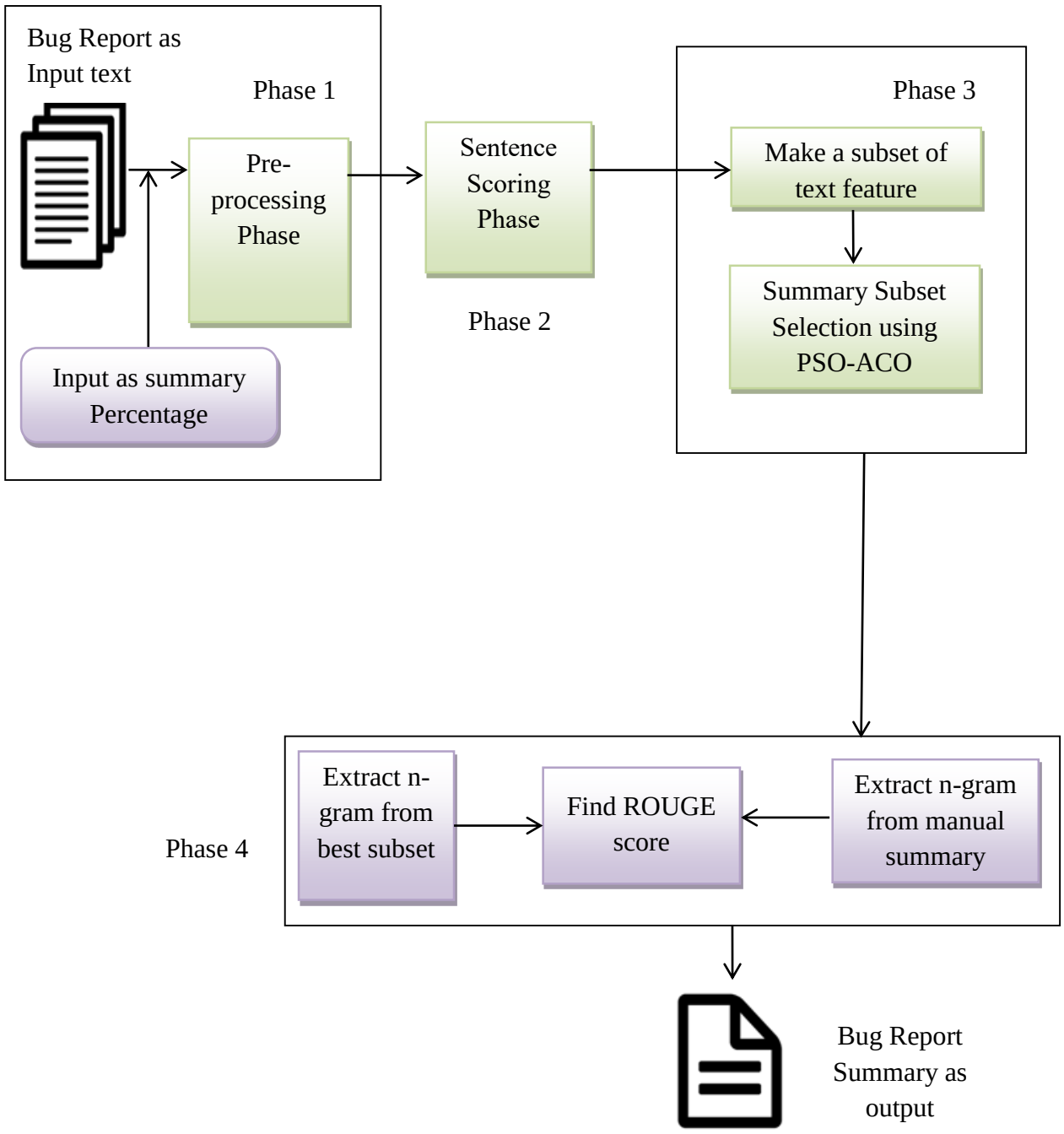


Figure 3.1: Proposed PSO-ACO based Bug Report Summarization Model

- Informativeness: This criterion determines the specificity of system. It can be described as amount of data reflected in a given document (D) using term (t).

- Phraseness: This criterion correspondence to extract the multiple words. The phraseness score describes the rigidity of words in multi-word arrangement. This criterion computes the significance of a term based on relative frequencies of terms and their segments i.e. unigrams or frequencies.

Furthermore, previous study demonstrates that Kullback–Leibler divergence (KLdiv) method provides better results using both of criterias [111]. The basic aim of KLdiv method is to compute the divergence between two probability distributions [112]. It also reduces the sparsity of text through semantic information of bug report summary. Further, KLdiv method is divided into Kullback–Leibler divergence Informativeness (KLI) and Kullback–Leibler divergence Phraseness (KLP). The probability of each unigram, bigram and trigram features are computed using equations 3.4 -3.6 and stored in a matrix.

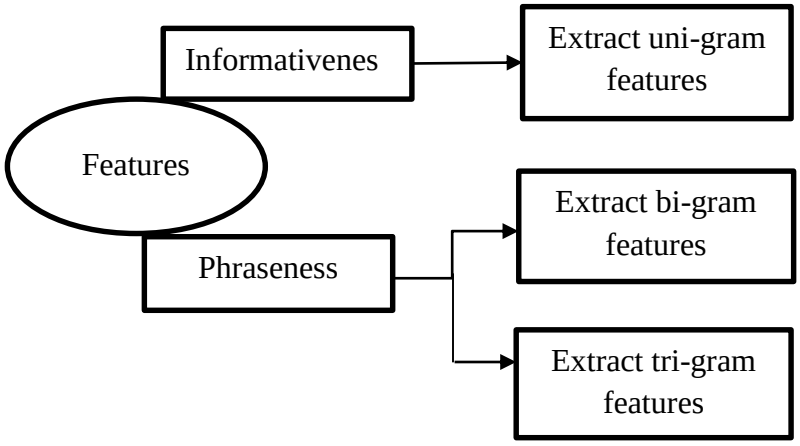


Figure 3.2: Feature Weighting Module

$$\mathrm{KLIP(w)} = \mathrm{KLI(w)} + \mathrm{KLP(w)} \tag{3.4}$$

$$\mathrm{KLI(w)} = \mathrm{p(w|S)log}\frac{\mathrm{p(w|S)}}{\mathrm{p(w|C)}} \quad \text{// For unigram} \tag{3.5}$$

$$\mathrm{KLP(w)} = \mathrm{p(mw|S)log}\frac{\mathrm{p(mw|S)}}{\prod_{r=1}^{m}\mathrm{p}(\mu_r|S)} \quad \text{// For bigram and trigram} \tag{3.6}$$

In equations 3.4-3.6, p(w|S) is the probability of word ‘w’ in sentence ‘S’, p(w|C) is the collection of language , p(mw|S) is the probability of multiword in S and $\mathrm{p}(\mu_r|S)$ is the probability of r^{th} unigram inside n-gram (mw).

The score of all words are combined to obtain the sentence scores (SS). Further, the subsets of these sentences are evaluated using user provided summary percentage. The subset score (SuS) is computed for every subset. Here, the PSO-ACO technique is applied to determine the optimal subsets from the given summary subsets based on subset score. Each subset is represented using vector in a search space and described as $S_l(l = 1,2,3 \ldots . m)$. The steps of the preprocessing and sentences scoring phases are mentioned in Algorithm 3.1.

Algorithm 3.1

Input: Bug reports as text input, D

Output: Summary subsets (SuS) with the respective score, $SuS_l = \{S_1, S_2, \ldots \ldots . . S_m\}$, $m = B$ by using equations 3.1-3.2.

1. PRE-PROCESSING PHASE

1.1. Initially, pre-processing of a bug report is executed.

1.2. Tokenization(D_i)

1.3. Stop word removal(D_i)

1.4. Rooting(D_i)

2. SENTENCE SCORING PHASE

2.1. Extract informative features (unigram, KLI) by using equation 3.5

2.2. Extract phraseness features (bigram and trigram, KLP) by using equation 3.6

2.3. Combination of two scores KLI and KLP using equation 3.4.

2.4. KLIP score of all words are added to get the sentence scores (SS).

2.5. Generate the subset according to user summary percentage.

2.6. Calculate the summary subset score (SuS) for every subset.

The output of the second phase is sentences scores. These scores are given as input to third phase.

3.3.3 Summary Subset Selection Phase

In this phase, PSO-ACO technique is used to determine the optimal summary subsets. The steps of PSO-ACO technique is described in Algorithm 3.2. Figure 3.3 illustrates the flow chart of the proposed PSO-ACO technique. The algorithm starts with initialization of the population of PSO algorithm in terms of sentence score and other user defined parameters like, maximum number of iteration, β_{p1}, β_{p2}, and $\alpha(t)$.

Algorithm 3.2: Proposed PSO-ACO Based Subset Selection Algorithm

Input: Summary subset (m) with respective score

Output: Summary subset (n) with best optimized score, whereas n<m

Step 1: Initialize the particles and user define parameters viz population, iteration etc

Step 2: For, each subset or particle, d= 1,2…N

Compute fitness function and Particles initial position= subset score

Step 3: Subset best known position = initial position: $\text{pbest}_d \leftarrow \text{Pos}_d^{(t)}$

Step 4: Assume, initial velocity is zero

Step 5: While (condition is not met), do following:

Step 6: Update the Particle's velocity using equation 3.7

$$\text{Vel}_d^{(t+1)} = \alpha(t)\text{Vel}_d^t + \beta_{p1}\text{rand}_{p1}(t)\left(\text{pbest}_d - \text{Pos}_d(t)\right) + \beta_{p2}\text{rand}_{p2}(t)\left(\text{gbest}_d - \text{Pos}_d(t)\right) \quad (3.7)$$

Step 7: Update the Particle's position using equation 2

$$\text{Pos}_d^{(t+1)} = \text{Pos}_d^{(t)} + \text{Vel}_d^{(t+1)} \qquad (d = 1,2 \ldots\ldots N) \quad (3.8)$$

Step 8: If $f(\text{Pos}_d^{(t)}) < f\left(\text{Pos}_d^{(t+1)}\right)$

$\text{Pos}_d^{(t+1)} \leftarrow \text{Pos}_d(t)$

Step 9: If $f(\text{Pos}_d^{(t+1)}) < f(\text{pbest}_d)$

$\text{pbest}_{jd} \leftarrow \text{Pos}_d^{(t+1)}$

Step 10: If $f(\text{pbest}_d) < f(\text{gbest}_d)$

$\text{gbest}_d \leftarrow \text{pbest}_d$

End while

End for

Step 11: Sort the particles according the gbest and Calculate ROUGE score

Step 12: Initialize the ants and other user defined parameters of ACO algorithms

Step 13: Initialize the pheromones using equation 3.9

$$p_A(T) = \frac{(\tau_A(T))^{\alpha}.\eta_A^{\beta}}{\sum_A(\tau_A(T))^{\alpha}.\eta_A^{\beta}} \quad (3.9)$$

Step 14: Update the pheromones using equation 3.10

$$\tau_A(T+1) = \rho\tau_A(T) + \Delta\tau_A(T) \quad (3.10)$$

Step 15: If (value of subset is not changed)

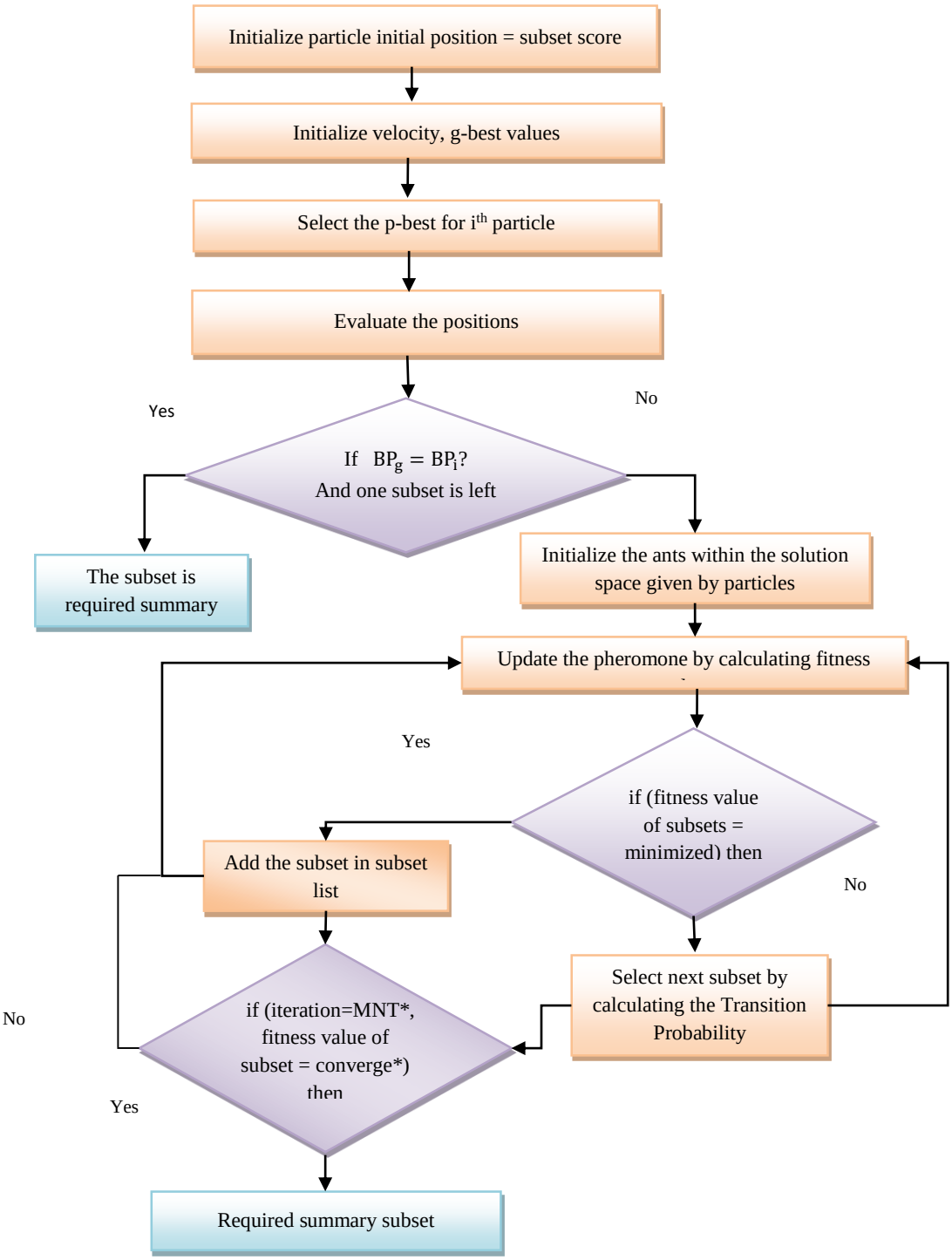
Add the subset in subset list and calculate the transition probability

Select the subset with higher probability and calculate ROUGE score

Step 16: Else, update the pheromones of all subsets and go to step 13

Step 17: Obtain the final summary subset with best optimized score

*Vel_{id}^t is the old velocity, $\text{Vel}_d^{(t+1)}$ is the new velocity, $\text{Pos}_d^{(t)}$ represent as current position , $\text{Pos}_d^{(t+1)}$is updated particle position , gbest_d is the global best position , pbest_d is the personal best position, β_{p1}, β_{p2} are the learning factors as positive constant , $\text{rand}_{p1}(t), \text{rand}_{p2}(t)$ are random numbers between [0,1] and $\alpha(t)$ is the inertia weight . η_β describe the background information of features to improve the results, $\tau_A(T)$ is pheromone amount for the Ath feature in time T, α and β are the control parameters that provides the pheromone and background information, and $p_A(T)$ is the Transition probability , ρ is defined as evaporation rate of pheromone trail and lies between [0, 1], $\Delta\tau_A(T)$ is the pheromone trail amount added to Ath feature between time Δ and ΔT



MNT* = Maximum Number of Iteration
Converge* = Not able to get different results in fixed number of iteration.

Figure 3.3: Flowchart of PSO-ACO based Summary Subset Selection Algorithm

The particles are described in terms of subset score assume initial velocity of particles is zero. Next step is to, calculate the cosine similarity of subsets as fitness function. Until the termination condition is not met, update the velocity and position of particles using equations 3.7 and 3.8. At the end of every iteration, compare the particle's current position with its personal best. If personal best position is better than update the current position. The personal best position of all particles is compared with global best position. If personal best position of any particle is better than global best position than update its global best position. This process continues on until the personal best position is equal to global best position of particle. After that, sort the subsets according to its global best position. The output of PSO algorithm is given as input to ACO algorithm. Initialize the user define parameters of ACO algorithm. Initialize the pheromones and update its value using equations 3.9 and 3.10. If the value of the subset does not change after the few iterations, then calculate its transition probability otherwise update the pheromones of all subsets. The output of algorithm is optimal summary subset.

3.3.4 Performance Evaluation Phase

This phase evaluates the performance of proposed model. The performance of model is evaluated using ROUGE score parameter [113]. The system generated summary is compared with the manual summary to determine the efficiency of PSO-ACO technique. Therefore, all possible n-grams words (w) are extracted from system generated and manual summaries subsets using equations 3.11-3.13.

$$\mathrm{P}\left(\mathrm{w_i}|\mathrm{w_0} \ldots \ldots \mathrm{w_i} - 1\right) = \mathrm{P}\left(\mathrm{w_i}\right) \tag{3.11}$$

$$\mathrm{P}\left(w_i|w_0 \ldots \ldots w_i - 1\right) = \mathrm{P}\left(w_i|w_{i-1}\right) \tag{3.12}$$

$$\mathrm{P}\left(\mathrm{w_i}|\mathrm{w_0} \ldots \ldots \mathrm{w_i} - 1\right) = \mathrm{P}\left(\mathrm{w_i}|\mathrm{w_{i-1}w_{i-2}}\right) \tag{3.13}$$

The n- gram represents the combination of n continuous words into a single word. In this experiment unigram, bigram and trigram are considered. Moreover, common occurrence of n-grams is computed between system generated and manual summaries.

3.4 Experimental Results and Discussion

This section describes the experimental results of proposed PSO-ACO model. The proposed model is implemented using Java platform. The system is equipped with an Intel Core i5 (7th generation),

8GB of DDR4 memory and NVIDIA GEFORCE GPU, and Windows 10 operating system with 64- bit and CPU @ 2.70GHz. The performance of proposed model is evaluated using Rastkar dataset [24,25]. This dataset contains thirty-six bug reports from four different projects. These projects are KDE, Eclipse Platform, Gnome and Mozilla. These thirty six bug reports are divided into three summary sets. The ROUGH score parameter is described using equation 3.14.

$$\text{ROUGE} = \frac{\sum_{C=RSS} \sum_{gram_n \in C} Count_{match}(gram_n)}{\sum_{C=RSS} \sum_{gram_n \in C} Count(gram_n)} \quad (3.14)$$

In equation 3.14, RSS is the reference summary set, $Count_{match}(gram_n)$ is the maximum no. of common n-grams between system generated and reference summaries. $Count\,(gram_n)$ is the number of n-grams in the reference summary. The user defined parameters setting of proposed PSO-ACO model is presented in Table 3.1.

3.4.1 RESULTS

This subsection presents the experimental results of proposed PSO-ACO model. The system-generated summaries are compared with three manual summaries of each bug report. These summaries are denoted as Summary Set 1, Summary Set 2 and Summary Set 3. In this work, thirty percent of summary is considered to conduct the experiments. The performance of PSO-ACO model is compared with well-known supervised and unsupervised models such as particle swarm optimization (PSO), grasshopper approach (EGA), bug report classifier (BRC), email classifier (EC) and email meeting classifier (EMC) [24,25]. The PSO and grasshopper models are unsupervised in nature, whereas rest of are supervised in nature [114, 31]. The basic aim of this work is to explore the problem space efficiently and analyze the trigram and bigram more effectively. The ROUGE score parameter is used to evaluate the experimental results. It is also mentioned that both of PSO-ACO and PSO based summarization models are developed in this work.

3.4.1.1 Summary Set 1

The summary set 1 consists of the summaries generated through software company expert for thirty-six bug reports. These summaries are compared with system generated summaries using ROUGE score parameter. The system generated summaries are computed using proposed PSO-ACO model. The performance of the proposed PSO-ACO model is compared with several other

models such as PSO, EGA, BRC, EC and EMC [25,31]. The experimental results of proposed PSO-ACO model and other models are presented in Table 3.2. It is seen that proposed PSO-ACO model provides higher ROUGE score rate as compared to other models. It is also noticed that EGA exhibits worst performance among all models. The average ROUGE score obtained through proposed PSO-ACO model is 92.60. It is also revealed that PSO based summarization model gives better results than BRC, EC, EMC and EGA models. But, the integration of PSO-ACO techniques improves the results of PSO technique. Figure 3.4 demonstrates the graphical representation of simulation results of the proposed PSO-ACO model and other models being compared using Summary Set 1. It is stated that proposed PSO-ACO model achieves better quality results than other models.

Table 3.1: PSO-ACO parameters setting

Parameters of PSO-ACO	Values
Particles	100
Iterations of Particles	500
C_1	1.9
W	0.4-0.9
C_2	1.9
V_{MAX}	0.1
ANTS	100
ρ	0.5
Iterations of Ants	500
α	0.7-1
β	2-5
Initial Value of Pheromone	AVG(GLOBAL BEST)
Pheromone Intensity	AVG(LOCAL BEST)

3.4.1.2 Summary Set

The summary set 2 contains the summaries generated by technical expert. Summary set 2 is considerably different from summary set 1. The system generated summaries are compared with summaries generated through technical expert. Table 3.3 illustrates the simulation results of proposed PSO-ACO model and other models using Summary Set 2. ROUGE score parameter is used as performance measure to validate the results of PSO-ACO based summarization model

Table 3.2: Comparison of proposed PSO-ACO, PSO, BRC, EC, EMC and EGA techniques using ROUGE score parameter for Summary Set 1

Sr. No.	Proposed PSO-ACO	PSO	BRC	EC	EMC	EGA
1	79.03	77.9	92	79.9	85	74.6
2	88.54	87.4	88.4	86.5	87.6	77.2
3	93.33	92.8	84.6	90.9	89.1	78.1
4	92.16	91	74.5	86.5	86	77.7
5	88.85	88.2	75.3	84.8	90.3	78
6	86.23	85	70.6	87.5	89.4	79.1
7	96.77	96	86.3	91.6	90.5	80.2
8	96.6	93.5	91.9	89.2	93.9	79.8
9	90.33	90.2	92.3	89.5	90.7	79.1
10	96.08	93.4	90.9	91.8	92.6	79.8
11	96.49	95.6	87.3	91.7	91.6	79.8
12	93.61	92.6	91.4	91.1	87.1	79.4
13	94.44	93	93.8	88.1	91.9	79.7
14	91.41	88.9	92.1	83.4	86.5	79.5
15	96.55	95.2	90.4	92.9	93.4	80.1
16	93.98	93	92.7	92.3	92.8	80
17	94.12	93.4	91.5	92	91.1	80.1
18	95.57	94.7	90.5	90.9	91.4	80.7
19	92.29	91.5	82.6	90.4	91.9	80.1
20	91.97	92.9	89.3	92.7	90.9	79.5
21	95.89	94.4	91.5	90.7	94	79.6
22	94.33	93.7	89.5	91.9	89.8	79.1
23	98.49	98	81.8	94.5	91.7	78.7
24	91.53	90.4	91.7	88.2	68.4	78
25	90.55	87.2	90	84.6	87.5	77.7
26	94.03	93.2	89.9	61.5	69.1	78.4
27	93.88	91.4	80.6	67.5	90.4	78.2
28	91.47	90.6	89.3	91	88	78.2
29	94.16	93.3	86.4	66.9	89.5	78.8
30	90.05	88.5	81.4	85.5	87.4	78
31	93.26	91.9	83	89.1	89.9	77.3
32	92.33	91.8	82.8	89.8	91.6	64.1
33	93.66	91.3	88.1	89.5	88	51
34	96.12	95.1	87.6	60.9	64.1	38
35	88.86	87.7	82	64.6	43.3	35.7
36	86.53	83.1	76.4	49.3	43.3	35.8
Average	**92.6**	**91.3**	**86.7**	**84.7**	**85.5**	**74.2**

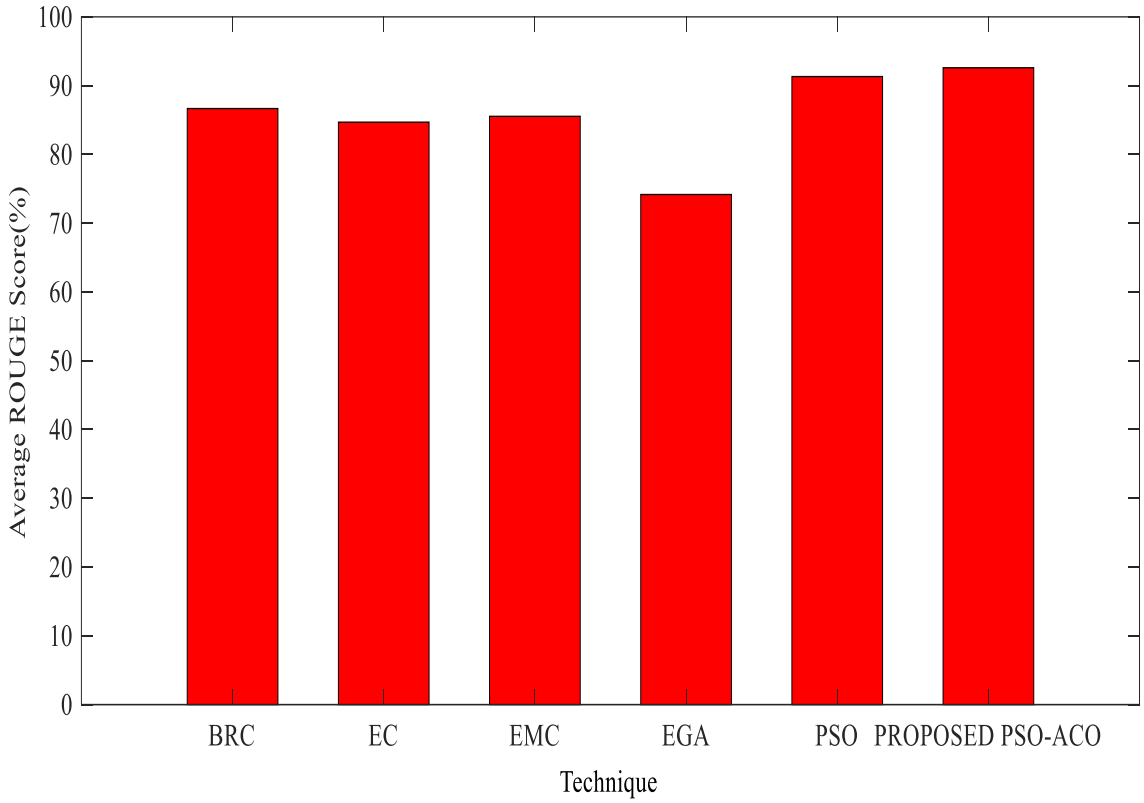


Figure 3.4: Simulation results of proposed PSO-ACO, PSO, BRC, EC, ECM and EGA techniques on Summary Set 1

It is revealed that the proposed PSO-ACO based summarization model achieves higher ROUGE score in comparison to other models being compared. The ROUGE score rate of proposed PSO-ACO model is 87.92. It is also observed that EC model exhibits worst performance among all other models. But, it is seen that there is no significant difference between the performance of PSO- ACO and PSO based summarization models. But, the performance of the PSO algorithm is slightly improved using integration of PSO-ACO. Figure 3.5 demonstrate the experimental results of proposed PSO-ACO model and other model using Summary Set 2. ROUGE score parameter is used to illustrate the simulation results of different models. It is revealed that there is significant difference between the performances of proposed PSO-ACO model and other models like BRC, EC, EMC, EGA. But, it is observed that the performance of PSO and PSO-ACO based summarization models is almost similar.

3.4.1.3 Summary Set

The summary set 3 contains the summaries described through an English exper

Table 3.3: Comparison of proposed PSO-ACO, PSO, BRC, EC, EMC and EC techniques using ROUGE score parameter for Summary Set 2

Sr. No.	Proposed PSO-ACO	PSO	BRC	EC	EMC	EGA
1	75.68	74.58	70.76	52.07	64.27	71.75
2	82.81	81.62	85.25	57.65	63.48	74.28
3	92.12	91.33	81.65	59.83	64.88	75.1
4	89.4	88.16	68.12	57.38	63.95	74.64
5	85.05	83.98	73.23	55.53	66.06	74.81
6	83.81	82.6	60.78	58.28	66.83	75.74
7	93.06	92.25	83.26	59.88	66.6	76.85
8	92.23	87.39	88.32	58.51	68.34	76.07
9	88.72	88.15	90.14	59.15	66.96	73.75
10	91.81	89.3	86.65	59.94	68.16	74.26
11	91.4	90.52	80.59	60.29	65.7	74.51
12	91.26	90.34	88.44	59.04	58.42	74.46
13	87.83	86.79	91.25	52.67	68.2	74.35
14	73.73	71.21	89.07	54.3	64.33	74.56
15	93.1	91.68	85.69	60.91	68.26	77.53
16	92.08	91.06	72.32	60.4	67.25	77.29
17	91.02	90.14	88.98	59.92	66.98	76.62
18	90.22	89.62	86.98	59.3	67.93	77.44
19	89.11	88.27	92.95	60.08	66.48	77.18
20	91.2	91.97	84.84	60.66	65.62	75.53
21	91.49	90.02	89.81	58.8	70.79	62.4
22	86.87	86.38	84.39	60.75	65.93	61.63
23	96.27	95.87	90.85	61.21	64.14	62.13
24	88.86	87.75	86.76	54.84	22.22	61.12
25	82.5	76.78	88.36	25.59	63.43	59.87
26	85.67	84.78	83.49	28.22	45.2	61.6
27	87.76	84.66	83.65	58.18	64.51	74.35
28	90.76	89.87	88	59.55	62.18	74.76
29	89.92	88.79	83.03	55.93	66.28	74.76
30	80.39	78.99	63.33	55.98	64.26	74.3
31	90.29	88.94	81	59.39	65.33	72.57
32	89.81	89.24	81.82	58.92	45.19	59.87
33	87.78	87.5	73.93	59.13	64.57	47.12
34	90.96	89.89	79.92	58.48	58.91	34.62
35	86.39	85.56	74	50.82	21.6	32.59
36	73.63	66.91	72.7	22.3	43.23	32.2
Average	**87.92**	**86.47**	**82.06**	**55.39**	**61.29**	**68.13**

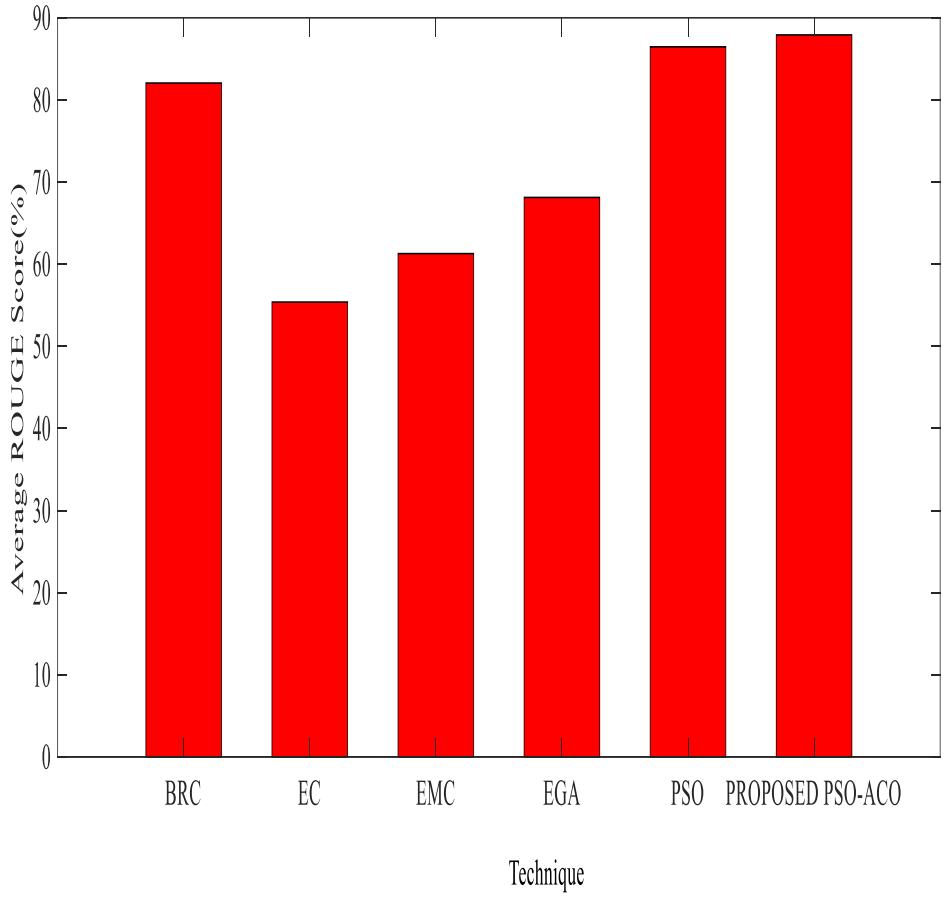


Figure 3.5: Simulation results of proposed PSO-ACO, PSO, BRC, EC, ECM and EGA techniques on Summary Set 2

The simulation results of the system generated summaries are also compared with summary set 3. Table 3.4 depicts the simulation results of proposed PSO-ACO model and other models such as BRC, EC, EMC, EGA, PSO models. From simulation results, it is seen that proposed PSO-ACO model achieves better quality results than PSO, BRC, EC, EMC, EGA models. The proposed PSO-ACO model obtains 84.51 ROUGE score rate. It is revealed that EMC models provides worst results among all other classifiers for summary set 3. It is also stated that the significant difference is occurred between the performance of proposed PSO-ACO and PSO based summarization models. Figure 3.6 depicts the graphical representation of simulation results of proposed PSO-ACO model and PSO, BRC, EC, EMC, EGA models using Summary Set 3. It is seen that the average ROUGH score of the proposed PSO-ACO model is better than other compared models. Hence, it is stated that PSO-ACO based summarization model is capable to generate the short summaries without affecting the structure of bug report

Table 3.4: Comparison of proposed PSO-ACO, PSO, BRC, EC, EMC and EC techniques using ROUGE score parameter for Summary Set 3

Sr. No.	Proposed PSO-ACO	PSO	BRC	EC	EMC	EGA
1	75.47	74.37	43.89	51.2	41.19	69.29
2	80.42	79.23	54.38	55.98	40.54	70.85
3	89.29	88.7	50.24	56.29	42.91	71.52
4	81.72	80.16	62.96	53.78	41.08	70.98
5	82.33	81.17	66.63	54.19	42.11	71.87
6	82.6	81.4	57.67	55.57	43.47	72.68
7	86.12	85.32	63.64	56.41	42.68	73.59
8	91.26	83.91	54.88	56.28	45.15	73.18
9	84.61	84.94	65.47	57.17	43.08	70.64
10	89.32	86.41	74.77	57.75	44.72	71.43
11	87.71	86.84	60.8	58.19	42.83	72
12	89.57	87.72	75.25	56.73	39.56	72.15
13	83.59	82.47	67.37	49.54	43.87	71.99
14	68.68	66.16	75.37	52.19	40.04	72.63
15	91.87	90.42	72.9	60.29	45.193	76.13
16	91.49	90.45	67.55	59.44	44.55	75.2
17	88.9	87.88	64.75	58.16	44.17	74.27
18	86.71	86.6	75.08	57.84	44.25	75.28
19	87.77	86.93	62.95	59.21	43.28	75.32
20	90.27	90.69	54.59	58.19	43.67	73.4
21	85.33	83.87	74.57	55.94	45.18	60.45
22	84.39	83.96	62.92	59.64	43.09	59.98
23	95.38	94.97	76.65	60.61	43.29	60.72
24	87.97	86.85	66.69	53.45	21.99	58.44
25	77.77	73.51	74.83	24.5	40.31	56.91
26	78.9	76.83	70.29	26.86	22.51	58.35
27	83.47	80.57	65.51	56.57	40.96	70.67
28	90.05	89.16	60	56.05	41.9	70.91
29	80.37	78.99	67.61	51.73	41.32	58.17
30	77.55	76.19	77.86	53.25	41.1	58.74
31	84.9	83.55	71.27	56.61	42.04	57.1
32	86.84	86.27	77.58	56.16	21.71	45.17
33	83.25	82.21	75.05	27.4	41.84	32.84
34	81.22	80.45	71.11	27.67	17.77	21.1
35	84.12	83.01	74.66	49.24	21.03	31.88
36	71.06	64.7	69.5	21.57	20.54	31.51
Average	**84.51**	**82.97**	**66.86**	**51.71**	**39.03**	**63.54**

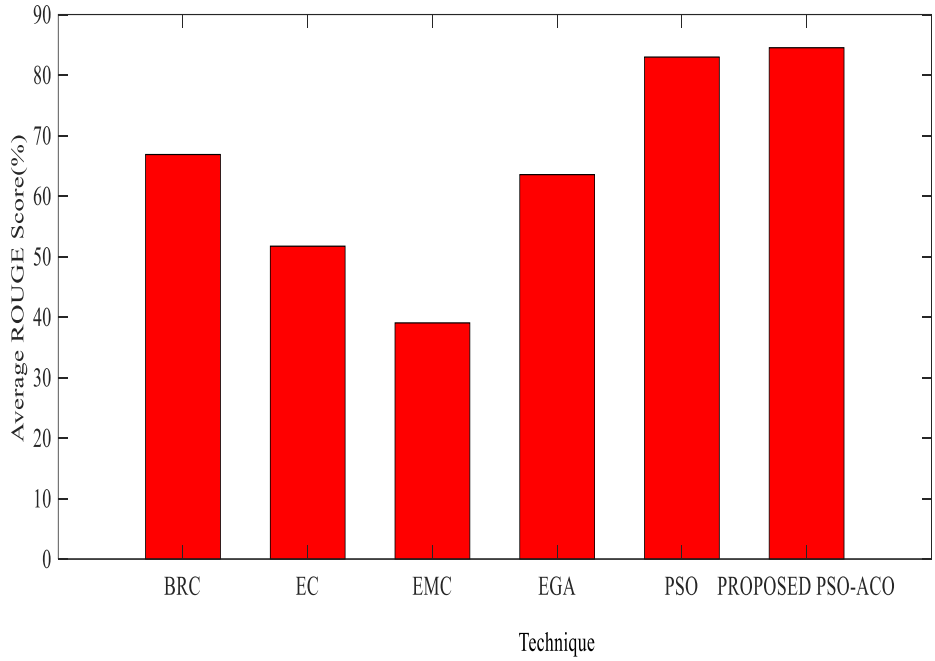


Figure 3.6: Simulation results of proposed PSO-ACO, PSO, BRC, EC, ECM and EGA techniques on Summary Set 3

The average results of three summary sets using all techniques are reported in Table 3.5. It is observed that proposed PSO-ACO technique provides better results than other techniques. Whereas, EC techniques exhibits worst performance among all techniques. Hence, it is said that proposed PSO-ACO is one of the efficient and effective model for bug report summarization. Furthermore, two fundamental research questions are also designed to differentiate the performance of proposed PSO-ACO model and other existing models.

Table 3.5: Average Rough score of proposed PSO-ACO, PSO, BRC, EC, EMC and EC techniques

Dataset	Techniques					
	Proposed PSO-ACO	PSO	BRC	EC	EMC	EGA
Summary Set 1	92.6	91.3	86.7	84.7	85.5	74.2
Summary Set 2	87.92	86.47	82.06	55.39	61.29	68.13
Summary Set 3	84.51	82.97	66.86	51.71	39.03	63.54

3.4.1.4 Statistical Test

This subsection deals with the statistical analysis of the simulation studies. The objective of the statistical analysis is to find the significant differences between the performance of proposed

technique and other techniques being compared. To validate the existence of the proposed technique and prove its significance, some statistical tests are applied on the experiment results. These tests have been widely applied in the machine learning domain [133]. A statistical test is applied to check the substantial differences between the performances of techniques. In this study, Friedman and Wilcoxon tests are employed for demonstrating the statistical analysis [134-135]. The value of α (level of confidence) is set to 0.1. The results of Friedman test are illustrated in Tables 3.6-3.7. Table 3.6 summarizes the average ranking of each technique computed through Friedman test using average ROUGH score parameter. The critical value of Friedman test at the confidence levels 0.1 is 11.070504. The corresponding p value and statistics result are reported in Table 3.7. It is clearly seen that null hypothesis is rejected at the confidence level 0.1 and significant differences are occurred between the performances of all techniques. From the literature, it is noted that Friedman test cannot distinguish the datasets which are used for comparison as it assigns equal importance to each dataset [134]. In this experiment, the p−values of Wilcoxon's rank sum test are also conducted. Wilcoxon's rank sum test is a nonparametric statistical test. It is more sensitive than Friedman test as it assumes proportionality of differences between two pairs samples. Moreover, it is safer than Friedman test as it does not assume the normal distributions and the outliers affecting lesson. The aim of this test is to indicate the proposed PSO-ACO technique provides notable improvement as compared with other bug report summarization techniques. The result of this test is illustrated in Table 3.8. The p-values of proposed PSO-ACO in comparison to other techniques are reported in Table 3.8. It is observed that all p-values are less than 0.1. Hence, it is stated that proposed PSO-ACO technique is statistical better than other compared techniques.

Table 3.6: Average ranking of techniques using Friedman tests

Techniques	Proposed PSO-ACO	PSO	BRC	EC	EMC	EGA
Ranking	1.00	2.00	3.33	5.33	5.00	4.33

Table 3.7: Results of Friedman test based on avg. ROUGH score parameter

Method	Statistical value	p value	Critical Value	Hypothesis
Friedman	13.47619	0.019302	11.070504	Rejected

Table 3.8: P−values of Wilcoxon rank sum test (pairwise method)

Technique 1	Technique 2	P −value
Proposed PSO-ACO	PSO	0.09246
	BRC	0.05628
	EC	0.01317
	EMC	0.007819
	EGA	0.002691

RQ1: Does the performance of proposed PSO-ACO model is better than PSO model?

Ans. RQ1): To address the summarization issues of bug reports, two models are developed in this work. These models are PSO based subset selection model and PSO-ACO based sub selection model for summarization task. The results of these models are presented in Tables 3.2-3.4 and Figures 3.4-3.6. It can be seen that Average ROUGE score of proposed PSO-ACO based subset selection approach is in between 92% to 85 %. While, the Average ROUGE score of PSO approach is between 91% to 83%. On the analysis of ROUGE score parameter, it is concluded that PSO-ACO based model gives better results than PSO model. The bug report summary subset evaluated using PSO-ACO are more effective and produced a less redundant summary. Moreover, problem space is fully explored using PSO-ACO based model and effectively analyzed bigram and trigram in the sentences. It is noted that PSO based summarization model sometimes stuck in local optima and converged. So, the feasible solution is not obtained every time through PSO model. To overcome the local optima problem of PSO model, it is integrated with ACO technique. The ACO technique optimizes the summary subsets produced by PSO model. The simulation results of PSO-ACO model also favor the above statement and it is concluded that incorporation of ACO technique with PSO model enhances the simulation results of PSO model considerably.

RQ2: Are the unsupervised techniques more feasible than supervised techniques for bug report summarization?

Ans. RQ2): In this work, two types of models i.e. supervised and unsupervised are considered to address the summarization issues.The PSO-ACO, PSO and EGA models are based on the

unsupervised techniques. Whereas, BRC, BC and EMC models consist of the supervised techniques. The simulation results of both models are illustrated in Tables 3.2-3.5. It can be said that the unsupervised techniques give better quality results than supervised techniques. The unsupervised techniques can be an effective tool for bug report summarization task and not required training data. On the other hand, supervised techniques require training data that can enhance time and cost factors. Hence, it can be stated that unsupervised techniques can be a feasible option for bug report summarization process and also reduce time and cost.

3.5 Summary

Bug report summarization is the part of the bug tracking system. In this chapter, PSO-ACO based bug summarization model is proposed for bug report summarization. The proposed model takes the advantage of KLIP approach. The aim of proposed model is to determine the optimum subset of summary. The efficiency of the proposed model is tested over Rastkar dataset and simulation results are compared with existing supervised and unsupervised models. The results showed that proposed PSO-ACO model provides better efficiency than PSO, Existing Grasshopper, Bug Report Classifier, Email Classifier and Email Meeting Classifier models. It is also observed that combination of PSO and ACO enhances the simulation results of PSO model in efficient manner.

CHAPTER 4
RFB METHOD FOR BUG SEVERITY CLASSIFICATION

4.1 Introduction

The primary task of bug severity classification model (BSCM) is to classify the data in different severity classes [52]. The BSCM consists of different machine learning techniques to predict severity classes of bugs. The severity classification can be formulated as a classification problem and bug dataset is developed using the content of bug reports [65]. To design the bug dataset, semantic and syntactical relationship between content of bug report is identified. Further, this relationship is used to determine the relevant features for severity classification. But, several issues are associated with feature extraction techniques especially in case of bug severity prediction [62-71]. These issues are highlighted as

- Sentences are considered as stack of words with unordered sequence of words
- Ignore the semantic relationship between words
- Interpretation of the slightly variant sentences

These issues affect the prediction rate of classification techniques and in turn, poor classification results can be obtained through classifiers. Furthermore, several researchers have been used bag of word and term frequency methods to determine the appropriate features for classification task [65-69].

Bug Report ID #: 63456 Title: Not able to upload an image on the Drupal website. Description: Drupal website functionality is not useful. I am not able to upload the image.	Bug Report ID #: 63457 Title: Multiple users can't access in Drupal website. Description: Drupal website functionality is not very good. The image uploading is too slow when concurrent users access the system.

Figure 4.1: Example of bug reports [6]

Figure 4.1 depicts the two bug reports with IDs 63456 and 63457. The bug ID 63456 describes the sentences as "Drupal website functionality is not good". Whereas, in bug ID 63457, it can be read as "Drupal website functionality is not very good". It is observed that both of sentences consist of same bag of words and classify these sentences in same class. But, bug ID 63456 is more critical than bug ID 63457. Because, images are not uploaded on the website and entire

image functionality is affected. In previous studies, n-gram method is also reported to determine the word order. This method also provides the same word order, but it can enhance the dimension and sparsity factors in data [59]. In turn, the computational cost of classification model is tremendously increased. It is also noticed that, feature extraction techniques cannot capture all latent features due to linear equation and fixed length of polynomial terms [59]. So, to address the above mentioned issues of feature selection techniques, a deep learning based framework is used in this chapter to determine the relevant features for prediction task [115]. Deep learning framework also improves the generality of model by considering different kernels with filter. Moreover, random forest with boosting method (RFB) is implemented to predict severity of bugs. The proposed RFB model is self-capable to learn feature representation without manual intervention of feature engineering. This model also explores the semantic and non-linear features mapping of bug reports. The simulation results of proposed RFB model is compared with existing model [65]. Further, two research question is also designed to validate the capability of proposed RFB model. These research questions are listed as

RQ 1: Can deep learning approach extract relevant features for automated BSCM?

RQ 2: Is integration of random forest and boosting method improve the performance of BSCM than traditional machine learning technique?

4.2 Proposed BSCM

The objective of the proposed BSCM is to predict different severity classes of bugs. The proposed model is based on the deep learning framework and RFB techniques. The deep learning framework is considered to identify more appropriate features for the prediction of severity classes. Whereas, RFB severs as classifier and predicts labels of severity classes. Figure 4.2 illustrates the working of proposed BSCM. It consists of five steps and these steps are Preprocessing, N-gram Extraction (NE), Feature Extraction with Deep Learning and Classification using RFB.

4.2.1 Preprocessing

The task of preprocessing step is to determine unwanted words from the bug reports and remove it. These words can degrade the learning performance of classification system. The preprocessing step reduces the feature space of bug reports. In turn, the effort cost of triage is

minimized [109]. The preprocessing step consists of three tasks. These tasks are summarized as

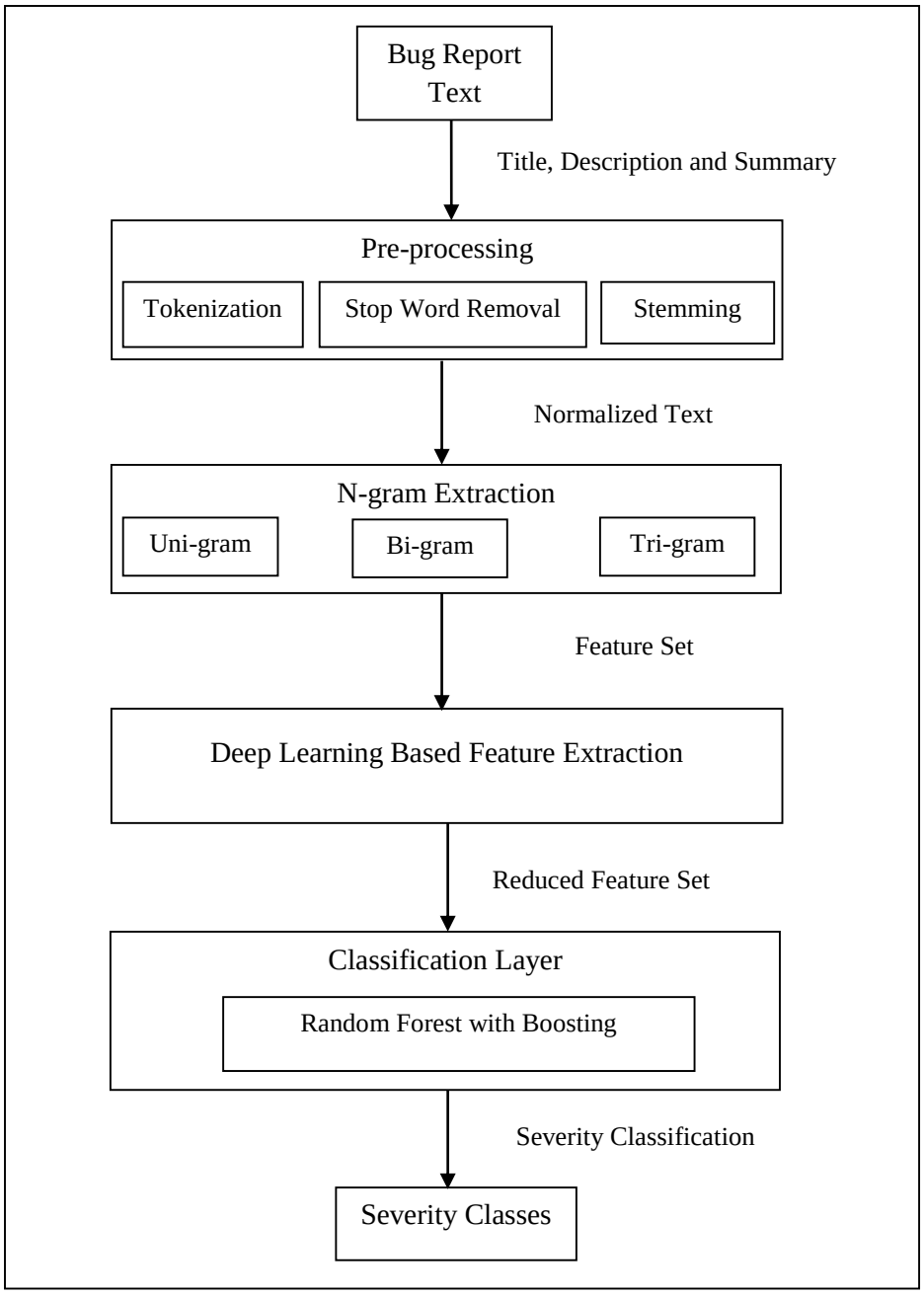


Figure 4.2: Proposed BSCM based on Deep Learning and RFB techniques

- Tokenization: In tokenization, stream of text is divided into several words, numbers, punctuations. The aim of tokenization process is to determine the unique tokens in each

bug reports. It is also noted that punctuations can be replaced with blank spaces, non-printable escape characters are removed, and capital words are converted into lowercase.

- Stop-word removal: It is referred to determine the stop words in given big reports. The stop words can be described as verbs, nouns, articles, pronouns, adverbs, prepositions etc.
- Stemming: It is the process to identify the common stem of words. These common stem of words act as features and can be stored in feature set. Suppose different occurrence of a word in bug report is give as "move", "moves", "moved", and "moving". All these occurrences can be replaced with word "move" and it is known as feature. Mathematically, it can be represented using equations 4.1-4.2.

$$BR = \{S_1, S_2 \dots \dots . S_j\} \tag{4.1}$$

$$S_j = \{f_1, f_2 \dots . f_k\} \tag{4.2}$$

4.4.2 N-gram Extraction (NE)

N-gram method explores the semantic relationship between the content of bug reports. This method also computes the frequency of features and describes the feature vectors in more meaningful manner [116]. The N-gram model can be described through features and these features are identified using unigram, bigram and trigram.

$$p(f_i|f_1, \dots \dots \dots . f_{i-1}) = p(f_i|f_{i-k+1}, \dots \dots, f_{i-1}) \tag{4.3}$$

In the equation 4.3, f_i represents the i^{th} feature and p denotes a probability function. In unigram, it is assumed that next features are independent to each other and mutual information cannot be considered among features. Hence, the conditional probability for unigram method is computed using equation 4.4.

$$p\left(f_1^k|\omega_1\right) = \prod p(f_i|\omega_1) \tag{4.4}$$

Here, ω_1 denotes the independent features, f_i represents the i^{th} feature and p denotes a probability function.

In case of bigram method, it is stated that continuous features share language information. So, the conditional probability of bigram method is measured using equation 4.5.

$$p\left(f_1^k|\omega_2\right) = \prod p(f_{i+1}|f_i, \omega_2) \tag{4.5}$$

In equation 4.5, ω_2 denotes the dependency of two adjacent features, f_i and f_{i+1} represent the two adjacent features and p denotes the probability function.

Trigram method is similar to bigram method. It trigram method, three consecutive features are considered to compute the conditional probability. It is computed using equation 4.6.

$$p(f_1^k|\omega_3) = \prod p(f_{i+1}|f_i, f_{i-1}, \omega_3) \quad (4.6)$$

In equation 4.6, ω_3 describes the relationship between three consecutive features i.e. , f_{i-1}, f_i and f_{i+1}. It is observed that different N-gram methods are taken into consideration to determine the frequency of features. These methods can be applied to analyze the sentence individually and sometime all these methods are applied to determine the all possible features for the given bug reports. The relationship between N-gram features can be described using equation 4.7.

$$p(f_1^k, f_{k+1}) = \sum p(f_1^k) p(f_{k+1}|f_k \dots f_{k-m}, \omega_i) p(\omega_i) \quad (4.7)$$

In equation 4.7, f_1^k denotes the mean of feature string with "n" number of features, f_{k+1} denotes the $(K+1)^{th}$ feature, ω_i denotes the probability of i^{th} feature string and $p(\omega_i)$ describes the probability of assumption i.

4.4.3 Feature Extraction with Deep Learning

This subsection presents the deep learning technique for feature extraction. The deep learning technique is employed to determine the reduced set of features from given set of features. The aim of deep learning technique is to determine more relevant features from the feature set. The feature extraction process with deep learning consists of five layers. These layers are described as Input, Convolutional, Activation, Dropout, Max pooling and Fully connected layer. The schematic diagram of feature extraction with deep leaning is demonstrated using Figure 4.3.

Convolutional Layer: The work of convolutional layer is to scan the input data through various kernels. These kernels correspond to different features. This layer also reduces the dimension of dataset using non-linear features and computed 2D convolutional. Initially, all features are computed using N-gram techniques and given to convolutional layer to generate new features. Suppose, $t_j \in \mathbb{R}^i$ and it can be interpreted as i^{th} dimension of j^{th} feature in a given $\mathbb{R}$ feature space and all other features can be represented using equation 4.8.

$$t_{1:n} = t_1 \oplus t_2 \dots \dots \oplus t_n \quad (4.8)$$

In equation, $\oplus$ denotes concatenation operator that executes filter operation $\in \mathbb{R}^i$ and t denotes the features.

$$f_i = l(w \cdot t_{j:j+g-1} + b) \quad (4.9)$$

In equation 4.9, w is a weight matrix, $\mathbf{t_{j:j+g-1}}$ denotes the window size, **b** is bias term related to feature space ($\mathbb{R}$) and **l** denotes a non-linear sigmoid function. The equation 4.9 is used to extract the new feature map and it is illustrated using equation 4.10.

$$f = [f_1, f_2, \ldots, f_{n-h+1}] \tag{4.10}$$

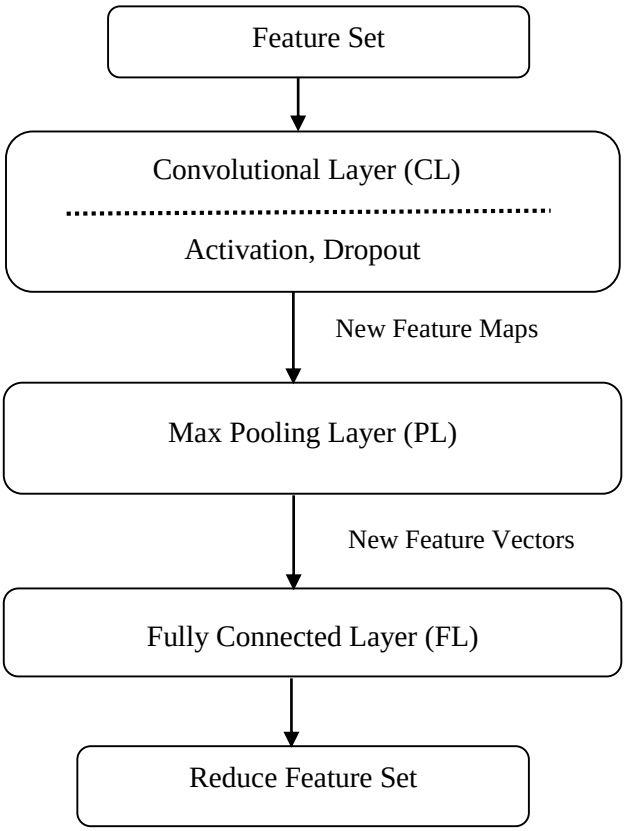


Figure 4.3: Proposed Deep Learning Based Feature Extraction

Max Pooling Layer: This layer reduces the size of feature map and it can be computed using max value. The main work of this layer is to pass valid information to the next layer using consecutive operation on complex features and also address the overfitting issue

Dropout Layer: The dropout layer addresses the co-adaption issue of features. This layer invokes a dropout rate function to disconnect the neurons from the connected layers. The dropout is set to 0.2 and training data is also efficiently generalized at this point.

Activation Function: The role of activation function is to activate the neurons. The neurons are activated on the basis of information. If, the information is appropriate, the corresponding neuron is activated, otherwise, it is deactivated. Further, the activation function also computes a neuron value for each neuron. In this work, sigmoid and Tanh activation functions are used.

Fully Connected Layer: In this layer, all extracted features are combined to generate feature set.

4.4.4 Classification Layer

The classification layer of proposed BSCM implements ensemble RFB technique as classifier. RFB technique is the integration of random forest (RF) [117] and boosting method [118]. In RFB technique, RF is used to determine different tree structures from the given dataset with different threshold. But, it is not capable to take intelligent decision regarding the combination of different trees. Hence in RFB technique, RF method is used to construct all possible tress using features space with respect severity classes. Whereas, boosting method is applied to compute the threshold values for model selection in testing phase. The algorithmic steps of RFB technique is presented in Algorithm 4.1 and Figure 4.4.

Algorithm 4.1 : Random Forest with Boosting (RFB)

Input: Training samples with CN features and labels
Output: Trained RFB

Step 1: Randomly select "k" features from total "m" features, Where k << m

Step 2: Among the "k" features, calculate the node "d" using the best split point.

Step 3: Split the node into daughter nodes using the best split

Step 4: Repeat 1 to 3 steps until "l" number of nodes will not be reached

Step 5: Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of decision trees

Step 6: Determine class label of training data

Step 7: Compute weighted error rate of decision tree

Step 8: Calculate the weight of decision tree's (w) using equation 4.11

$$w = {}^{1}/_{2} \log \frac{(M-1)(1-e)}{e} \quad (4.11)$$

Step 9: If (w >0)
Update weight of training samples

Step 10: Else reject the tree, repeat steps 7-10

* M is the number of classes, *e is the error rate of incorrect classified samples

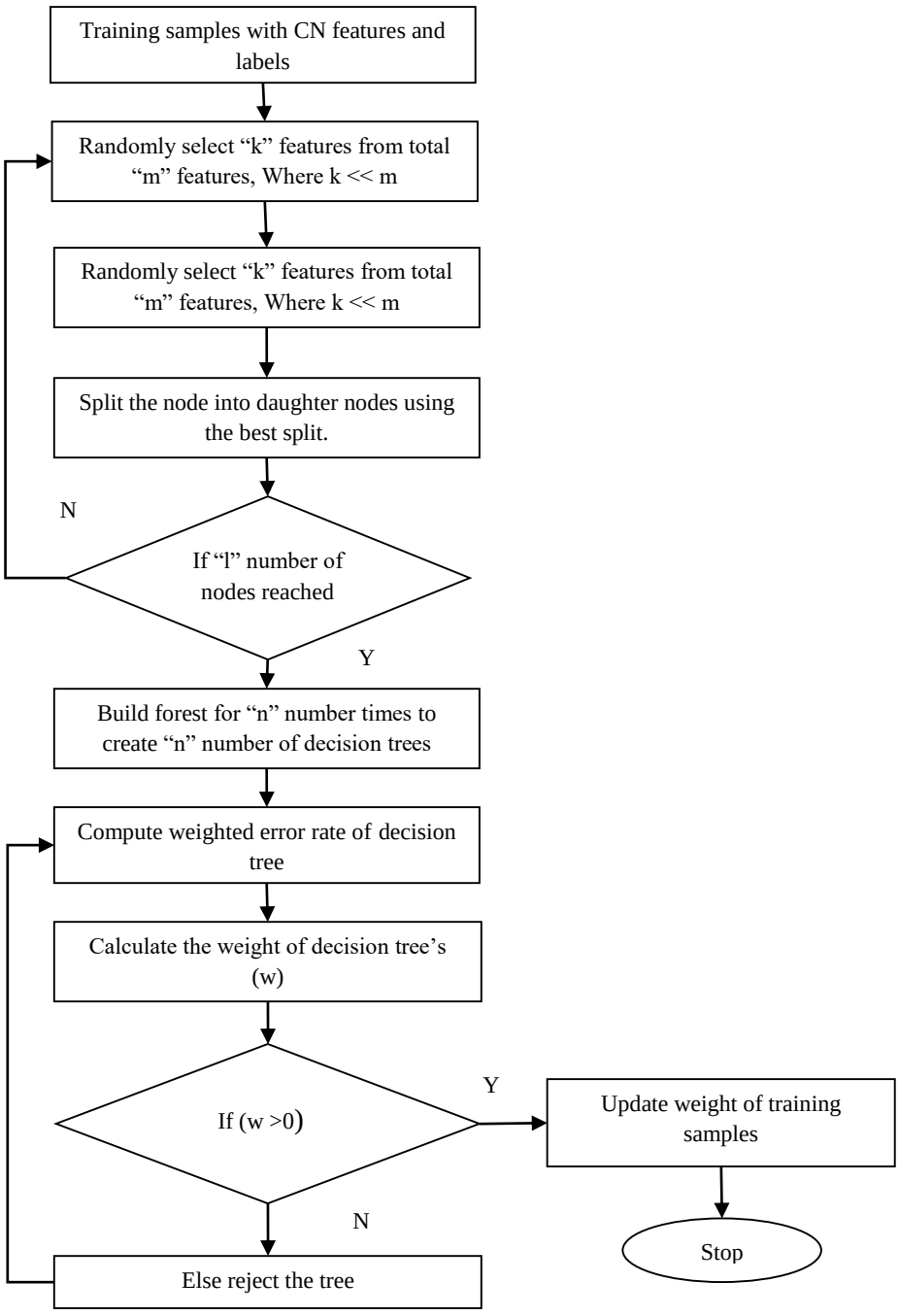


Figure 4.4: Flowchart of proposed random forest with boosting classifier

4.3 Experimental Results and Discussion

This section presents the simulation results of proposed BSCM. The effectiveness of the proposed model is tested over five bug report datasets. These datasets are Mozilla, Eclipse, JBoss, OpenFOAM and Firefox. The performance of proposed BSCM is evaluated using F-measure, Precision, Accuracy and Recall parameters [119]. The proposed model is implemented

using window 10 based system having Intel Core i5 (7th generation) processor, 8GB RAM and NVIDIA GEFORCE GPU and CPU @ 2.70GHz. Deep learning based feature selection technique is implemented using Keras and TensorFlow.

4.3.1 Performance Measure

This subsection describes the various performance measures that are considered to evaluate the performance of proposed BSCM model. These performance measures are listed below.

- **Accuracy:** This measure evaluates the performance of proposed model in terms of correctly classified data instance. It is computed using equation 4.12.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4.12}$$

In equation 4.12, TP denotes true positive instances, TN represents true negative instances, FP denotes false positive instance and FN denotes false negative instances.

- **Precision:** It indicates positive instances that can be predicted as actually positive. It is also known as Type-1 error. This parameter is computed using equation 4.13.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.13}$$

In equation 4.13, TP denotes the true positive instances that are positive, whereas FP denotes the false positive instances i.e. false instances that are predicted as positive instances.

- **Recall:** This parameter indicates actual positive instances among all positive instances. It is also known as Type-2 error and computed using equation 4.14.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.14}$$

In equation 4.14, TP denotes the true positive instances that are actual positive, whereas FN denotes the false negative instances i.e. positive instances that are predicted as false instances.

- **F-measure:** It is described in terms of precision and recall. It is different than accuracy parameter. The accuracy parameter considers the positive as well as negative data instances. But, F-measure only considers the positive data instances either identified as positive or negative. F-measure parameter is computed using equation 4.15.

$$\text{F} - \text{measure} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{4.15}$$

In equation 4.15, TP defines number of reports correctly labeled to a class, TN denotes number of reports correctly rejected from a class, FP denotes number of reports predicted correct, but

actual labelled as incorrect and FN denotes reports predicted incorrect, but actually labelled as correct.

4.3.2 Dataset

In this work, the five open source datasets are used [65]. These datasets are Mozilla, Eclipse, JBoss, OpenFOAM and Firefox. Table 4.1 presents the characteristics of these datasets.

Table 4.1 : Characteristics of bug report datasets used for experiments

Datasets	Total Instances	Features	Classes	Severity Classes
Mozilla	539	90	7	Blocker, Critical, Enhancement, Major, Normal, Minor, Trivial
Eclipse	693	68	5	Blocker, Critical, Enhancement, Major, Normal
JBoss	573	75	5	High, Low, Medium, Unspecified, Urgent
OpenFOAM	795	100	8	Blocker, Crash, Feature, Major, Minor, Text, Trivial, Tweak
Firefox	620	85	7	Blocker, Critical, Enhancement, Major, Normal, Minor, Trivial

4.3.3 Parameter Settings

This subsection illustrates the user defined parameters setting of proposed BSCM. The parameter settings of proposed model are presented in Table 4.2. Moreover, cross entropy is considered as loss function for the optimization process.

4.3.4 Results

This subsection describes the simulation results of proposed BSCM. The performance of BSCM is tested over five different datasets and evaluated using different performance measures presented in subsection 4.3.1. Further, the simulation results of proposed model are taken in terms of binary classification and multi class classification. The simulation results of binary classification are compared with existing Zhou et al. model [26].

Table 4.2: Parameters setting of proposed BSCM

Parameter	Value
Input	1000*200
Dropout Rate	0.2
Activation Function	sigmoid, Tanh
Filter Size	20
Stride	1
Depth	128
Pooling	Max Value
Fully connected layer	Random forest

4.3.4.1 Binary Classification Results

This subsection presents the simulation results of proposed BSCM using binary classification. For binary classification, bug datasets are divided into two classes i.e. bug and non-bug. The simulation results of proposed model are compared with Zhou et al. model [65]. The precision, recall and f-measure parameters are considered to evaluate the performance of proposed model. Table 4.3 illustrates the simulation results of proposed model and Zhou et al. model. It is observed that proposed model obtains better quality results than Zhou et al. model. It is seen that proposed model obtains the f-measure rate in the range of 91- 99%, while the f-measure rate of Zhou et al. model ranges in between 79-93%. On the analysis of precision parameter, it is stated that precision rate of proposed model is in between 92-99%. Whereas, Zhou et al. model achieves the precision rate in the range of 80-93%. Moreover, it is also revealed that proposed model obtains higher recall rate using all datasets i.e. 92-99%. Whereas, the recall rate of Zhou et al. model is ranging in between 80-93%. Hence, it is concluded that proposed model improves the severity classification rate significantly than Zhou et al. model. From simulation results, it is noticed that the proposed deep learning based feature extraction technique determine the relevant features for prediction task. It is stated that proposed feature extraction technique enhances the results of proposed BSCM.

Table 4.3: Experimental results of proposed model and Zhou et al. model using five datasets

Datasets	Zhou et al.			Proposed model		
	Precision (%)	Recall (%)	F-measure (%)	Precision (%)	Recall (%)	F-measure (%)
Mozilla	82.60	82.40	81.70	98.48	98.52	98.12
Eclipse	81.80	82.10	81.60	99.38	99.48	99.12
JBoss	93.70	93.70	93.70	98.88	98.95	98.42
OpenFOAM	85.30	85.30	84.70	95.25	95.35	94.55
Firefox	80.30	80.50	79.50	92.75	92.95	91.95

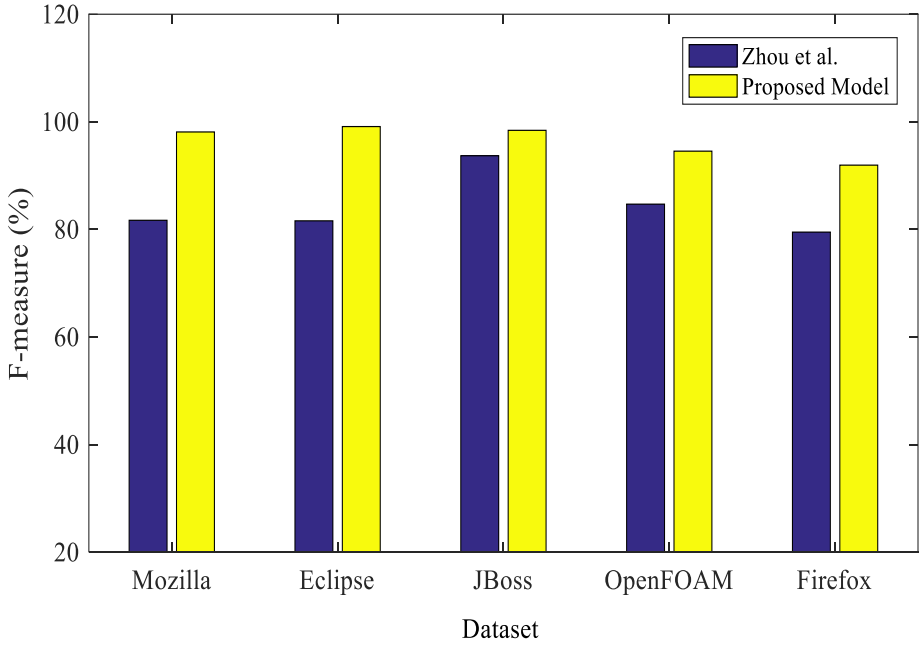


Figure 4.5: Comparison of proposed model and Zhou et al. model using F-measure parameter

Figure 4.5 demonstrates the simulation results of proposed model and Zhou et al. model for all datasets using f-measure parameter. It is seen that proposed model obtains higher f-measure rate than Zhou et al. model using all dataset. It is also noticed that proposed model obtains higher f-measure rate for Eclipse dataset i.e. 99.12 % than Mozilla, JBoss, OpenFOAM and Firefox datasets.

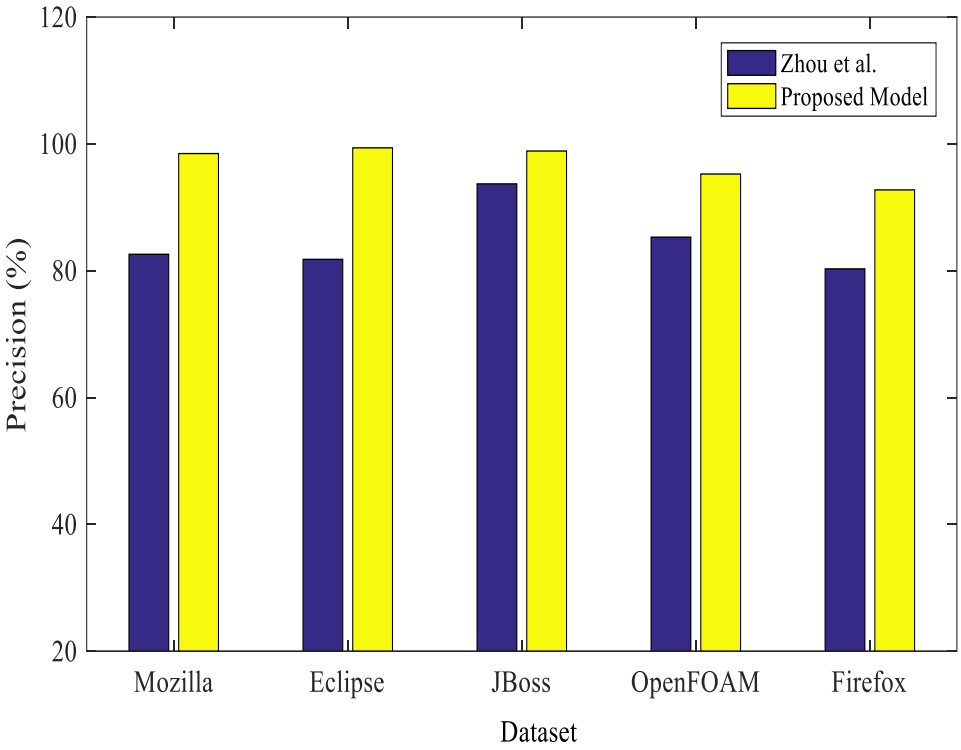


Figure 4.6: Comparison of proposed model and Zhou et al. model using precision parameter

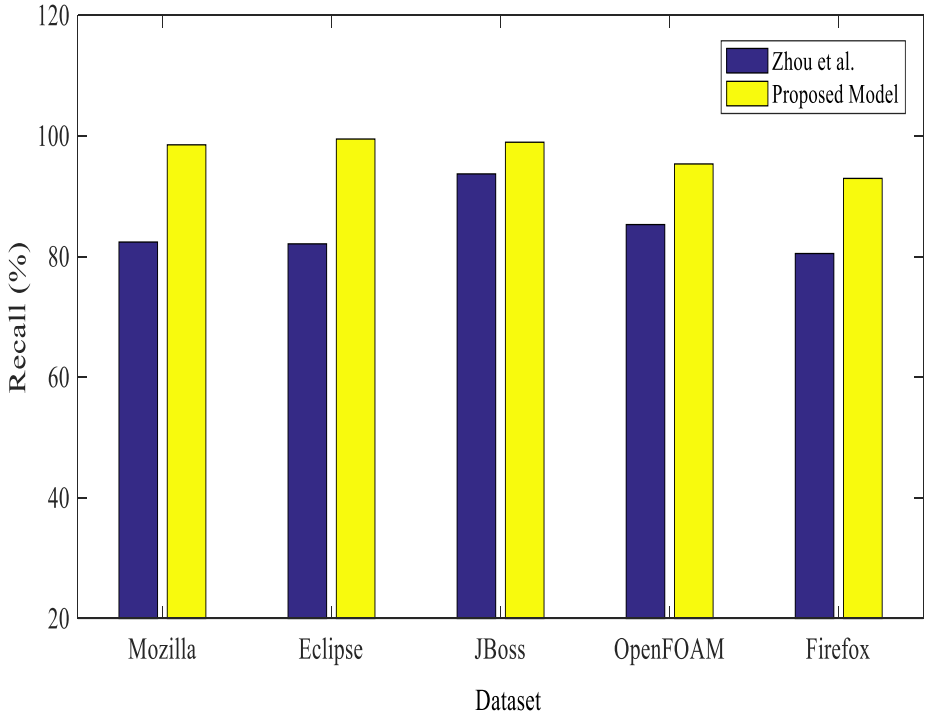


Figure 4.7: Comparison of proposed model and Zhou et al. model using recall parameter

The precision parameter results of proposed model and Zhou et al. model using all datasets is presented in Figure 4.6. It is stated that proposed model provides higher precision rate than Zhou et al. model. Further, it is also observed that proposed model obtain better precision rate for Eclipse dataset. Figure 4.7 shows the simulation results of proposed model and Zhou et al. model using recall parameter. Again, it is seen that proposed model obtains better recall rate as compared to Zhou et al. model. It is also noted that proposed model obtains higher recall rate for Eclipse dataset in comparison to other datasets being taken.

4.3.4.2 Multi- Class Classification Results

This section presents the simulation results of proposed BSCM using multi class severity prediction. The multi class severity prediction is more difficult than binary severity prediction. Hence, efficiency of the proposed model is also investigated using multi class severity prediction. It is also mentioned that previous works is not reported on multi class severity prediction in literature. The same datasets are used for multi class severity prediction. The details of these datasets along with the severity classes is presented in Table 4.1. The simulation results are evaluated using f-measure, recall, precision and accuracy parameters. The simulation results of proposed model using all datasets are illustrated in Table 4.4. It is also observed that proposed model effectively computes the f-measure, recall, precision and accuracy rates for each severity class of every dataset. The average precision rate of proposed model for Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets is 94.47%, 96.19%, 94.55%, 98.67% and 97.33% respectively. Whereas, average recall rate obtained by the proposed model for Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets is 91.46%, 93.57%, 91.76%, 95.96% and 94.76% respectively. The average F-measure rate of Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets is 93.27%, 94.86%, 93.88%, 97.23% and 95.96% respectively. The average accuracy rate of proposed model is 94.92%, 95.76%, 94.91%, 98.89% and 97.22% for Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets respectively. It is also noticed that JBoss dataset achieves higher average accuracy, precision, recall and f-measure rate than others datasets. Figures 4.8-4.11 demonstrates the simulation results of proposed model using accuracy, precision, recall and f-measure parameters for all datasets in graphical manner. It is stated that average results of all classes of each dataset is used to plot these figures. Hence, it is concluded that the proposed model is also capable and efficient for multi class severity prediction.

Table 4.4: Experimental results of proposed model for severity classification using five datasets

Datasets	Classes	Precision (%)	Recall (%)	F-measure (%)	Accuracy (%)
Mozilla	Blocker	82.9	80.23	81.54	85.45
	Critical	96.43	95.34	94.34	96.33
	Enhancement	95.33	80.22	95.45	97.43
	Major	98.45	97.45	92.34	96.3
	Normal	97.44	95.34	96.45	95.24
	Minor	94.35	96.34	97.44	96.34
	Trivial	96.35	95.34	95.33	97.33
Firefox	Blocker	86.16	83.35	84.73	83
	Critical	97.97	92.23	95.27	97.92
	Enhancement	98.3	92.23	95.98	97.55
	Major	97.98	97.61	96.64	97.19
	Normal	97.28	96.9	97.64	97.53
	Minor	97.23	97.04	97.24	98.33
	Trivial	98.43	95.68	96.52	98.85
Eclipse	Blocker	80.34	81.23	83.45	83.45
	Critical	97.33	89.23	96.35	98.33
	Enhancement	98.34	90.29	95.35	98.32
	Major	99.4	97.85	95.85	97.22
	Normal	97.35	97.29	98.4	97.24
Jboss	High	96.8	97.29	97.84	98.29
	Low	98.27	97	96.75	98.93
	Medium	99.08	95.45	96.72	99.22
	Unspecified	99.59	93.68	97.08	99.19
	Urgent	99.59	96.37	97.76	98.82
OpenFOAM	Blocker	83.45	82.34	83.45	83.4
	Crash	98.7	98.42	98.89	99.38
	Feature	99.28	97.81	98.33	98.56
	Major	99.61	98.51	97.11	99.54
	Minor	99.06	95.29	97.02	99.28
	Text	99.29	91.21	97.3	99.77
	Trivial	99.45	95.52	97.05	99.22
	Tweak	99.82	99.02	98.57	98.68

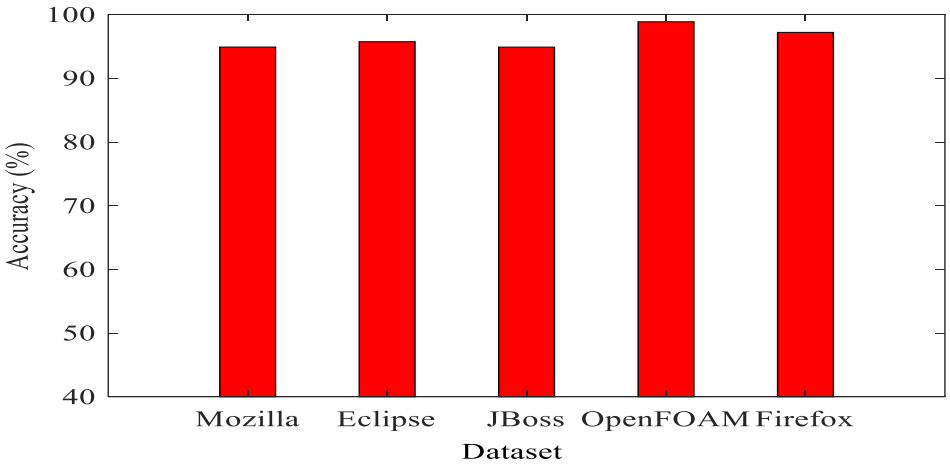


Figure 4.8: Illustrates the results of proposed BSCM using accuracy parameter of all datasets

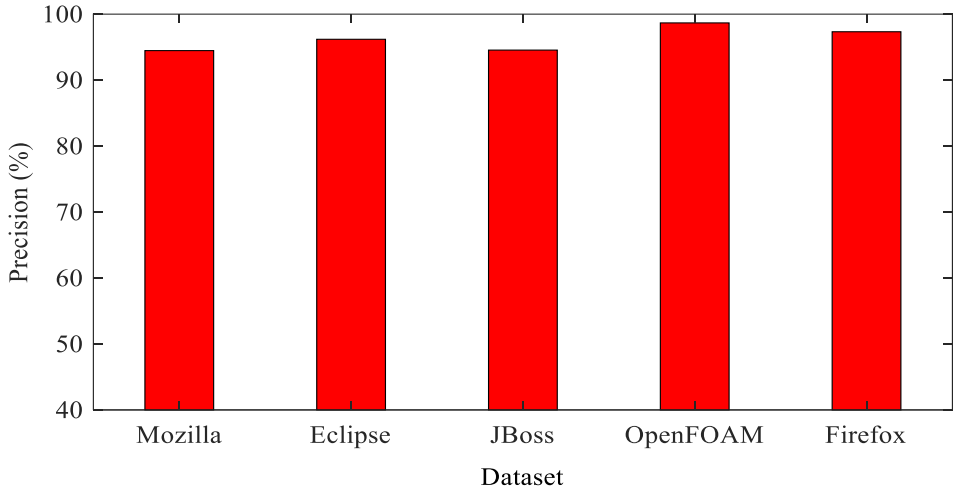


Figure 4.9: Illustrates the results of proposed BSCM using precision parameter of all datasets

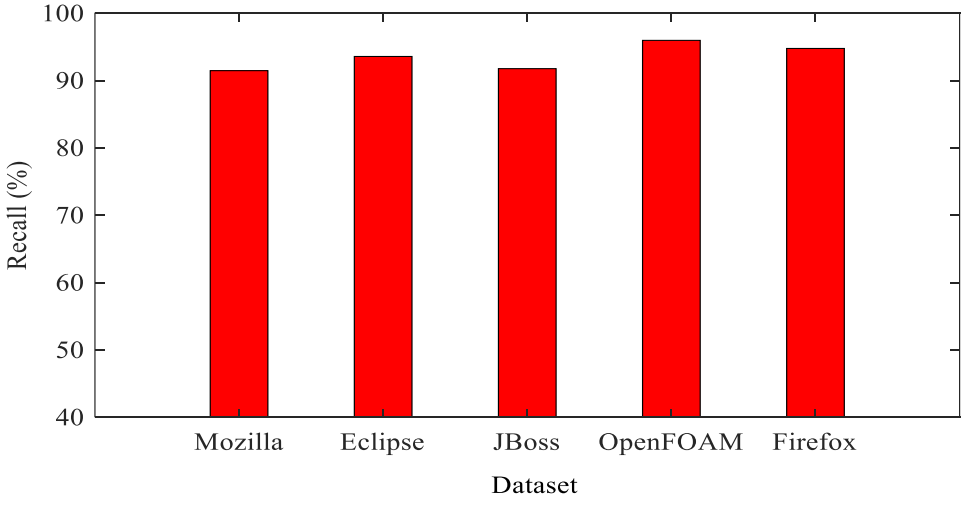


Figure 4.10: Illustrates the results of proposed BSCM using recall parameter of all datasets

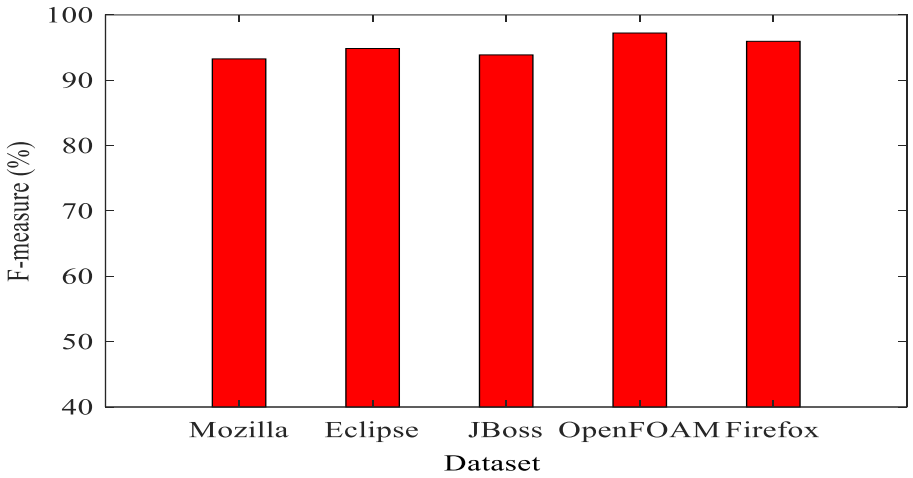


Figure 4.11: Illustrates the results of proposed BSCM using f-measure parameter of all datasets

RQ1: Can deep learning approach extract relevant features for automated BSCM?

Ans. RQ1): In the proposed BSCM, two techniques are employed i.e. deep learning and RFB. The deep learning techniques is employed to extract relevant features and RFB is applied for classification and prediction task. RFB is the combination of random forest and boosting methods and applied to predict the severity classes. In proposed model, initially N-gram techniques are applied to extract the features from bug reports. The extracted features are stored in a feature set. But, it is noted that all features are not equally important and some of are irrelevant. In turn, the performance of bug severity model can be degraded. To improve the performance of bug severity model, various researchers have adopted feature extraction technique to refine the feature set. In this work, deep learning based feature extraction technique is applied to determine the relevant features form feature set. It is stated that N-gram technique extracts 90, 68, 75, 100, 85 features for Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets respectively. Further, deep learning based feature extraction method is applied on above mentioned feature set to identifies relevant features for design the final dataset. It is stated that deep learning technique significantly reduces features in feature set for all datasets. The relevant features determined through deep learning technique for Mozilla, Eclipse, JBoss, OpenFOAM and Firefox datasets are 7, 5, 5, 8 and 7 respectively. These features are used to design the final datasets and these datasets are used to test the efficiency of proposed BSCM. The simulation results of proposed model are reported in Table 4.3 and 4.4 for binary and multi class severity

prediction respectively. It is stated that proposed model obtains state of art results for severity prediction as compared Zhou et al. model. The significant improvement can be seen in simulation results of proposed model. It is only possible due to extraction of relevant features using deep learning based feature extraction technique. Hence, it is stated that deep learning technique extracts relevant features for automated BSCM and also improve the severity prediction results.

RQ2. **Is integration of random forest and boosting method improve the performance of BSCM than traditional machine learning technique?**

Ans. RQ2): In BSCM, RFB classifier is proposed for prediction the severity classes. The RFB classifier is the combination of random forest and boosting methods. In RFB, random forest method constructs all possible tree structure using the features of datasets. Whereas, boosting method computes threshold values of each node of the tree. The simulation results of proposed BSCM model is compared with Zhou et al. model. This model consists of Multinomial Naive Bayes (MNB) classifier for severity prediction. The simulation results of both models are reported in Table 4.3. It is observed that proposed model provides more significant results than Zhou et al. model. It is noticed that average f-measure of proposed model ranges in between 91% to 99 %, whereas f-measure rate of Zhou et al. model is in between 79-93%. Similar types of results are obtained for recall and precision parameters using proposed model and Zhou et al. model. Hence, it can be said that proposed model outperforms than Zhou et al. model. In proposed model, RFB classifier is implemented for severity prediction. So, it can be stated that integration of random forest and boosting method improves the performance of BSCM and also provides better results than traditional machine learning technique.

4.4 Summary

In this chapter, a bug severity model based on deep learning and RFB method is proposed for severity classification, called BSCM. In the proposed model, deep learning based feature extraction technique is applied to determine the relevant features for the prediction of severity classes. Further, RFB classifier is developed to classify the data in different severity classes. The RFB technique is the combination of random forest and boosting methods. The performance of the proposed model is tested over five different datasets. The simulation results are evaluated using f-measure, precision, recall and accuracy parameters. Moreover, the results are also

interpreted using binary severity prediction and multi class severity prediction. The performance of proposed model is compared with Zhou et al. model for binary bug severity prediction. It is observed that proposed model provides better results than Zhou et al. model. It is also stated that proposed model also obtains good results for multi class severity prediction. Hence, it can be said that proposed model is one of efficient, effective and capable model for bug severity prediction.

CHAPTER 5
DEVELOPER RECOMMENDATION SYSTEM FOR BUG ASSIGNMENT

5.1 Introduction

A bug repository consists of large number of bug reports. These bugs can be characterized as valid, invalid, important, unique, critical, duplicate, blocker, unimportant and so on [82]. All these bugs are analyzed through triager. The task of triager is to identify the real and important bugs that can require immediate attention of the developer to resolve it. Hence, to identify the appropriate developer for resolving the bugs is one of difficult task [83]. On the other side, large number of bugs are deposited in bug repository per day and size of bug repository increases tremendously. It becomes challenging for a triager to handle the bugs manually and also determine the important bugs that need to be addresses quickly [84-86]. So, to determine the appropriate developer to address the important bugs efficiently and quickly is challenging task for bug assignment model [95-97]. This chapter addresses the appropriate developer issue of bug assignment model. This issue is handled through severity prediction of bugs and further, the developers are assigned to resolve the bugs on the basis of three metrics. Hence, in this chapter, a new bug assignment model is developed to determine the appropriate developer. The proposed model is based on swarm intelligence [120] and machine learning (ML) approaches [121]. The swarm intelligence technique is applied to extract the important attributes of bug reports. Whereas, ML techniques are employed to identify the bugs severity.

5.2 Proposed Developer Recommendation (DevRE) System

This section presents the developer recommendation system for bug assignment based on swarm intelligence and ML approaches. The aim of proposed model is to predict the bugs severity and choose the appropriate developer for resolving the bugs according bugs severity. Further, in this work, ant colony optimization (ACO) based feature weighting technique is applied to determine the important attributes from the feature set. The NB and SVM approaches are used to measure the severity of bugs [122-123]. Further, a list of developer is available as per bugs severity that can solve the bugs efficiently and quickly in past. The schematic working of proposed model is

illustrated in Figure 5.1 and consists of four phases. These phases are preprocessing, feature extraction, ACO based feature weighting, and developer assignment.

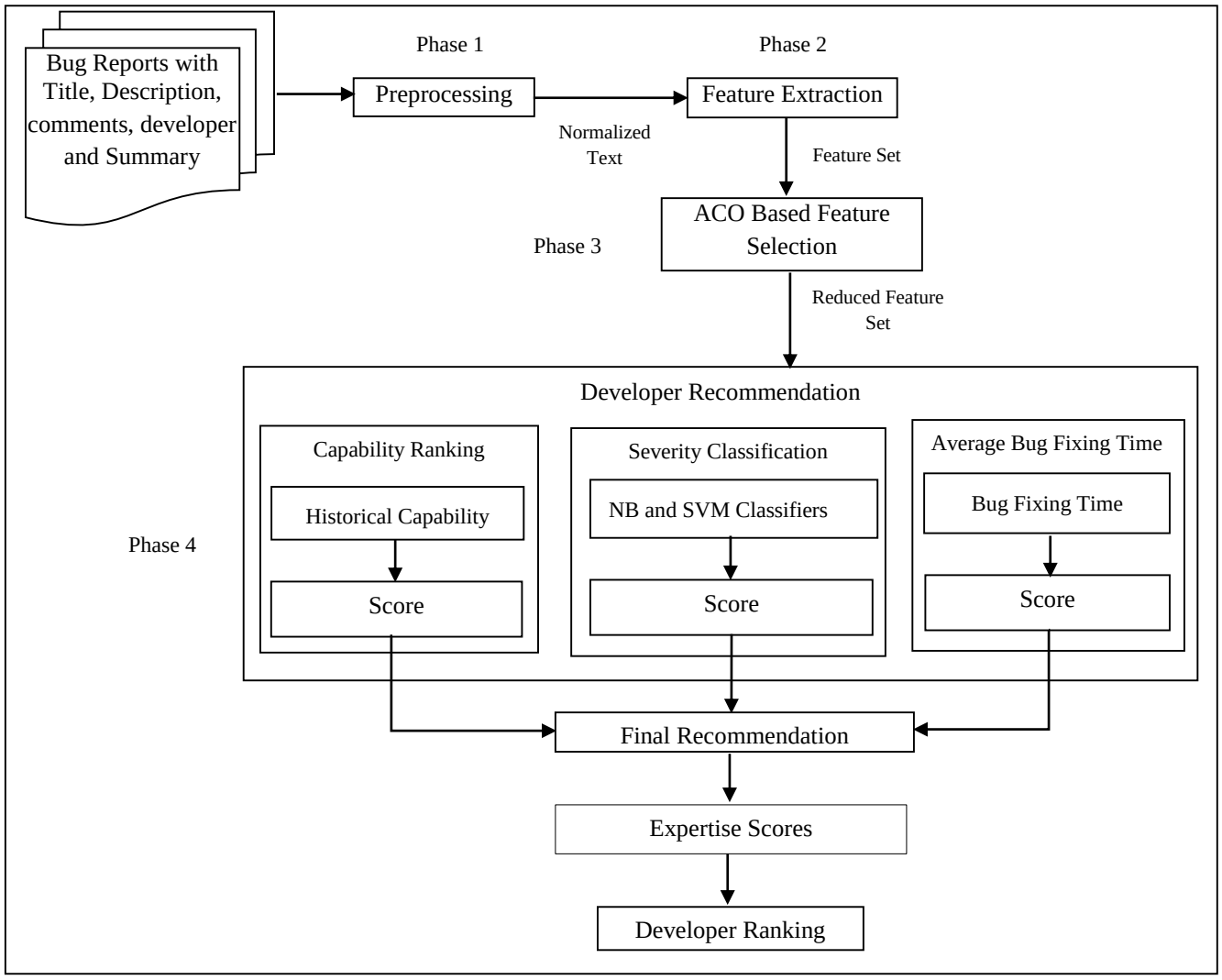


Figure 5.1: Proposed Developer Recommendation System

5.2.1 Preprocessing Phase

The first phase of proposed bug assignment model is preprocessing phase. In this phase, various preprocessing methods are employed on the content of bug reports. The bug reports are described in any natural language especially English and consist of paragraphs. The objective of preprocessing phase is to determine the tokens from paragraph and this process is known as tokenization. Further, stop words and unwanted words are removed from the list of tokens [110].

5.2.2 Feature Extraction Phase

In this phase, TF-IDF and N-gram methods are applied to extract the features and also described the textual features as vector-space model [116,124]. This model describes the bug reports in terms of weighted features and this process is illustrated using equation 5.1.

$$BR_x = (w_1 f_1, w_2 f_2, \dots \dots . w_n f_n) \tag{5.1}$$

Where, n denotes the total number of features and $w_i f_i$ is the weight of i^{th} feature.

Further, it is stated that TF-IDF is a statistical method for extracting the informative features. This method considers the frequency of words to identify the features in bug reports and importance of features is measured through its occurrence. Words (feature) that are frequently occurred in bug reports, having more significance and weight than other words or feature. Sometimes, it is not necessary that frequency of words can be considered as significant parameter to measure the importance of features. Assume, $tf(x, y)$ is the term frequency of y^{th} feature in the bug report BR_x , $df(y)$ is the document frequency of y^{th} feature and N represents the total number of bug reports in the corpus. TF-IDF is computed using equation 5.2.

$$TF - IDF(x, y) = tf(x, y) \times \log\left(\frac{N}{df(y)}\right) \tag{5.2}$$

TF-IDF represents each feature in vector space and corresponding bug report vector is described using equation 4.3.

$$V_{BR_x} = \left(tf(BR_x, y_1), tf(BR_x, y_2) \dots \dots . tf(BR_x, y_x)\right) \tag{5.3}$$

A feature in vocabulary expresses the dimension of bug report vector. Suppose, $tf(BR_2, y_1)$ is the term frequency with feature y_1 in bug report BR_2. If a query feature is not present in bug report, then zero value is put in the matrix for corresponding feature. If, this value is multiplied with other features, the values of these features will be zero. Hence, the corresponding bug report is not retrieved, and it is known as data sparsity. The data sparsity issue is handled through N-gram method. Because, it follows word pairs paradigm i.e. bigrams and trigrams and also preserve the semantic relationship of texts. It is based on the feature segment and weight of N-gram is calculated using equation 5.4.

$$F(f_n | f_{n-N+1}^{n-1}) = \frac{C(f_{n-N+1}^{n-1} f_n) + 1}{C(f_{n-N+1}^{n-1}) + v} \tag{5.4}$$

Where, f_n is the next feature, f_{n-N+1}^{n-1} represents the sequence of features, v is the total number of possible (N-1) gram and C is the probability.

Algorithm 5.1: Preprocessing and Feature weighting

Input: $BR_h = (BR_1, BR_2, \ldots\ldots BR_h)$ a textual bug reports with seven classes, h=7.

Output: $BR_h = (BR'_1, BR'_2 \ldots\ldots\ldots BR'_h)$ feature set of bug reports.

Step 1: Read each bug report, bug reports of class, from the bug report dataset.

Step 2: Tokenized the text and removed the stop words from the text.

Step 3: Calculate the occurrence weight for every unigram feature of the BR, by TF-IDF approach.

Step 4: Calculate the occurrence weight for every bi-gram and tri-gram in bug report BR, by N- gram approach.

5.2.3 Feature Selection Phase

The aim of this phase is to select the relevant features for measuring bugs severity. In feature extraction phase, lot of features are extracted based on statistical and semantic information of bug reports. But, it is observed that all features are not equally important. Sometimes, large number of features can result in low prediction rate and also responsible for non-linearity problem in data. In turn, the performance of classifiers can be degraded. Hence, feature selection is an important technique that can improve the performance of classifiers. In this technique, relevant features are selected from the feature set. In this work, ACO based feature selection technique is applied to select the optimum features for bug severity. This algorithm is developed by Dorigo et al. for solving the constrained and unconstrained optimization problems [125]. This algorithm has been applied to solve variety of optimization problems such as function optimization, clustering, wireless sensor network, image processing, routing etc. [126-129], and obtains optimal results for these problems. In feature selection phase, an ACO based feature weighting technique is implemented to identify the relevant features for bug severity prediction. The aim of this technique is to compute the weight of each features presented in the feature set. The weight of features is used to determine the significance of features and further, the features with higher weight are selected for bugs severity prediction. The working of ACO based feature weighting technique is described as follows. To determine the relevant features, the entire dataset divides into k number of clusters. Initially, k number of cluster centers are randomly chosen from the dataset. Further, the cluster centers are optimized using the ACO based clustering algorithm. The next step is to arrange the different data instances into k clusters using distance function. Generally, Euclidean distance is used for data arrangement in respective

clusters [130]. Now, a weight function is used to compute the weight of each feature of feature set and on the basis of weight function, features are selected for severity prediction. The working procedure of ACO based feature weighting is illustrated in figure 5.2. Consider, a bug report with n number of features, m number of data.

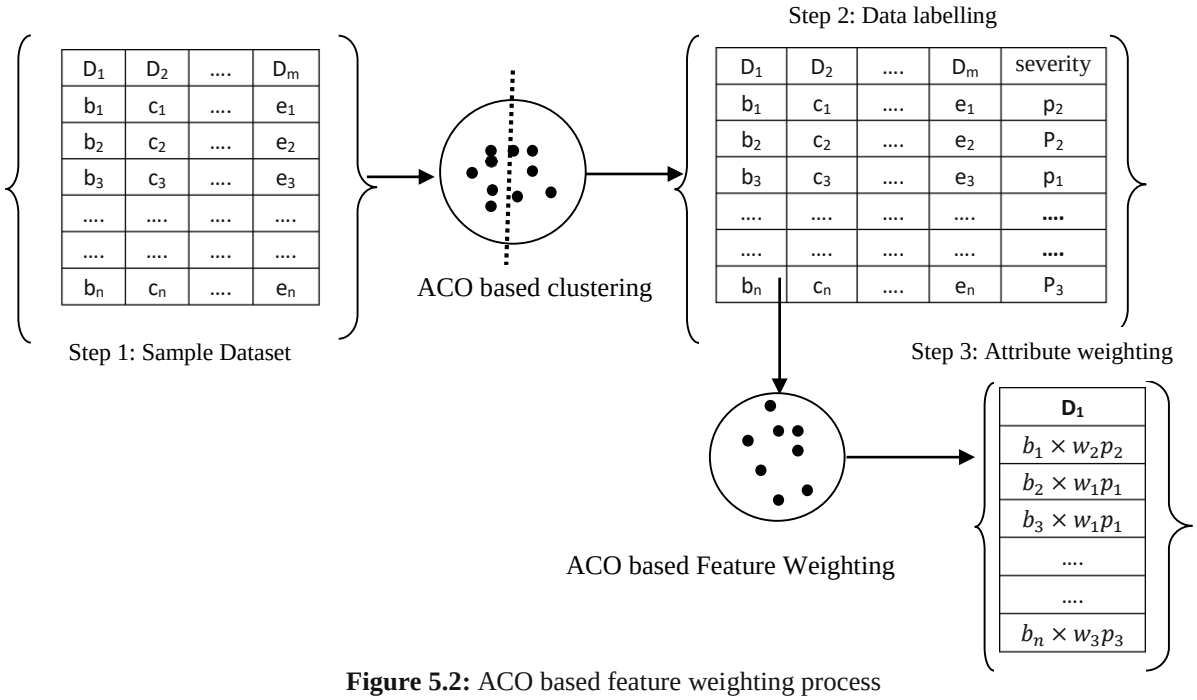


Figure 5.2: ACO based feature weighting process

instances, and k number of clusters. The features of bug report are described using $D_1, D_2, D_3, \ldots, D_n$. Step 1 illustrates the features of bug dataset and corresponding values of these features. Step 2 corresponds to bug severity levels identified through ACO based clustering algorithm and the predicted severity levels can be associated with each bug report such as $SL_1, SL_2, SL_3, \ldots, SL_n$. A weight function ($W_g$) is also computed for each severity level. This weight function is multiplied with features values to obtain the final weight of features. In step 3, weight of each feature bug dataset is computed through a weight function. The weight function (w_g) is described using equation 5.5 and it is computed for each attribute of sample dataset. The weight coefficient is computed using following equations.

$$w_g = \frac{\sum_{g=1}^{D} F_g}{\sum_{g=1}^{D} C_g} \quad \text{i. e. } F_g = \frac{\sum_{k=1}^{SL} \sum_{i=1}^{n} x_{i,k}}{i} \tag{5.5}$$

In equation 5.5, w_g represents the weight coefficient of g^{th} feature, F_g denotes the average weight of g^{th} feature, C_g denotes the cluster center of g^{th} feature computed through ACO

algorithm, i denotes number of bug reports with same severity level within in the features, SL denotes the severity level of bug reports. The steps of proposed ACO based feature weighting technique is given in Figure 5.3 and Algorithm 5.2

Algorithm 5.2 : Proposed ACO Based Feature Weighting Technique

Input: Bug report dataset
Output: Reduced feature set

Step 1: Load the dataset and specify the numbers of initial clusters centers

Step 2: Initialize the population of ants, set the intensity of pheromone trial and maximum iteration.

Step 3: Randomly identified initial cluster centers for every ant

Step 4: Evaluate the value objective function using sum of squared distances and assign the items to the clusters with minimum objective function

Step 5: While (condition is not met)

Step 6: Calculate the transition probability using equation 5.6

$$p_A(T) = \frac{(\tau_A(T))^{\alpha}.\eta_A^{\beta}}{\sum_A(\tau_A(T))^{\alpha}.\eta_A^{\beta}} \quad (5.6)$$

Step 7: Update the pheromones using equation 5.7.

$$\tau_A(T+1) = \rho\tau_A(T) + \Delta\tau_A(T) \quad (5.7)$$

Step 8: Determine the new positions of ants

Step 9: Recalculate the value of objective function

Step 10: End while

Step 11: Obtained the optimal clusters centers

Step 12: Determine the final weights of each features (f_i) by using equation 5.8

$$f_i = \left(\sum_{i=1}^{d}\sum_{j=1}^{h}\sum_{k=1}^{k}\frac{X_{ih}}{C_{ik}}\right) \times \frac{1}{d} \quad (5.8)$$

Step 13: Select the features with maximum weight.

* η_β describe the background information of features to improve the results, $\tau_A(T)$ is pheromone amount for the Ath feature in time T, α and β are the control parameters that provides the pheromone and background information, and $p_A(T)$ is the Transition probability , ρ is defined as evaporation rate of pheromone trail and lies between [0, 1], $\Delta\tau_A(T)$ is the pheromone trail amount added to Ath feature between time Δ and ΔT , d is the total number of features, h is the number of data instance, k is the number of clusters, X_{ih} is the ith feature of hth data instance, C_{ik} is the ith feature of kth cluster

The output of feature selection phase is reduced set of features. Further, these features are used to train the machine learning classifier and predict the bugs severity.

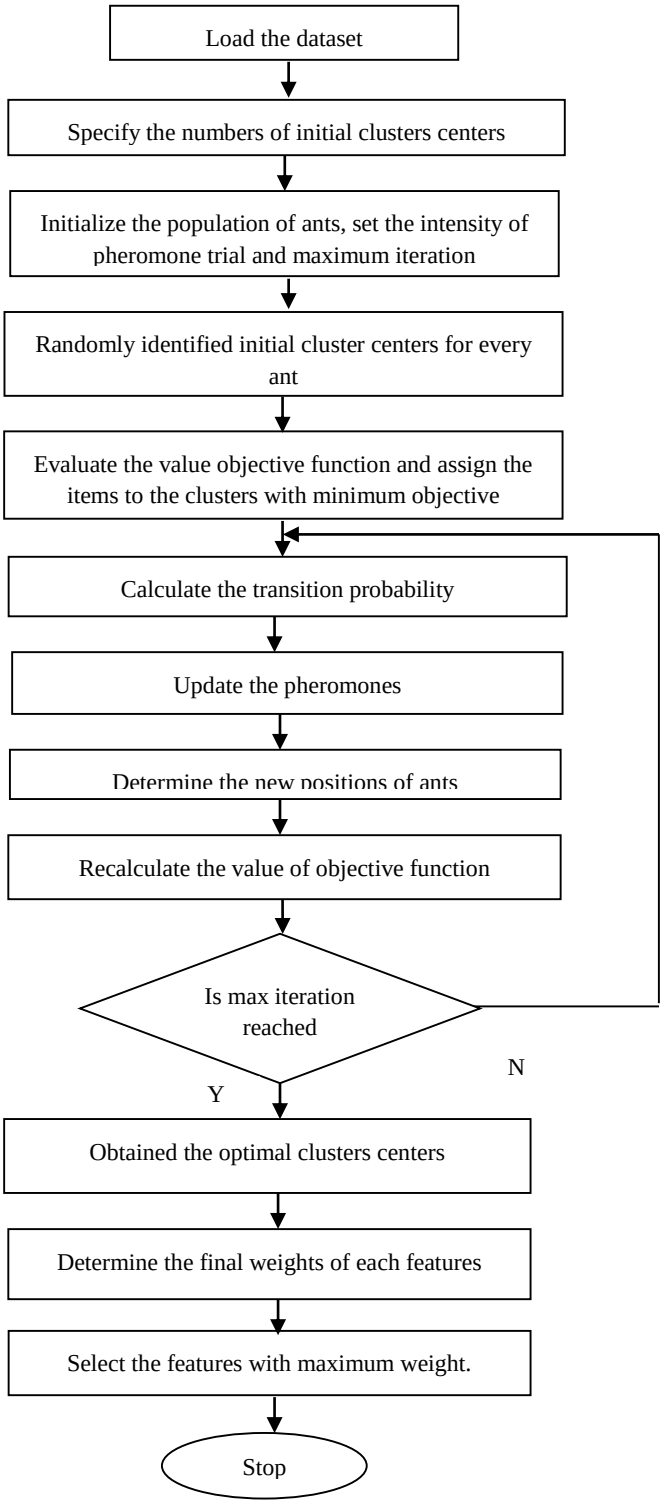


Figure 5.3: Flow chart of proposed ACO Based Feature Weighting Technique

5.2.4 Developer Recommendation Phase

This subsection presents the idea of developer recommendation system. This phase provides a list of relevant developer to solve the bug reports. The appropriate developers are identified using metrics, similarity and recommendation processes. These processes are described below.

5.2.4.1 Proposed Metrics: To determine the relevant developer, three matrices are proposed in this work. These metrics are capability ranking, bug severity level and average bug fixing time. The final ranking of developer is devised on the basis of these metrics. These metrics are described as follows.

- **Capability Ranking(C_{dev}):** This ranking is based on the developer's skills for solving the bug reports. It is noticed that each bug report requires some specific skill to solve it. It is stated that a developer can handle the bug reports efficiently, if similar type of bug is solved in near past. Hence, capabilities of developer's can be explored for solving bug reports. Further, it is also observed that capabilities of developer's can be tested through machine learning approaches with respect to historical data. If, a new bug is reported in bug repository, its severity is measured. A relevant developer is identified using machine learning approaches for solving it. The capability ranking is based on the skills of developers to fix the bugs. It can be computed using the past experience of developers for fixing the bugs. So, in this work, capability ranking is used to rank the developers instead of machine learning classifiers.
- **Bug Severity Level:** In this work, bug severity is also considered as possible metric for rank the developers. It is fact that all bugs are not equally important and not required immediate attention. Some bugs are classified as more severe than other and these bugs require immediate attention of developer. Otherwise, unusual things will be reported. So, the severity of bug reports is important aspect and can be classified into different severity levels. It can be easily done through NB and SVM classifiers. It is stated that bug report datasets are available in literature and machine learning classifiers can be trained on these datasets and predicted the severity levels. If, new bugs are added into repository, severity level of these bugs can be predicted through machine learning classifiers. Further, the appropriate developer is allocated to fix the bugs. It is also assumed that list of developers is prepared according severity levels. But, a developer can fix the bugs with more than one severity levels. So, a weighted severity list is described for each developer that maintains the sequence of severity levels handled through developer. It can be defined as ratio of different severity level of bugs handled through a

developer and number of bugs fixed. The weighted severity list is computed using equation 5.9 for a developer.

$$SL_{list} = \sum_{i=1}^{SL} \beta \times \frac{Bug_i}{Total\ Bug} \tag{5.9}$$

In equation 5.9, Bug_i denotes i^{th} level of bug severity, SL denotes the number of severity levels, β represents the criticality of bugs to be resolved and SL_{list} denotes the severity list associated with developers.

- **Average Bug Fixing Time (T_{avg}):** This metric is described in terms of average time taken by a developer for fixing the bugs. Bug fixing time consists of various factors such as problem formulation through developers, solution construction, review and assess the code. It is also an important parameter to identify the relevant developer due to criticality of bugs. The average bug fixing time can be defined as ratio of total time taken by a developer to solve bugs and number of bugs solved.

On the analysis of bug reports, it is noticed that bug reports consist of different fields such as component, product, timestamp, severity level, expertise etc. Hence, the afore mentioned metrics are multiplied to obtain the final ranking of developers and it is computed using equation 5.10.

$$D_{rank} = \gamma_1 C_{dev} + \gamma_2 SL_{list} + \gamma_3 T_{avg} \tag{5.10}$$

In equation 5.10, D_{rank} denote the final rank of developers, C_{dev} describes the capability learning of developer, SL_{list} denotes severity level of bugs and T_{avg} denotes the average bug fixing time. γ_1, γ_2 and γ_3 are three control parameters such that $\gamma_3 < \gamma_1 < \gamma_2$ and value of these parameter is in the range of 0 and 1.

5.2.4.2 Similarity Process

Whenever, a new bug is reported in the bug repository system. The features are extracted from bug reports. Further, similarity process in invoked to measure the severity of bug reports and also to determine the appropriate developer. In this work, cosine similarity is adopted for measuring the similarity between bug reports and appropriate developer. Through the similarity process, a list of potential developers is prepared to fix similar type of bugs.

5.2.4.3 Recommendation Process

The final step of developer recommendation phase is to determine the rank of developers. Developers are ranked according their capability to handle and fix the bugs. Basically, developer metrics i.e. capability ranking, bug severity level and average bug fixing time are used to rank each developers and final ranking of developer is computed using equation 5.10. Moreover, a similarity measure is applied to determine capable developer based on final ranking. This process fetches the appropriate developer from existing system to fix the bugs and allocated to top ranked developer for its solution. If top ranked developer is not free, then it will be allocated to next one in the list. This process is continued until bugs are not fixed.

5.3 Results and Discussion

This section describes the simulation results of proposed bug assignment model. The proposed model is implemented using JAVA programming language. The performance of the proposed model is tested using four datasets. These datasets are Eclipse, Firefox, JBoss, OPENFoam and Mozilla. The details of these datasets are presented in Table 5.1. Further, the recall, precision, F-measure and accuracy parameters are considered to determine the severity levels of bugs [119]. Whereas, recall@5 and recall@10 parameters are used to measure the performance of proposed bug assignment model [131]. Hence, in this work, two experiment are conducted and these experiments are listed as

- Experiment 1: Bugs Severity Level Prediction
- Experiment 2: Identification of Relevant Developer for Fixing Bugs

Table 5.1: Details of bug reports considered for experiment [65].

Dataset	No. of Bugs	BTS	Severity Level
Mozilla	539	Bugzilla	7
Eclipse	693	Bugzilla	5
JBoss	573	Redhat Bugzilla	5
OpenFOAM	795	Manits	8
Firefox	620	Bugzilla	7

5.3.1 Experiment 1: Results and Discussion

This subsection presents the simulation results of bugs severity level prediction. To predict the bug severity levels, ACO-NB and ACO-SVM approaches are used. Initially, relevant features

of bugs reports are identified through ACO based feature weighting technique and further, SVM and NB machine learning classifiers are used to predict the bug severity level. Simulation results of these approaches are evaluated using accuracy, F-measure, recall and precision parameters and compared with SVM and NB approaches. Table 5.2 presents the simulation results of bug severity level prediction using five well known datasets. These datasets are Eclipse, Mozilla, JBoss, OpenFOAM and Firefox. It is observed that ACO-NB approach provides better results than ACO-SVM, SVM and NB approaches using all datasets. It is also noticed that SVM approach exhibits worst results for most of datasets. On other side, it is stated that ACO-SVM provides more accurate bug severity prediction in comparison to SVM and NB approaches. Hence, it is concluded that ACO based feature selection technique improves the simulation results of SVM and NB considerably.

Table 5.2: Comparison of proposed bug assignment model, NB and SVM Classifiers for all five datasets

Approaches	Datasets	Performance Parameters			
		Precision (%)	Recall (%)	F-measure (%)	Accuracy (%)
ACO-NB	Mozilla	78.75	77.51	78.61	80
	Eclipse	81.05	77.43	79.69	79.6
	JBoss	80.85	79.23	80.52	79.6
	OpenFOAM	77.59	76.85	77.71	78.25
	Firefox	77.04	77.22	77.62	79
ACO-SVM	Mozilla	75.71	77.09	76	76.21
	Eclipse	73.33	72.08	72.53	74.72
	JBoss	73.65	72.56	72.62	75.24
	OpenFOAM	76.46	73.33	74.52	74.52
	Firefox	71.78	68.01	68.78	71.78
NB	Mozilla	74.32	71.79	73.52	75.85
	Eclipse	72.85	71.22	72.52	73.4
	JBoss	74.05	70.42	72.69	74.8
	OpenFOAM	73.71	72.24	73.45	74.5
	Firefox	73.03	72.08	73.05	75.57
SVM	Mozilla	72.62	75.51	73.57	73.67
	Eclipse	73.56	68.73	70.89	71.67
	JBoss	73.95	69.12	70.91	71.67
	OpenFOAM	74.46	72.65	73.14	71.87
	Firefox	69.22	67.22	67.38	65.34

Figure 5.4 shows the simulation results of ACO-NB, ACO-SVM, NB and SVM using precision, recall, F-measure and accuracy parameters for Eclipse dataset. It is stated that ACO-NB

approach provides more significant results than other approaches being compared. It is also observed that SVM exhibits worst performance among all approaches. The simulation results of bug severity level prediction using Firefox dataset are presented in Figure 5.5. It is seen that ACO-NB approach obtains better result for severity levels prediction as compared to ACO-SVM, NB and SVM approaches. It is also revealed that NB approach gives more accurate results than ACO-SVM and SVM approaches due to linearity of data. It is noticed that if data is linear separable, then the performance of SVM classifier is not up to mark.

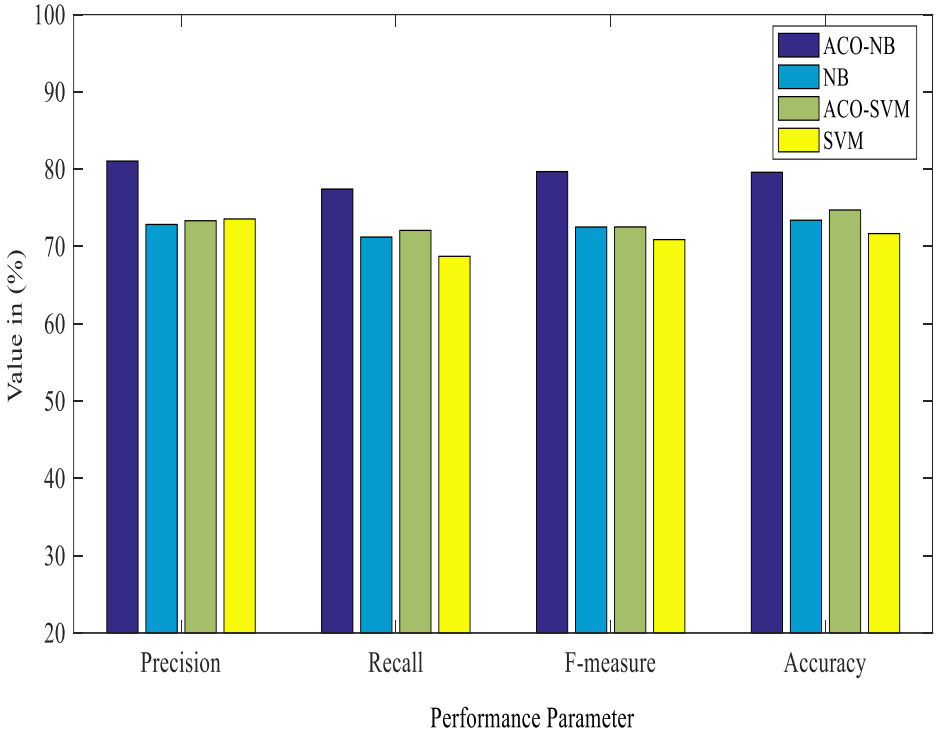


Figure 5.4: Bug severity level prediction using ACO-NB, NB, ACO-SVM and SVM approaches using Eclipse dataset

Figures 5.6 presents the simulation results of ACO-NB, NB, ACO-SVM and SVM approaches using OpenFOAM dataset. It is noticed that ACO-NB outperforms than other approaches being compared. It is also observed that SVM approach achieves better recall and precision rates as compared to NB approach. While, NB approach provides better result using F-measure and accuracy parameters for OpenFOAM datasets. The simulation results of all approaches for severity levels prediction using JBoss dataset are reported in Figure 5.7. It is stated that ACO-NB provides better results for severity levels prediction as compared to ACO-SVM, NB and SVM approaches. Further, SVM approach exhibits worse results for JBoss dataset using recall,

F-measure and accuracy parameters. It is also observed that precision rate of ACO-SVM, NB and SVM approaches is almost similar.

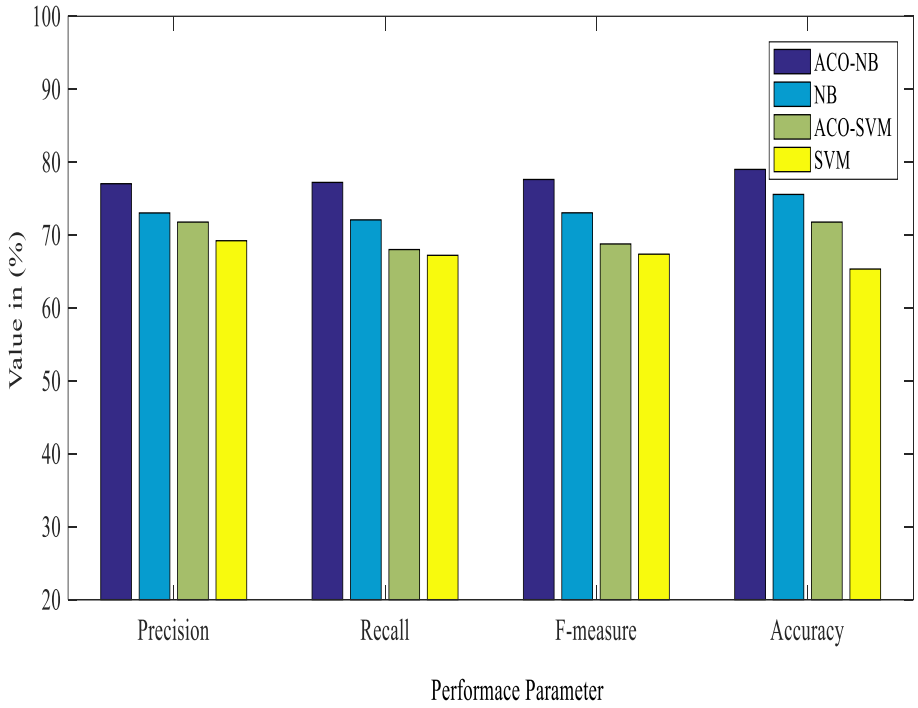


Figure 5.5: Bug severity level prediction using ACO-NB, NB, ACO-SVM and SVM approaches using Firefox dataset

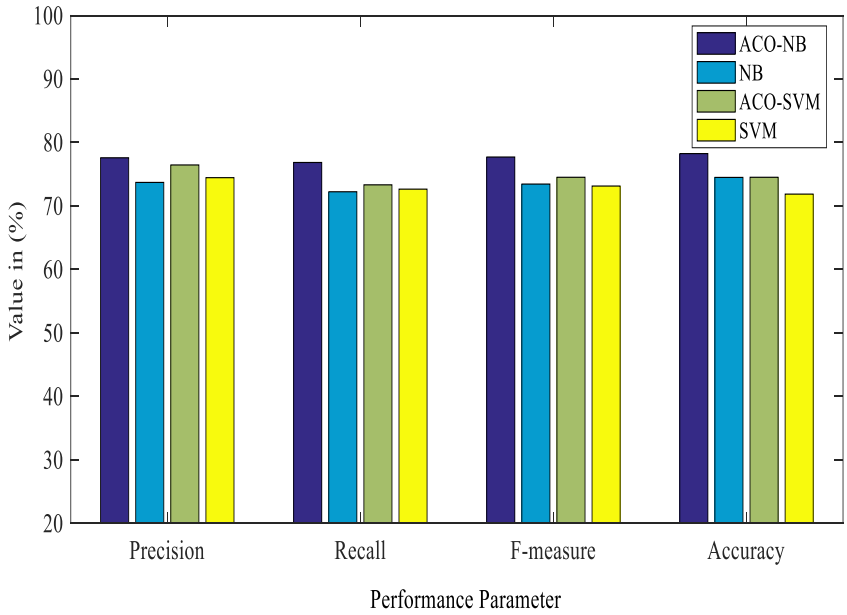


Figure 5.6: Bug severity level prediction using ACO-NB, NB, ACO-SVM and SVM approaches using OpenFOAM dataset

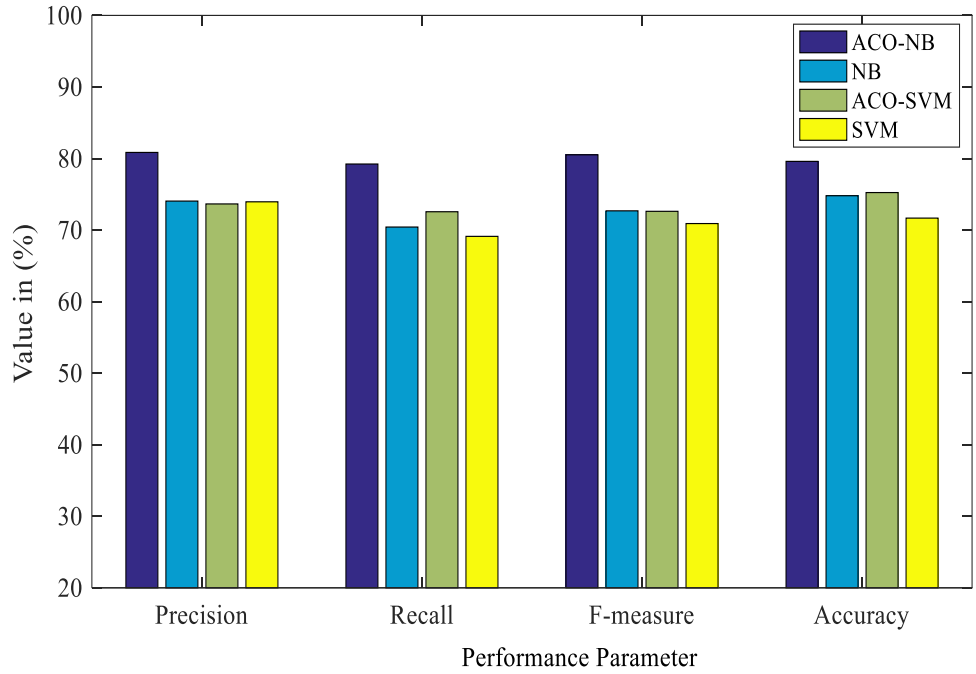


Figure 5.7: Bug severity level prediction using ACO-NB, NB, ACO-SVM and SVM approaches using Jboss dataset

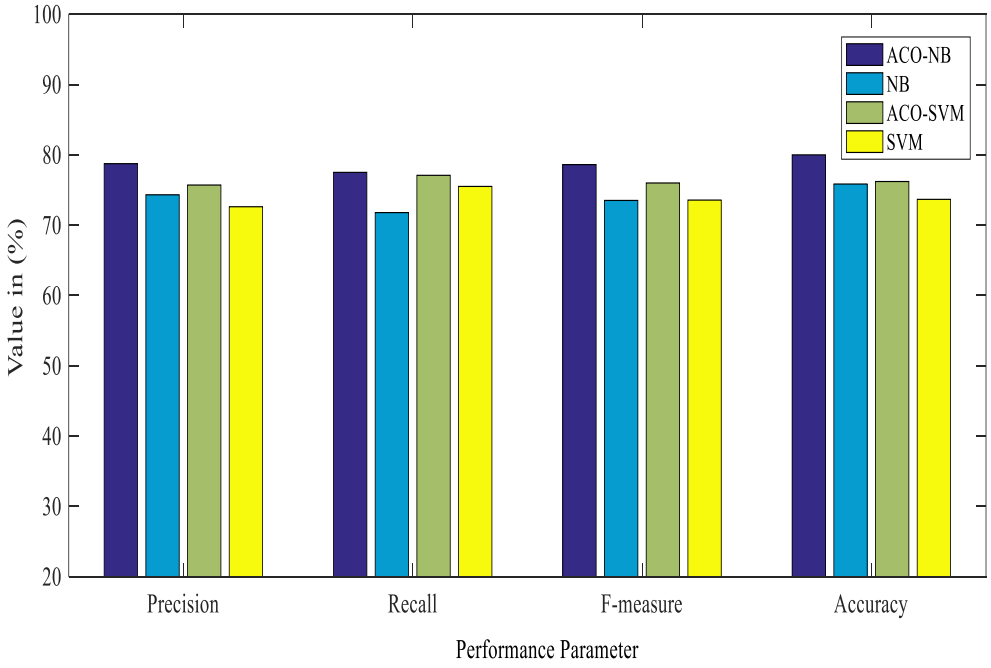


Figure 5.8: Bug severity level prediction using ACO-NB, NB, ACO-SVM and SVM approaches using Mozilla dataset

Figure 5.8 illustrates the results of bug severity level of ACO-NB, ACO-SVM, NB and SVM approaches using Mozilla dataset. It is noted that ACO-NB approach predicts the bugs severity

more accurately in comparison to other approaches. It is observed that SVM exhibits poor performance for Mozilla dataset in terms of precision and accuracy parameters. Whereas, the recall rate of NB approach is worse among all approaches. It is also observed that accuracy rate of ACO-SVM and NB approaches is almost similar. Hence, it can be revealed that ACO-NB approach provides more promising results for bugs severity level prediction.

5.3.2 Experiment 2: Results and Discussion

Experiment 2 corresponds to the identification of appropriate developers for fixing the bugs. To determine the appropriate developer, three metrics are deigned to rank each developer. It is also stated that weightage of severity level metric is higher than other two metrics. The efficacy of proposed DevRE system is tested using five bug datasets. Further, recall@5, recall@10, precision@5 and precision@10 parameters are used to evaluate the performance of proposed system. Table 5.3 presents the simulation results of proposed developer recommendation system. It is observed that proposed recommendation system works efficiently with recall@10 parameter than recall@5. It is also seen that proposed recommendation system achieves higher recall@10 rate i.e. 67.83 for Eclipse dataset, whereas, low recall@10 rate i.e. 51.07 is obtained for Firefox dataset. On the analysis of recall@5, it is noted that proposed recommendation system provides higher recall@5 rate i.e. 52.37 with Mozilla dataset and less recall@5 rate i.e. 37.56 for Firefox dataset. Further, it is noticed that proposed recommendation system gives better precision results using precision@10 than precision@5. It is observed that optimum precision@5 result are obtained for Firefox dataset, whereas Mozilla datasets exhibits worst precision results among all datasets. It is also seen that proposed system provides similar precision@10 rate i.e. 13.69 and 13.83 for Eclipse and Mozilla datasets respectively. The higher recall@10 rate obtains for Firefox dataset i.e. 18.45 among all datasets. The results of proposed developer identification system are also compared with other existing recommendation systems using Eclipse and Firefox datasets. These systems are ReComm, Drex-Frequency, Drex-Outdegree, Bugzie and Dretom [132,86,85,89]. Drex recommendation system is based on k-nearest-neighbor search and expertise ranking [86]. Bugzie recommender system is based on fuzzy set theory and cache method. It is developed for bug triage [85]. Dretom recommendation system is based on topic models and designed for bug resolutions [89]. The results are compared using recall and improvement parameters.

Table 5.3: Simulation results of proposed DevRE system using different bug datasets

Dataset	Parameter			
	Recall@5	Recall@10	Precision@5	Precision@10
Eclipse	49.68	67.83	19.34	13.69
Firefox	37.56	51.07	24.73	18.45
JBoss	41.24	54.67	18.27	15.53
OPENFoam	47.56	58.98	21.78	17.69
Mozilla	52.37	61.14	17.06	13.83

Table 5.4 illustrates the results of proposed recommendation system and other existing systems using recall@5 and improvement (imp) parameters. It is stated that proposed recommendation system obtains better results than other existing systems. The recall@5 rate of Eclipse and Firefox datasets are 49.68 and 37.36 respectively. It is also revealed that Dretom system exhibits poor performance among all recommendation systems. Further, it is also noticed that proposed DevRE recommendation system improves the simulation results significantly.

Table 5.4: Simulation results of proposed DevRE system and other existing recommendation systems using Eclipse and Firefox.

Dataset	Parameter	Eclipse	Firefox
Proposed DevRe	recall@5	49.68	37.56
ReComm	recall@5	41.95	26.99
	imp(%)	18.43	39.16
Drex-Frequency	recall@5	37.36	22.95
	imp(%)	32.98	63.66
Drex-Outdegree	recall@5	36.24	22.03
	imp(%)	37.09	70.49
Bugzie	recall@5	17.08	14.87
	imp(%)	190.87	152.59
Dretom	recall@5	14.7	14.89
	imp(%)	237.96	152.25

The simulation results of proposed DevRE system and other existing recommendation systems using recall@10 and improvement (imp) are reported in Table 5.5. The proposed DevRE system achieves higher recall@10 rate i.e. 67.83 and 51.07 for Eclipse and Firefox datasets as compared to rest of recommendation systems. It is also stated that Dretom system obtains poor results

among all compared recommendation systems. It is concluded that proposed DevRE system significantly improves the bug assignment recommendation rate.

Table 5.5: Simulation results of proposed DevRE system and other existing recommendation systems using Eclipse and Firefox.

Dataset	Parameter	Eclipse	Firefox
Proposed DevRe	recall@10	67.83	51.07
ReComm	recall@10	56.41	40.63
	imp(%)	20.24	25.7
Drex-Frequency	recall@10	47.98	35.56
	imp(%)	41.37	43.62
Drex-Outdegree	recall@10	47.75	35.95
	imp(%)	42.05	42.06
Bugzie	recall@10	30.39	25.02
	imp(%)	123.2	104.12
Dretom	recall@10	24.45	24.99
	imp(%)	177.42	104.36

Table 5.6 illustrates the simulation results of proposed DevRE system and other recommendation systems using precision@5 and improvement (imp) parameters. It is observed that proposed system gives better precision results than other existing recommendation systems using both of datasets.

Table 5.6: Simulation results of proposed DevRE system and other existing recommendation systems using Eclipse and Firefox.

Dataset	Parameter	Eclipse	Firefox
Proposed DevRe	precision@5	19.34	24.73
ReComm	precision@5	16.91	21.17
	imp(%)	14.37	16.82
Drex-Frequency	precision@5	16.79	20.87
	imp(%)	15.19	18.5
Drex-Outdegree	precision@5	15.27	20.21
	imp(%)	26.65	22.37
Bugzie	precision@5	9.8	14.44
	imp(%)	97.35	71.26
Dretom	precision@5	7.97	14.18
	imp(%)	142.66	74.4

But, the proposed system obtains better precision@5 results i.e. 24.73 for Firefox dataset rather than Eclipse dataset i.e. 19.34. The worse precision@5 results are reported by Dretom and Bugzie systems i.e. 14.18 and 14.44. Significant improvement is reported in precision@5 rate using proposed system.

The simulation results of proposed system and other recommendation systems are reported in Table 5.7. There is the significant difference between the performance of proposed DevRE system and other recommendation systems using precision@5 and precision@10 parameters. It is stated that proposed system provides better quality results than other existing system. Whereas, Dretom system exhibits worse performance among all datasets.

Table 5.7: Simulation results of proposed DevRE system and other existing recommendation systems using Eclipse and Firefox.

Dataset	Parameter	Eclipse	Firefox
Proposed DevRe	precision@10	13.69	18.45
ReComm	precision@10	11.82	16.34
	imp(%)	15.82	12.91
Drex-Frequency	precision@10	10.23	15.16
	imp(%)	33.82	21.7
Drex-Outdegree	precision@10	10.17	15
	imp(%)	34.61	23
Bugzie	precision@10	8.69	12.24
	imp(%)	57.54	50.74
Dretom	precision@10	6.68	11.97
	imp(%)	104.94	54.14

The performance of proposed DevRE recommendation system is also compared with some machine learning based recommendation systems. The accuracy parameter is considered to evaluate the performance of these systems. The performance of these recommendation systems is tested on Eclipse, Firefox, OPENFoam and Mozilla datasets. Table 5.8 presents the simulation results of proposed DevRE system, NB, SVM, and C4.5 based recommendation systems [97]. It is observed that proposed system obtains better quality results than other recommendation systems. The proposed DevRE system obtains higher accuracy rate i.e. 80.16 for Eclipse dataset

and low accuracy rate i.e. 76.43 for Mozilla dataset. It is also seen that NB based recommender system obtains worse results for Eclipse and Mozilla dataset, whereas, C4.5 provides worst results for Firefox and OPENFoam datasets. Hence, it is concluded that proposed DevRE system provides more effective and efficient results for bug recommendation as compared to existing ones.

Table 5.8: Simulation results of proposed DevRE system and other existing machine learning based recommendation system using accuracy parameter

Dataset	NB	SVM	C4.5	Proposed DevRe
Eclipse	65.22	77.21	73.55	80.16
Firefox	70.93	76.4	65.33	78.04
OPENFoam	67.95	77.44	66.62	79.74
Mozilla	61.41	72.9	69.31	76.43

5.4 Summary

In this chapter, a Developer Recommendation (DevRE) System is proposed for assigning the bugs to relevant developers. The proposed DevRE system consists of four phases. In the proposed model, ACO based feature weighting technique is implemented to determine the relevant features for severity prediction. Moreover, three metrics i.e. capability ranking, bug severity level and average bug fixing time are designed to determine appropriate developer for bug fixing. Further, NB and SVM classifiers are applied to measure the bugs severity level. Two experiments are considered to evaluate the performance of proposed bug assignment model. In first experiment, bugs severity levels are predicted through ACO-NB and ACO-SVM techniques. It is noticed that ACO-NB techniques provides more effective results for prediction of bugs severity levels. In second experiment, several existing recommendation systems are considered to compare the performance of proposed DevRE system for bug assignment. It is observed that proposed DevRe system provides more accurate results than other recommendation systems.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE OF WORK

In this thesis, the bug resolution techniques are studied and categorized. The aim of this work is to overcome the data dimensionality, sparsity and estimation issues of bug reports and also improve the solution quality of bug resolution process. Thus, the primary objectives of the thesis are to implement domain specific bug report summarization technique, improve the accuracy of bug severity classification model and develop a model for bug assignment. To address the above mentioned issues, three different models are developed for bug report summarization, bug severity classification and bug assignment. The entire work is categorized into three chapters.

In chapter three, a PSO-ACO based bug report summarization technique is developed to handle bug summarization issue. The aim of proposed technique is to determine the optimum subset of summary. In this chapter, a PSO-ACO based summary subset selection algorithm is proposed. The simulations results showed that PSO-ACO based algorithm provides better results for summarization task. It is also observed that combination PSO and ACO enhances the simulation results of PSO technique in efficient manner.

The chapter four of thesis addresses the accuracy issue of bug severity classification models. A bug severity model based on deep learning is proposed for severity classification, called BCR. In BCR model, CN method is also integrated to determine relevant set of features for severity classification. The performance of BCR model is compared with Zhou et al. model for binary bug severity classification. It was showed that BCR model provides better results.

Chapter five deals with bug fixing issue of bug BRP. In this chapter, a developer recommendation (DevRE) system is proposed to identify the relevant set of developers for bug fixing. The appropriate developer for fixing the bugs is identified through three metrics. The proposed DevRE system also integrates the ACO based feature weighting technique. The aim of ACO technique is to determine optimal features for severity prediction. Whereas, NB and SVM techniques are applied for classification of severity data. Further, severity classes are used to determine relevant developers for bug fixing.

6.1 Future Scope

This thesis addresses the three issues of BRP process i.e. bug report summarization, severity classification and bug assignment. To determine the optimal set of features, ACO and PSO

techniques is applied on bug report dataset. In future, some other meta-heuristic algorithms like ABC, Cuckoo search, Bat algorithm will be explored to determine optimal features. Further, NB, SVM and RFB classifiers are used to predict the severity classes and bug fixing. In future, some ensemble classifiers and meta-heuristic techniques will be considered for severity prediction and bug fixing. Furthermore, in future, bug duplicate detection, bug localization and patch generation issues will be taken into consideration and proposed new models to address these issues.

REFERENCES

[1] Charette, R.N., Why software fails [software failure]. *IEEE spectrum*, *42*(9), pp.42-49, 2005.

[2] Erlikh, L., Leveraging legacy system dollars for e-business. *IT professional*, *2*(3), pp.17-23, 2000.

[3] Zhang, T., Jiang, H., Luo, X., & Chan, A. T. A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal, 59*(5), 741-773, 2016.

[4] Aljedaani, W., &Javed, Y. Bug Reports Evolution in Open Source Systems. In *5th International Symposium on Data Mining Applications* (pp. 63-73). Springer, Cham, 2018.

[5] Angel, T. S., Kumar, G. S., Sehgal, V. M., & Nayak, G. Effective Bug Processing and Tracking System. *Journal of Computational and Theoretical Nanoscience, 15*(8), 2604-2606, 2018.

[6] Eclipse. http://eclipse.org/.

[7] Nazar, N., Hu, Y., & Jiang, H. Summarizing software artifacts: A literature review. *Journal of Computer Science and Technology*, *31*(5), 883-909, 2016.

[8] Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., &Guéhéneuc, Y. G. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON* (Vol. 8, pp. 304-318), 2008

[9] Thomas, S. W., Nagappan, M., Blostein, D., & Hassan, A. E. The impact of classifier configuration and classifier combination on bug localization. *IEEE Transactions on Software Engineering*, *39*(10), 1427-1443, 2013.

[10] Uddin, J., Ghazali, R., Deris, M. M., Naseem, R., & Shah, H. (2017). A survey on bug prioritization. *Artificial Intelligence Review*, *47*(2), 145-180.

[11] Jadhav, A., Jadhav, K., Bhalerao, A., & Kharade, A. A Survey on Software Data Reduction Techniques for Effective Bug Triage. *IJCSIT on Computer Science and Information Technologies*, *6*(5), 4611-4612, 2015.

[12] Akila, V., Zayaraz, G., & Govindasamy, V. Bug triage in open source systems: a review. *International Journal of Collaborative Enterprise*, *4*(4), 299-319, 2014.

[13] Strate, J. D., & Laplante, P. A. A literature review of research in software defect reporting. *IEEE Transactions on Reliability*, *62*(2), 444-454, 2013.

[14] Anvik, J., Hiew, L. and Murphy, G.C., Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering* (pp. 361-370). ACM, May. 2006.

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

[16] Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S., Extracting structural information from bug reports. In *Proceedings of the 2008 international working conference on Mining software repositories* (pp. 27-30). ACM, May.2008.

[17] Breu, S., Premraj, R., Sillito, J. and Zimmermann, T., Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work* (pp. 301-310). ACM, February.2010.

[18] Dit, B. and Marcus, A., Improving the readability of defect reports. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering* (pp. 47-49). ACM, November. 2008.

[19] Gasser, L. and Ripoche, G., Distributed collective practices and free/open-source software problem management: perspectives and methods. In *2003 Conference on Cooperation, Innovation & Technologie (CITE'03)(Université de Technologie de*, 2003.

[20] Haiduc, S., Aponte, J., Moreno, L. and Marcus, A., On the use of automated text summarization techniques for summarizing source code. In *2010 17th Working Conference on Reverse Engineering* (pp. 35-44). IEEE, October. 2010.

[21] Hamou-Lhadj, A. and Lethbridge, T., Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In *14th IEEE International Conference on Program Comprehension (ICPC'06)* (pp. 181-190). IEEE, June. 2006.

[22] Murray, G. and Carenini, G., Summarizing spoken and written conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 773-782). Association for Computational Linguistics, October. 2008.

[23] Nenkova, A. and Louis, A., Can you summarize this? Identifying correlates of input difficulty for generic multi-document summarization, 2008.

[24] Rastkar, S., Murphy, G.C. and Murray, G., Summarizing software artifacts: a case study of bug reports. In *2010 ACM/IEEE 32nd International Conference on Software Engineering* (Vol. 1, pp. 505-514). IEEE, May. 2010.

[25] Rastkar, S., Murphy, G.C. and Murray, G., Automatic summarization of bug reports. *IEEE Transactions on Software Engineering*, *40*(4), pp.366-380, 2014.

[26] Jiang, H., Li, X., Ren, Z., Xuan, J. and Jin, Z., Toward Better Summarizing Bug Reports with Crowdsourcing Elicited Attributes. *IEEE Transactions on Reliability*, *68*(1), pp.2-22, 2018.

[27] Huai, B., Li, W., Wu, Q. and Wang, M., Mining Intentions to Improve Bug Report Summarization. In *SEKE* (pp. 320-319), 2018.

[28] Ponzanelli, L., Mocci, A. and Lanza, M., Summarizing complex development artifacts by mining heterogeneous data. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (pp. 401-405). IEEE Press, May. 2015.

[29] Erkan, G. and Radev, D.R., Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, *22*, pp.457-479, 2004.

[30] Yeasmin, S., Roy, C.K. and Schneider, K.A., Interactive visualization of bug reports using topic evolution and extractive summaries. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 421-425). IEEE, September. 2014.

[31] Mani, S., Catherine, R., Sinha, V.S. and Dubey, A., November. Ausum: approach for unsupervised bug report summarization. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (p. 11). ACM, 2012.

[32] Lotufo, R., Malik, Z. and Czarnecki, K., Modelling the 'hurried'bug report reading process to summarize bug reports. *Empirical Software Engineering*, *20*(2), pp.516-548, 2015.

[33] Nithya, R. and Arunkumar, A., SUMMARIZATION OF BUG REPORTS USING FEATURE EXTRACTION, 2016

[34] Jiang, H., Nazar, N., Zhang, J., Zhang, T. and Ren, Z., Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates. *International Journal of Software Engineering and Knowledge Engineering*, *27*(06), pp.869-896, 2017.

[35] Ferreira, I., Cirilo, E., Vieira, V. and Mourao, F., September. Bug report summarization: an evaluation of ranking techniques. In *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)* (pp. 101-110). IEEE, 2016.

[36] Li, X., Jiang, H., Liu, D., Ren, Z. and Li, G., Unsupervised deep bug report summarization. In *Proceedings of the 26th Conference on Program Comprehension* (pp. 144-155). ACM, May. 2018.

[37] Bhattacharya, P. and Neamtiu, I., Bug-fix time prediction models: can we do better?. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (pp. 207-210). ACM, May. 2011.

[38] Giger, E., Pinzger, M. and Gall, H., Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering* (pp. 52-56). ACM, May. 2010.

[39] Kim, S., Zimmermann, T., Whitehead Jr, E.J. and Zeller, A., Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering* (pp. 489-498). IEEE Computer Society, May. 2007.

[40] Knab, P., Pinzger, M. and Bernstein, A., Predicting defect densities in source code files with decision tree learners. In *Proceedings of the 2006 international workshop on Mining software repositories* (pp. 119-125). ACM, May. 2006.

[41] Kochhar, P.S., Thung, F. and Lo, D., Automatic fine-grained issue report reclassification. In *2014 19th International Conference on Engineering of Complex Computer Systems* (pp. 126-135). IEEE, August. 2014.

[42] Ostrand, T.J., Weyuker, E.J. and Bell, R.M., Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, *31*(4), pp.340-355, 2005.

[43] D'Ambros, M., Lanza, M. and Robbes, R., An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (pp. 31-41). IEEE, May. 2010.

[44] D'Ambros, M., Gall, H., Lanza, M. and Pinzger, M., Analysing software repositories to understand software evolution. In *Software evolution* (pp. 37-67). Springer, Berlin, Heidelberg, 2008.

[45] Herzig, K., Just, S. and Zeller, A., It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 international conference on software engineering* (pp. 392-401). IEEE Press, May. 2013.

[46] Kim, S., Zhang, H., Wu, R. and Gong, L., Dealing with noise in defect prediction. In *2011 33rd International Conference on Software Engineering (ICSE)* (pp. 481-490). IEEE, May. 2011.

[47] Zeller, A., Can We Trust Software Repositories?. In *Perspectives on the Future of Software Engineering* (pp. 209-215). Springer, Berlin, Heidelberg, 2013.

[48] Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V. and Devanbu, P., Fair and balanced?: bias in bug-fix datasets. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (pp. 121-130). ACM, August. 2009.

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

[50] Lamkanfi, A., Demeyer, S., Soetens, Q. D., &Verdonck, T., Comparing mining algorithms for predicting the severity of a reported bug. In *2011 15th European Conference on Software Maintenance and Reengineering* (pp. 249-258). IEEE, 2011.

[51] Gegick, M., Rotella, P., &Xie, T., Identifying security bug reports via text mining: An industrial case study. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (pp. 11-20). IEEE, 2010

[52] Chaturvedi, K. K., & Singh, V. B., Determining bug severity using machine learning techniques. In *2012 CSI Sixth International Conference on Software Engineering (CONSEG)* (pp. 1-6). IEEE, September. 2012.

[53] Thung, F., Lo, D., & Jiang, L. Automatic defect categorization. In *2012 19th Working Conference on Reverse Engineering* (pp. 205-214). IEEE, October.2012.

[54] Yang, C. Z., Hou, C. C., Kao, W. C., & Chen, X., An empirical study on improving severity prediction of defect reports using feature selection. In *2012 19th Asia-Pacific Software Engineering Conference* (Vol. 1, pp. 240-249). IEEE, 2012.

[55] Somasundaram, K., & Murphy, G. C., Automatic categorization of bug reports using latent dirichlet allocation. In *Proceedings of the 5th India software engineering conference* (pp. 125-130). ACM, Februray.2012.

[56] Bhattacharya, P., Iliofotou, M., Neamtiu, I., &Faloutsos, M. Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 419-429). IEEE, 2012.

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

[58] Nagwani, N. K., Verma, S., & Mehta, K. K., Generating taxonomic terms for software bug classification by utilizing topic models based on Latent Dirichlet Allocation. In *2013 Eleventh International Conference on ICT and Knowledge Engineering* (pp. 1-5). IEEE, 2013.

[59] Roy, N. K. S., & Rossi, B., Towards an improvement of bug severity classification. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 269-276). IEEE, 2014.

[60] Limsettho, N., Hata, H., Monden, A., & Matsumoto, K., Automatic unsupervised bug report categorization. In *2014 6th International Workshop on Empirical Software Engineering in Practice* (pp. 7-12). IEEE, November. 2014.

[61] Chawla, I., & Singh, S. K., An automated approach for bug categorization using fuzzy logic. In *Proceedings of the 8th India Software Engineering Conference* (pp. 90-99). ACM, 2015.

[62] Zhang, T., Yang, G., Lee, B., & Chan, A. T., Predicting severity of bug report by mining bug repository with concept profile. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (pp. 1553-1558). ACM, April.2015.

[63] Sharma, G., Sharma, S., &Gujral, S., A novel way of assessing software bug severity using dictionary of critical terms. *Procedia Computer Science*, *70*, 632-639,2015.

[64] Gujral, S., Sharma, G., & Sharma, S., Classifying bug severity using dictionary based approach. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)* (pp. 599-602). IEEE, February. 2015

[65] Zhou, Y., Tong, Y., Gu, R., & Gall, H., Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, *28*(3), 150-176, 2016

[66] Jin, K., Dashbalbar, A., Yang, G., Lee, J. W., & Lee, B., Bug severity prediction by classifying normal bugs with text and meta-field information. *Advanced Science and Technology Letters*, *129*, 19-24, 2016.

[67] Pandey, N., Sanyal, D. K., Hudait, A., & Sen, A., Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, *13*(4), 279-297, 2017.

[68] Jindal, R., Malhotra, R., & Jain, A., Prediction of defect severity by mining software project reports. *International Journal of System Assurance Engineering and Management*, *8*(2), 334-351, 2017.

[69] Singh, V. B., Misra, S., & Sharma, M., Bug severity assessment in cross project context and identifying training candidates. *Journal of Information & Knowledge Management*, *16*(01), 1750005, 2017.

[70] Sharmin, S., Aktar, F., Ali, A. A., Khan, M. A. H., &Shoyaib, M., Bfsp: A feature selection method for bug severity classification. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pp. 750-754, IEEE, 2017.

[71] Kumari, M., Sharma, M., & Singh, V. B., Severity assessment of a reported bug by considering its uncertainty and irregular state. *International Journal of Open Source Software and Processes (IJOSSP)*, *9*(4), 20-46, 2018.

[72] Menzies, T., &Marcus, A., Automated severity assessment of software bug reports. In IEEE International Conference on Software Maintenance ICSM 2008, pp.346-355, IEEE, September. 2008.

[73] Nguyen T.T., Nguyen A.T. and Nguyen T.N., Topic- based, Time-aware Bug Assignment. ACM SIGSOFT Software Engineering Notes, 39(1): 1-4, 2014.

[74] Hassan, A.E. and Xie, T., Software intelligence: the future of mining software engineering data. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (pp. 161-166). ACM, November.2010.

[75] Xie, T., Zimmermann, T. and Van Deursen, A., Introduction to the Special Issue on Mining Software Repositories in 2010. Empirical Software Engineering, 17(4): 500-502, 2010.

[76] Bettenburg, N., Nagappan, M. and Hassan, A.E., Towards improving statistical modeling of software engineering data: think locally, act globally!. *Empirical Software Engineering*, *20*(2), pp.294-335, 2015.

[77] Guo P.J., Zimmermann T., Nagappan N. and Murphy B., not my Bug! and Other Reasons for Software Bug Report Reassignments. Preceding of the ACM 2011 conference on Computer supports cooperative work, 395-404, 2011.

[78] Jeong, G., Kim, S. and Zimmermann, T., Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 111-120, ACM, August. 2009.

[79] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes and P. Baldi, "Mining Eclipse Developer Contributions via Author-Topic Models," Proc of International Workshop on Mining Software Repositories, 2007.

[80] David M. Blei, Andrew Y. Ng and Michael I. Jordan, "Latent Dirichlet Allocation," Journal of Machine Learning Research, Vol. 3, pp. 993-1022, 2003.

[81] Xuan, J., Jiang, H., Ren, Z., Yan, J. and Luo, Z., Automatic bug triage using semi-supervised text classification. arXiv preprint arXiv:1704.04769, 2017.

[82] Bhattacharya, P. and Neamtiu, I., Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-10). IEEE, September. 2010.

[83] Anvik, J., & Murphy, G. C., Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM), 20*(3), 10, 2011.

[84] Zou, W., Hu, Y., Xuan, J., & Jiang, H., Towards training set reduction for bug triage. In *2011 IEEE 35th Annual Computer Software and Applications Conference* (pp. 576-581). IEEE, July. 2011.

[85] Tamrawi, A., Nguyen, T. T., Al-Kofahi, J. M., & Nguyen, T. N., Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 365-375). ACM, September. 2011.

[86] Wu, W., Zhang, W., Yang, Y., & Wang, Q., Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference* (pp. 389-396). IEEE, December.2011.

[87] Servant, F., & Jones, J. A., WhoseFault: automatic developer-to-fault assignment through fault localization. In *2012 34th International conference on software engineering (ICSE)*, pp. 36-46, IEEE, June. 2012.

[88] Xuan, J., Jiang, H., Ren, Z., & Zou, W., Developer prioritization in bug repositories. In 2012 34th International Conference on Software Engineering (ICSE), pp. 25-35, IEEE, June.2012.

[89] Xie, X., Zhang, W., Yang, Y., & Wang, Q, Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*, pp. 19-28, ACM, September. 2012.

[90] Bhattacharya, P., Iliofotou, M., Neamtiu, I., &Faloutsos, M., Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*, pp. 419-429, IEEE, June. 2012.

[91] Xia, X., Lo, D., Wang, X., & Zhou, B., Accurate developer recommendation for bug resolution. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 72-81, IEEE, 2013.

[92] Naguib, H., Narayan, N., Brügge, B., &Helal, D., Bug report assignee recommendation using activity profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 22-30, IEEE Press, May.2013.

[93] Zhang, T., Yang, G., Lee, B., & Lua, E. K., A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1, pp. 223-230, IEEE, 2014.

[94] Yang, G., Zhang, T., & Lee, B. (2014, July). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014*

IEEE 38th Annual Computer Software and Applications Conference, pp. 97-106, IEEE, July.2014.

[95] Zhang, W., Wang, S., & Wang, Q., KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Information and Software Technology*, *70*, 68-84, 2016.

[96] Liu, J., Tian, Y., Yu, X., Yang, Z., Jia, X., Ma, C., & Xu, Z., A multi-source approach for bug triage. *International Journal of Software Engineering and Knowledge Engineering*, *26*(09n10), 1593-1604, 2016.

[97] Yadav, A., Singh, S. K., & Suri, J. S., Ranking of software developers based on expertise score for bug triaging. *Information and Software Technology*, *112*, 1-17, 2019.

[98] Kanwal, J., & Maqbool, O., Bug prioritization to facilitate bug report triage. Journal of Computer Science and Technology, 27(2), 397-412, 2012.

[99] Changnon, S. (Ed.)., *METROMEX: A review and summary*, Vol. 18, Springer, 2016.

[100] Mosa, M. A., Hamouda, A., & Marei, M., Graph coloring and ACO based summarization for social networks. *Expert Systems with Applications*, *74*, 115-126, 2016.

[101] Mosa, M. A., Anwar, A. S., & Hamouda, A., A survey of multiple types of text summarization with their satellite contents based on swarm intelligence optimization algorithms. *Knowledge-Based Systems*, *163*, 518-532, 2019.

[102] Mosa, M. A., Hamouda, A., & Marei, M., Ant colony heuristic for user-contributed comments summarization. *Knowledge-Based Systems*, *118*, 105-114, 2017.

[103] Al-Abdallah, R. Z., & Al-Taani, A. T., Arabic single-document text summarization using particle swarm optimization algorithm. *Procedia Computer Science*, *117*, 30-37, 2017.

[104] Mandal, S., Singh, G. K., & Pal, A., PSO-Based Text Summarization Approach Using Sentiment Analysis. In *Computing, Communication and Signal Processing*, pp. 845-854, Springer, Singapore, 2019.

[105] Meena, Y. K., & Gopalani, D. Evolutionary algorithms for extractive automatic text summarization. *Procedia Computer Science*, *48*, 244-249, 2015.

[106] Shelokar, P. S., Siarry, P., Jayaraman, V. K., & Kulkarni, B. D., Particle swarm and ant colony algorithms hybridized for improved continuous optimization. *Applied mathematics and computation*, *188*(1), 129-142, 2007.

[107] Kıran, M. S., Özceylan, E., Gündüz, M., &Paksoy, T., A novel hybrid approach based on particle swarm optimization and ant colony algorithm to forecast energy demand of Turkey. *Energy conversion and management*, *53*(1), 75-83, 2012.

[108] Liang, J., Koperski, K., Dhillon, N. S., Tusk, C., & Bhatti, S., *U.S. Patent No. 8,594,996*. Washington, DC: U.S. Patent and Trademark Office, 2013.

[109] Denny, M. J., & Spirling, A., Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. *Political Analysis*, *26*(2), 168-189, 2018.

[110] Vijayarani, S., Ilamathi, M. J., & Nithya, M., Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, *5*(1), 7-16, 2015.

[111] Verberne, S., Sappelli, M., Hiemstra, D., &Kraaij, W., Evaluation and analysis of term scoring methods for term extraction. *Information Retrieval Journal*, *19*(5), 510-545, 2016.

[112] Van Erven, T., &Harremos, P., Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory*, *60*(7), 3797-3820, 2014.

[113] Lin, C. Y., Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74-81, 2004.

[114] Rautray, R., Balabantaray, R. C., & Bhardwaj, A., Document summarization using sentence features. *International Journal of Information Retrieval Research (IJIRR)*, *5*(1), 36-47, 2015.

[115] Wei, Y., Zhao, Y., Lu, C., Wei, S., Liu, L., Zhu, Z., & Yan, S., Cross-modal retrieval with CNN visual features: A new baseline. *IEEE transactions on cybernetics*, *47*(2), 449-460, 2016.

[116] Wang, R., Zhao, H., Lu, B. L., Utiyama, M., &Sumita, E., Bilingual continuous-space language model growing for statistical machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(7), 1209-1220, 2015.

[117] Xia, X., Togneri, R., Sohel, F. and Huang, D., Random forest classification based acoustic event detection utilizing contextual-information and bottleneck features. Pattern Recognition, 81,pp.1-13, 2018.

[118] Gupta, B., Awasthi, S., Gupta, R., Ram, L., Kumar, P., Prasad, B.R. and Agarwal, S., Taxi travel time prediction using ensemble-based random forest and gradient boosting model. In Advances in Big Data and Cloud Computing , . Springer, Singapore , pp. 63-78, 2018.

[119] Huang, Y.J., Powers, R. and Montelione, G.T., Protein NMR recall, precision, and F-measure scores (RPF scores): structure quality assessment measures based on information retrieval statistics. Journal of the American Chemical Society,127(6), pp.1665-1674, 2005.

[120] Aghdam, M. H., Ghasem-Aghaee, N., & Basiri, M. E., Text feature selection using ant colony optimization. *Expert systems with applications*, *36*(3), 6843-6853, 2009.

[121] Khan, A., Baharudin, B., Lee, L. H., & Khan, K., A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, *1*(1), 4-20, 2010.

[122] Jiang, L., Li, C., Wang, S., & Zhang, L., Deep feature weighting for naive Bayes and its application to text classification. *Engineering Applications of Artificial Intelligence*, *52*, 26-39, 2016.

[123] Mohammad, A. H., Alwada'n, T., & Al-Momani, O., Arabic text categorization using support vector machine, Naïve Bayes and neural network. *GSTF Journal on Computing (JoC)*, *5*(1), 108, 2016.

[124] Onan, A., Korukoğlu, S., & Bulut, H., Ensemble of keyword extraction methods and classifiers in text classification. *Expert Systems with Applications*, *57*, 232-247, 2016.

[125] Dorigo, M., & Blum, C., Ant colony optimization theory: A survey. *Theoretical computer science*, *344*(2-3), 243-278, 2005.

[126] Gao, S., Wang, Y., Cheng, J., Inazumi, Y., & Tang, Z., Ant colony optimization with clustering for solving the dynamic location routing problem. *Applied Mathematics and Computation*, *285*, 149-173, 2016.

[127] Sun, Y., Dong, W., & Chen, Y., An improved routing algorithm based on ant colony optimization in wireless sensor networks. *IEEE communications Letters*, *21*(6), 1317-1320, 2017.

[128] Sama, M., Pellegrini, P., D'Ariano, A., Rodriguez, J., & Pacciarelli, D., Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological*, *85*, 89-108, 2016.

[129] Rashno, A., Sadri, S., & SadeghianNejad, H., An efficient content-based image retrieval with ant colony optimization feature selection schema based on wavelet and color features. In *2015 The International Symposium on Artificial Intelligence and Signal Processing (AISP)*, pp. 59-64, IEEE, March. 2015.

[130] Su, M. C., & Chou, C. H., A modified version of the K-means algorithm with a distance based on cluster symmetry. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6), 674-680, 2001.

[131] Che, G., & Yu, D., Performance Evaluation for Tape Storage Data Recall with T10KD Drive. In *2018 New York Scientific Data Summit (NYSDS)*, pp. 1-6, IEEE, August. 2018.

[132] Xuan, J., Jiang, H., Zhang, H., & Ren, Z. Developer recommendation on bug commenting: a ranking approach for the developer crowd. *Science China Information Sciences*, *60*(7), 072105, 2017.

[133] Kumar, Y., & Sahoo, G. Hybridization of magnetic charge system search and particle swarm optimization for efficient data clustering using neighborhood search strategy. Soft Computing, 19(12), 3621-3645,2015.

[134] Sayed, G. I., Darwish, A., & Hassanien, A. E. Binary Whale Optimization Algorithm and Binary Moth Flame Optimization with Clustering Algorithms for Clinical Breast Cancer Diagnoses. Journal of Classification, 1-31, 2019.

[135] Sahoo, G. A two-step artificial bee colony algorithm for clustering. Neural Computing and Applications, 28(3), 537-551,2017.

PUBLICATIONS FROM THESIS

Journals

Accepted

1. Kukkar, A. and Mohana, R., 2018. Feature Weighting with for Swarm Intelligence Optimization as a Tool for Bug Report Summarization. Journal of Advanced Research in Dynamical and Control Systems, pp. 2122-2133**. (Published, Scopus)**
2. Kukkar, A. and Mohana, R. Bug Report Summarization by using Swarm Intelligence Approaches. Recent Patents on Computer Science. **(Accepted, Scopus)** DOI No:10.2174/2213275912666 181205154129.
3. Kukkar, A., Mohana, R., Nayyar, A., Kim, J., Kang, B. G., & Chilamkurti, N. (2019). A Novel Deep-Learning-Based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting. Sensors, 19(13), 2964. **(SCI, Scopus, Published)**

Communicated

1. Kukkar, A. Mohana, R and Kumar, Y. An Ant Colony Optimization based Developer Recommendation System for Bug Assignment, Applied Intelligence, Springer **(SCI, Communicated)**
2. Kukkar, A. Mohana, R and Kumar, Y. An Ant Colony Optimization Based Feature Weighting Technique for Bug Severity Classification. International Journal of System Assurance Engineering and Management, Springer **(Scopus, Communicated)**

Conferences

1. Kukkar, A. and Mohana, R., 2018. A Supervised Bug Report Classification with Incorporate and Textual Field Knowledge. Procedia Computer Science, Elsevier 132, pp.352-361. **(Published, Scopus)**
2. Kukkar, A. and Mohana, R., 2019. An Optimization Technique for Unsupervised Automatic Extractive Bug Report Summarization. In International Conference on Innovative Computing and Communications (pp. 1-11). Springer, Singapore. **(Published, Scopus)**
3. Kukkar,A. Mohana, R and Kumar, Y., Does bug report summarization help in enhancing the accuracy of bug severity classification?. Procedia Computer Science, Elsevier **(Accepted, Scopus)**