Automated Duplicate Defect Detection for Bug Tracking Systems

Author

Momina Nasar

FALL 2019 - MS-19 (CE) 00000319911

Supervisor

Dr. Wasi Haider

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

FEB 2022

Automated Duplicate Defect Detection for Bug Tracking Systems

Author

Momina Nasar

FALL 2019 - MS-19 (CE) 00000319911

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

MS Computer Engineering

Thesis Supervisor:

Dr. Wasi Haider

Thesis Supervisor's Signature:-_____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

FEB 2022

DECLARATION

I certify that this research project, *"Automated Duplicate Defect Detection for Bug Tracking Systems"* is my own work under Dr. Wasi Haider's supervision. The work has not been submitted to anyone else for review. The use of content from other sources has been acknowledged and referred to appropriately.

_____

Signature of Student

Momina Nasar

FALL 2019 - MS-19 (CE) 00000319911

LANGUAGE CORRECTNESS CERTIFICATE

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

_____

Signature of Student

Momina Nasar

FALL 2019 - MS-19 (CE) 00000319911

_____

Signature of Supervisor

Dr. Wasi Haider

COPYRIGHT STATEMENT

2. Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
3. The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
4. Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

ACKNOWLEDGEMENTS

All praises are to Allah. The favors He has bestowed on me have led to the successful completion of my research work. No hurdle was complex enough not to resolve, and I have eternal gratitude for that.

I sincerely thank my supervisor, Dr. Wasi Haider, for his extreme patience and expert guidance. He played an integral role in keeping my direction focused and ensuring I completed all my milestones promptly. I thank my GEC members, Dr. Usman Akram and Dr. Sajid Gul Khwaja, for filling in any gaps in my methodology and offering their kind help. I am lucky to have had the support of three of the soundest academic minds of the department.

This section is incomplete without mentioning the unwavering support of my family and friends. My parents' blind trust in my capabilities and constant prayers culminated in this achievement. My husband, Bilal, with his continuous encouragement and my siblings, Hamna & Ahmad, with their silent support, pushed me to step out of my comfort zone and perform even better than I had imagined. My best friends, Maryam & Shah Rukh, were my emotional strength and helped out whenever I was technically blocked.

I am indebted to every one of you, along with every other individual who may have played even a small but significant role in helping me complete my research and degree. I am grateful for being allowed to leave a small mark in academia for generations to come.

For Family & Friends

ABSTRACT

A bug tracking system (BTS) keeps track of the status of a software system in real time. The bug report it generates is sent to the software developer or centre for maintenance whenever it identifies an abnormal scenario. The freshly reported defect, on the other hand, could be a repeat in the master report repository of an old issue with a remedy already present. This situation results in an onslaught of replicate reports of bugs, making the software development life cycle difficult to manage. As a result, in the software industry, it is now an essential task to find repeat reports of bugs early. This research proposes a two-tier method based on topic-based clustering done by LDA approach, multimodal representation of text using W2V, FT, GloVe and a measure of unified text similarity utilizing similarities of the Cosine and Euclidean nature to solve this challenge. The Eclipse dataset, which contains over 80,000 bug reports and includes both master and duplicate reports, is used to validate the suggested method. This investigation focuses primarily on the report descriptions in order to identify duplication. For Top-N proposals, the recommended two-tier technique has achieved a 75% recall rate, which is higher than the traditional one-on-one classification model.

Keywords: topic modelling, machine learning, natural language processing, bug tracking, multimodality

Table of Contents

DECLARATION ..... **i**

LANGUAGE CORRECTNESS CERTIFICATE ..... **ii**

COPYRIGHT STATEMENT ..... **iii**

ACKNOWLEDGEMENTS ..... **iv**

ABSTRACT ..... **vi**

Table of Contents ..... **vii**

List of Figures ..... **x**

List of Tables ..... **xi**

CHAPTER 1: INTRODUCTION **12**

1.1. Background **12**

1.1.1. Bug Report: *12*

1.1.2 Duplicate Bug Report *14*

1.1.2. Bug Tracking Systems *15*

1.2. Motivation **16**

1.3. Aim and Objective **16**

1.4. Approach **17**

CHAPTER 2: SYSTEMATIC LITERATURE REVIEW **18**

2.1. Introduction **18**

2.2. Review Methodologies **19**

2.2.1. Searching Process *19*

A. Query Search *19*

B. Database Search *19*

2.2.2. Selection and Rejection Criteria *20*

2.2.3. Categories Definition *21*

2.2.3.1. Attribute selection 21

2.2.3.2. Feature Weighting 21

2.2.3.3. Classification model 22

2.2.4. Data Extraction and Synthesis *22*

2.3 Results and Analysis **23**

2.3.1. Categorization w.r.t Feature Weighting *23*

2.3.2. Categorization w.r.t Classification model *25*

2.3.3. Identification of Retrieval *26*
2.3.4. Evaluation Metrics *27*
2.4. Answers of Research Questions **27**
CHAPTER 3: PROPOSED METHODOLOGY AND IMPLEMENTATION **29**
3.1. Data Set **29**
3.1.1 Source *29*
3.1.2. Attributes *29*
3.1.2.1 Issue Id: 29
3.1.2.2 Priority: 29
3.1.2.3 Component: 29
3.1.2.4 Duplicated issue: 29
3.1.2.5 Title: 29
3.1.2.6 Description: 29
3.1.2.7 Status: 29
3.1.2.8 Resolution: 30
3.1.2.9 Version: 30
3.1.2.10 Created time: 30
3.1.2.11 Resolved time: 30
3.2 Pre-Processing **30**
3.2.1 Text Cleaning: *30*
3.2.2 Stop Word Removal: *30*
3.2.3 Tokenization: *31*
3.2.4 Lemmatization: *31*
3.2.5 Stemming *31*
3.5.6. Bag of Words generation: *31*
3.3 Proposed Methodology **32**
3.3.1 LDA Topic Modelling *32*
3.3.1.1 Initialization: 32
3.3.1.2 Random topic assignment: 32
3.3.1.3 Reassignment: 32
3.3.1.4 Model updation: 33
3.3.1.5 Iteration: 33
3.3.2 Classification *35*
3.3.2.1 Formation of Feature Vectors: *36*
3.3.2.2 Single-Modality Feature Vector: 36
3.3.2.3 Multi-Modality Feature Vector: 36
3.3.3 Top-N Recommendations: *36*

3.3.4.1 Cosine Similarity: 37
3.3.4.2 Euclidean Similarity: 37
3.4 Implementation 38
3.4.1 Computing Environment 38
3.4.2 Experimentation 38
3.4.3 Evaluation Criteria 38
CHAPTER 4: RESULTS AND DISCUSSION 39
CHAPTER 5: CONCLUSION 52
CHAPTER 6: FUTURE WORK AND RECOMMENDATIONS 53
References 53

List of Figures

Figure 1 First Type of Big Report 15
Figure 2 Second Type of Bug Report 15
Figure 3 Selection and Rejection Criteria 21
Figure 4 Data Pre Processing Flow 30
Figure 5 Actual Data After Processing 31
Figure 6 LDA Topic Modelling 32
Figure 7 LDA Based Topic Modelling Flow 35
Figure 8 LDA Based Topic Modelling 39
Figure 9. RR Across all Models for 2.5k samples for w1=0.5, w2=0.5 40
Figure 10. RR% Comparison of M1,M2,M3 & M4 for w1=0.5, w2=0.5 41
Figure 11. RR Comparison of M1,M2,M3 & M4 for w1=0.1, w2=0.9 42
Figure 12. RR Comparison of M1,M2,M3 & M4 for w1=0.2, w2=0.8 43
Figure 13 RR Comparison of M1,M2,M3 & M4 for w1=0.3, w2=0.7 44
Figure 14 RR Comparison of M1,M2,M3 & M4 for w1=0.4, w2=0.6 45
Figure 15. RR Comparison of M1,M2,M3 & M4 for w1=0.6, w2=0.4 46
Figure 16. RR Comparison of M1,M2,M3 & M4 for w1=0.7, w2=0.3 47
Figure 17. RR Comparison of M1,M2,M3 & M4 for w1=0.8, w2=0.2 48
Figure 18. RR Comparison of M1,M2,M3 & M4 for w1=0.9, w2=0.1 49
Figure 19. M4 Performance 50
Figure 20. M4 Trend for 2.5K Samples 51

List of Tables

Table 1 Information Categories and Items in Bug Reports....12
Table 2 Query Search....19
Table 3. Database Search....20
Table 4 Data Extraction and Synthesis Form....22
Table 5. Categorization w.r.t Feature Weighting....23
Table 6. Categorization w.r.t Classification model....25
Table 7. Identification of Retrieval....26
Table 8. Evaluation Metrics....27
Table 9 Performance Analysis W.R.T Recal Rate, For Top 2.5K, W1= 0.5, W2 =0.5....40

CHAPTER 1: INTRODUCTION

1.1. Background

The maintenance of software is an important part of the software development life cycle. Bug reports (also known as Issue or Defect reports) are essential for software maintenance. Bug reports can be used to guide developers in repairing software defects, estimating problem-fixing time, deciding which bug should be solved first, deciding who should patch a particular bug, and assisting in locating the issue position that needs to be fixed. It can also provide useful information on the software project's progress.

Bugs are usually reported to a bug tracking system (also known as a bug tracking repository or issue tracking system) [1], which keeps track of them which is used to monitor software maintenance tasks. Bug tracking systems are designed to help developers, testers, and consumers not only define and report system issues, but also to store bugs and other pertinent information while tracking the status of each issue [1] [2]. As a result, a bug tracking system can improve the speed with which flaws are discovered, as well as the quality of software and its ability to satisfy client needs.

1.1.1. Bug Report:

A bug report contains various information which is relevant to a specific bug. Bug reports are a type of structured record that contains several forms of information concerning bugs, as shown in Table 1.

Table 1 Information Categories and Items in Bug Reports

Category	Attribute	Explanation
Numerical Information	Bug ID	The Bug unique identification number, which is automatically generated
	Created date (Open date, Open date)	The date of submission
	Close data	The date of closing the bug
	Dup_ID	If the bug report is a duplicate, the Dup ID is the

		ID of the bug report's master bug report.
	Response time	How soon the development team replies to a user's submission is determined by the time between submission and first comment.
Textual Information	Header (title, summary)	A concise and brief description of the bug
	Description	A detailed outline of what went wrong
	Steps to reproduce	A description of how to reproduce the failure
	Author	The name of person write the bug report
	Assign	The name of person is assigned to fix the bug
	Comments	The discussion about the bug
Categorical Information	keywords	A few words to summarize the bug
	Type	The bug report's nature, such as defect, enhancement, or feature
	Version	The version of the software was used
	Firmware (Operating system)	The firmware where the bug resided
	Severity	How serious of the failure
	Frequency	The times of the failure happened
	Urgency	How important of fixing the failure
	Category (component)	The component where the bug resided

	Platform	The several types of systems that potentially be harmed by the bug
	Correlating classes, methods, fields	pattern or subject of a bug
	Priority	The priority of the bug report based on its severity.
	Status	To check if the bug report is new, or assigned, or fixed, or closed
	Resolution	To indicate whether a bug or error is resolved.
	Product (Hardware information)	The product where the bug resided
Execution (Crash) information	Call stack trace (Exception stack frames)	Stack frames which are recorded during the execution of a program, representing function calls or procedures
	Appendices (Patches or Screenshots)	Attachments such as screenshots of failure

1.1.2 Duplicate Bug Report

There are two sorts of duplicates, as indicated in Figure 1 and Figure 2, according to Runeson et al. [3]. The first category of duplicate bug reports uses similar languages to describe the same failure. The second type of duplicate bug reports which describe two different failures generally use different vocabularies. But those failures are caused by the same underlying bug.

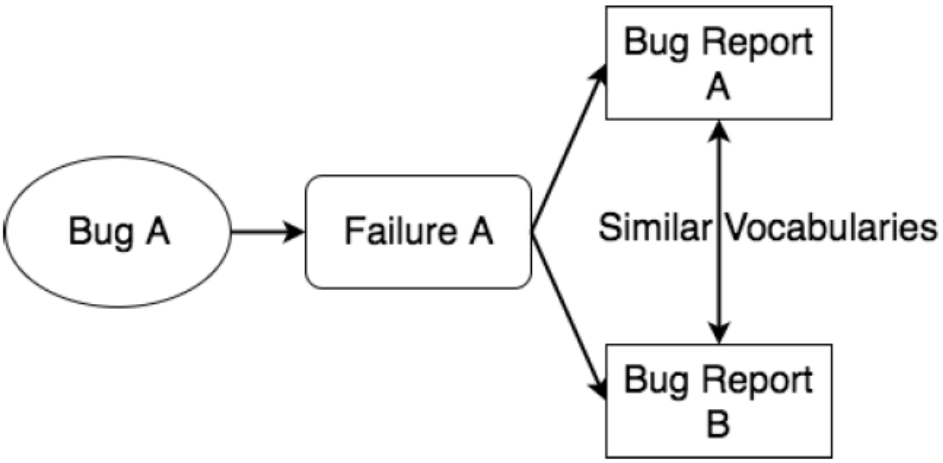


Figure 1 First Type of Big Report

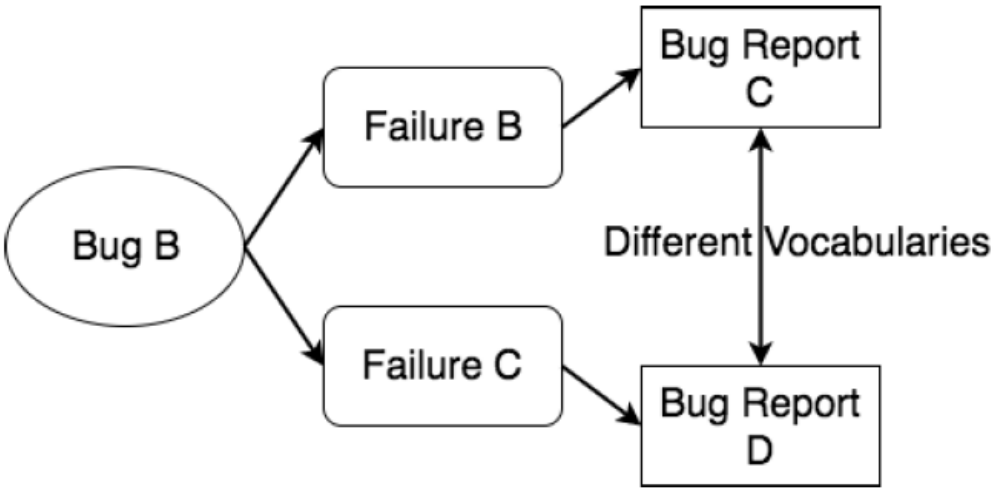


Figure 2 Second Type of Bug Report

A master (original) bug report is the first bug report submitted for a specific bug in a system, while duplicate bug reportsaries are the successive bug reports for the same bug.

1.1.2. Bug Tracking Systems

Bug reports are typically maintained in a database, which is referred to as a bug tracking system (bug repository). Bug tracking systems, like any other information system, are used to store and retrieve bug report data [4]. Its primary use is to track a specific bug. Some are commercial, such as Atlassian JIRA and FogBugz, while others, such as Bugzilla and ITracker, are open source initiatives. During the development and management of software projects, it is a centralised database that serves as a communication and coordination hub.

1.2. Motivation

Many bug reports submitted are duplicates (duplicate issue reports are created when many users submit bug reports for the same problem), resulting in a huge number of duplicate bug reports in a bug tracking system [5]. According to studies, the rate of duplicate bug reports might range from 25 to 30%. [5] [6] [7]. Duplicate bug reports lead to a situation that same issues are assigned to multiple developers who reproduce and fix the issue for the similar cause, which is a waste of effort and cost.

The important motivations for detecting duplicate bug reports are described below according to [3] [8] [9] [10]:

- When duplicate bug reports are assigned to multiple developers, it wastes the developers' time and effort.
- Addressing duplicate bug reports as a separate bug after a bug report has been addressed is a waste of effort.
- Finding duplicate bug reports can also help with problem fixing because some duplicate bug reports may provide additional information.

To identify duplicate bug reports, an analyst known as a triager now reads and reviews each existing bug report manually to identify duplicate bugs and determine whether a new incoming bug report reflects a new bug or an already reported bug [5]. He (She) uses either his (her) memories and experience or the bug repository's search capabilities [5]. The process is difficult and time-consuming, especially when considering the normal number of bug reports written everyday [11]. As a result, technologies for automating the detection of duplicate bug reports are required. New bug reports are directly compared with existing bug reports to find comparable bug reports, lowering bug report processing time by eliminating the need for a human to spend time reading, understanding, and searching. It has the potential to improve software maintenance efficiency.

Although there has lately been a lot of research on automatically detecting duplicate bug reports, for example, several variants of information-retrieval algorithms have been examined, there is still a lot of potential for improvement in the duplicate detection process. Furthermore, only a few tools were tested in an industrial setting.

1.3. Aim and Objective

Our study's goal is to find duplicate bug complaints. Because of the following reasons, automated duplicate bug report detection is a difficult challenge [12], [13]

- Bug reports are written in a natural language manner. In the event that several aberrant behaviours are caused by the same software bug, these reports may use different terminology to describe the same bug. In natural language comprehension, there is a linguistic ambiguity and variability problem, which makes it difficult to distinguish reports that are comparable.
- The bug tracking system has a high amount of bug reports. The number of bug reports in the TROUBLE of the case corporation – Axis is over 100,000, which is a significant number when compared to the amount of work involved.
- The quality of a part of bug reports is not sufficient, for example, the bug report contains weak, inadequate, missing, or even incorrect information.

We want to build a system that automatically recognizes duplicate bug reports.

1.4. Approach.

We proposed a double tier model. The model will look for the Top-N similar master reports whenever a brand-new report is submitted. It consists of two steps that are clustering and classification In clustering, preprocessing of the date set is done and is fed into LDA-based topic modelling. Classification comprehends numerous steps that includes single and multi-modal text representation using Word2vec (W2V), top-N clusters selection, GloVe, Fasttext (FT), and measure of similarity of text which is attained by fusing similarity measures of the Cosine and Euclidean nature via assigning different weights.

CHAPTER 2: SYSTEMATIC LITERATURE REVIEW

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

2.1. Introduction

There are always challenges that arise throughout the development, maintenance, and use of a software application, placing the system's overall functionality in peril due to intrinsic defects. The software industry spends billions of dollars on product maintenance [14], [15]. A Bug Tracking System (BTS), like Bugzilla1, recognises a technical disagreement in a software environment as a bug and creates a bug report. Every problem report is assigned to software developers for further assistance as needed, based on its severity. The biggest difficulty with the procedure is the chance of multiple bug reports being filed with similar types of issues, resulting in a high likelihood of having the same or similar solution repeatedly. As a result, duplicate bug reports [16], [17] become a problem. These bug reports are published in plain text, which might be difficult to decipher, making it harder to spot duplicates because different terms can be used to describe the same issue. As a result of this stumbling block, the developer is forced to look for the same solutions for diverse defect reports, resulting in software companies spending more than 45 percent of their budget on bug patches [14]. If the repository already has a master report of incoming bug reports and the report is still given to the developer, a significant amount of resources will be wasted on an issue that has already been fixed or is now being resolved. As a result, it's critical to spot duplicate bug reports.

As a result, there is a compelling need to conduct a thorough evaluation of current literatures on bug duplication. The goal is to lay a solid foundation for a comprehensive comparison of present and future automated duplicate bug report identification systems. This contribution is especially valuable for academics who want to contribute to this study area by developing new ways to automated duplicate bug report detection or improving or refining existing ones.

The research questions that we will answer to identify and categorize existing automated duplicate bug reports detection methods are as follows:

RQ1 What are the existing methods to detect duplicated bug reports automatically?

RQ2 What are different categories of all automated duplicate bug reports detection methods?

RQ03 How are the methods evaluated?

2.2. Review Methodologies

Kitchenham [14] outlined a well-defined methodology for conducting a systematic literature review (SLR). SLR is most commonly performed using this pattern by researchers. Throughout order to gain more exact and accurate responses to our queries, we followed this strategy in our research.

2.2.1. Searching Process

A. Query Search

Kitchenham [14] [18] advised that terms and structured search strings from RQ1 be classified according to Population, Intervention, Comparison, and Outcomes.

To create the search string, the detected keywords were sorted into groups and their synonyms were considered..

Table 2 Query Search

Concept	**Searched String**
Bug	(bug*; defect*; failure; error*; problem*; issue*; crash*)
Detection	detect*, rank*, retriev*, link*, connect*
Duplicate	(duplicat*; same; similar*)
Report	(report*; document*; text*)

B. Database Search

Those search strings are used to conduct a manual search to identify as much relevant primary studies as possible on four digital databases: IEEE Xplore, ACM Digital Library and Scopus.

Table 3. Database Search

Database	Website	Search Result	Setting
IEEE Xplore	http://ieeexplore.ieee.org	2092	Metadata Only
ACM Digital Library	http://dl.acm.org	386	Default
Scopus	https://www.elsevier.com/	3940	Default

We used the below search Query:

(bug* OR defect* OR failure* OR error* OR problem* OR issue* OR crash*) AND (detect* OR rank* OR retriev* OR link* OR connect*) AND ((duplicat* OR similar* OR same) AND (report * OR document * OR text)

2.2.2. Selection and Rejection Criteria

During the primary studies screening procedure, the following selection and rejection criteria were utilised to determine whether an article's content was relevant based on titles, abstracts, keywords, full-text reading, and quality assessment to omit studies that did not answer the RQ1.

The following are the selection criteria:

- Automated duplicate bug report detection research must have been addressed in studies.
- Peer-reviewed studies must be published in journals, conferences, and workshops
- Studies in the subject of software engineering are required.

The following exclusion criteria were used to reject a study:

- Any article that is not peer-reviewed (grey literature, books, presentations, blog posts, etc.
- Any article that is not available in full-text,
- Any conference proceedings (e.g., messages from chair of editorial boards, etc.

- Any article that is a duplicate of other studies are all excluded.

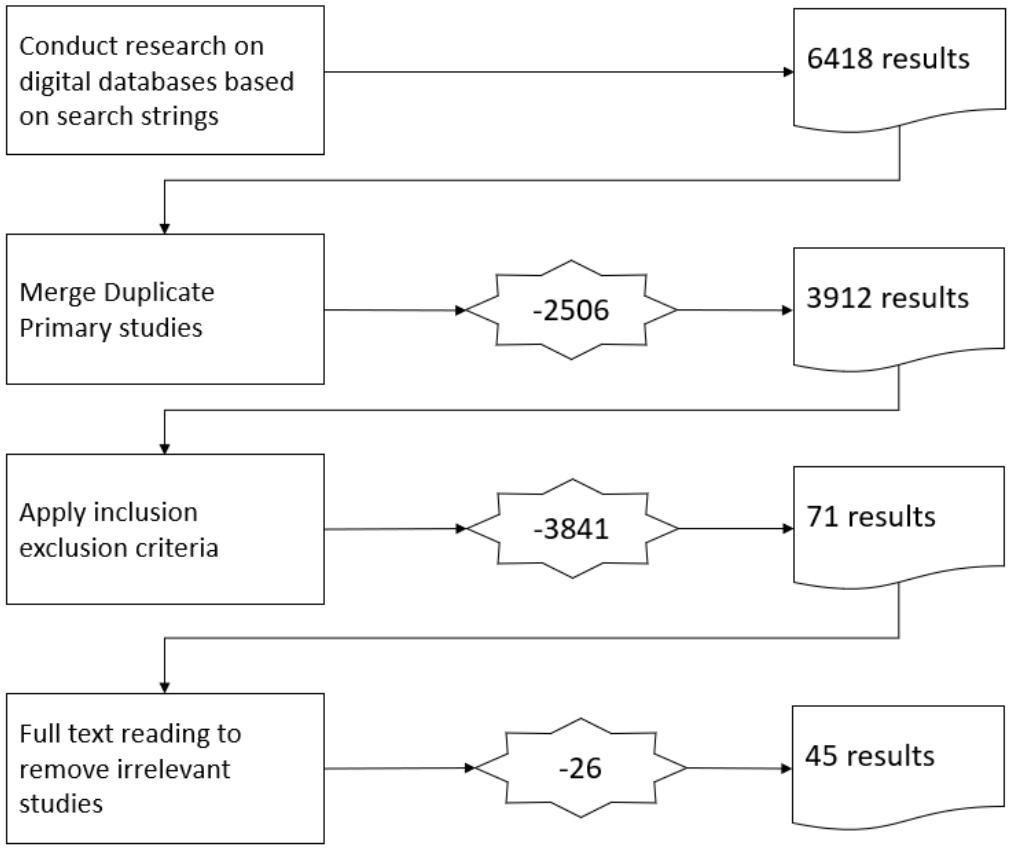


Figure 3 Selection and Rejection Criteria

2.2.3. Categories Definition

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

2.2.3.1. Attribute selection

The attribute selection task entails picking a subset of the originally available attributes to be used in the model generation process. General-purpose attribute selection algorithms can be used to choose attributes for a variety of target algorithms, as well as – in certain cases – distinct target tasks..

2.2.3.2. Feature Weighting

Characteristic weighting is a crucial step in text categorization that determines the importance of each document's feature.

2.2.3.3. Classification model

A classification model attempts to draw a conclusion from the training data. It will forecast the class labels/categories of the new data.

2.2.4. Data Extraction and Synthesis

To record all the relevant information that will allow us to answer the proposed RQ1, we extracted data based on the following data extraction form as shown in Table 4-8, which was adapted from other similar studies [18] [19]. There is a data item and a value in each data extraction field. The resulting database collects general studies information and specific data information addressing the RQ1, RQ2, RQ3 and RQ4.

Table 4 Data Extraction and Synthesis Form

Category	Data Field	Value
General Information	Article Title	Name of the articles
	Author Name	Set of Names of the authors
	Year of Publication	Calendar year
	Type of Contribution	Model, theory, framework, or approach, instructions, lessons gained, counsel or implications, and tools are examples of study outcomes. These numbers were taken from [19].)
	Type of Research	Research types (evaluation research, solution proposal, validation research, philosophical papers, opinion papers, experience papers, etc. These values were adapted from [19].)
	Research Method	Research methods employed as part of the research process (case study, industry report, experiment, survey, action research, mixed methods, grounded theory, design science, opinion paper, and not stated. These values were adapted from [20].)
Specific Information	Keywords	Concept labels or keywords found in the abstracts and conclusion

	Method	Activities that are proposed to automatically detect duplicated bug reports
	Evaluation	Activities that are proposed to evaluate a method
	Dataset	Databases that are mentioned in the primary studies comprise defect reports
	Metric	Metrics are proposed to measure a method
	Performance	Outcomes of evaluation that measure the performance of a method

We applied a qualitative synthesis method and performed a statistical analysis of the data extracted from the primary studies separately based on the classification scheme with the main goal of understanding, structuring, and classifying current research on the automated duplicate bug reports detection.

2.3 Results and Analysis

The application of the SMS protocol yielded the results which are presented in this section. Before presenting results, As illustrated in Figure, we consider the various publication venues. In systematic mapping research, examining specific publication outlets is typical. [21].

2.3.1. Categorization w.r.t Feature Weighting.

Table 5. Categorization w.r.t Feature Weighting

Model	Scheme	Reference	Number
Vector Space Model	TF-IDF	[4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]	19
	BM25-based (BM25, BM25F, BM25Fext)	[23], [24], [25], [15], [26], [27], [12], [16], [28]	9
	IDF (weight = log2(frequency))	[29], [30], [9], [12]	4

	TF-IDF-CFC weighting	[31], [16]	2
	Term frequency (TF), Total term frequency (TTF)	[12]	1
	Log-based (weight = 1 + log(frequency))	[32]	1
	Log-IDF, Log-Entropy, TF-Entropy	[8]	1
	Jelinek Mercer language model	[15]	1
	Dirichlet language model	[15]	1
	Temporal k-occurrence model	[11]	1
	Wiki-similarity	[12]	1
Character-Level N-Gram Model	Character n-gram feature extractor similarity computation module	[33], [34], [35], [19], [36], [21]	6
	Top-N similar bug reports retrieval module	[33]	1
	An adaptation of the INCLUS algorithm	[37]	1
	Correlation coefficient BM25 ranking function	[16]	1
Topic Model	LDA	[24], [8], [26], [38], [36], [39], [28], [40]	8
	LSI	[8], [14], [12]	3
	Labelled LDA	[26], [28]	2
	Hierarchical Dirichlet Process	[41]	1
Word Embedding Model	Word2vec	[16], [22], [40]	3
	FastText	[40]	1

	GloVe	[40]	1

2.3.2. Categorization w.r.t Classification model

Table 6. Categorization w.r.t Classification model

Classification model	Method	Reference	Occurrences
Discriminative model (Binary Classification, Cluster Modeling)	Support Vector Machine	[29], [25], [31], [35], [20], [16], [28]	7
	Naıve Bayes	[35], [38], [20], [28]	4
	Logistic Regression	[26], [38], [41], [28]	4
	K-NN	[26], [35], [38], [40]	4
	ZeroR	[26], [38], [28]	3
	LinearSVM	[35], [19]	2
	Gradient descent(Stochastic gradient descent algorithm)	[29], [9]	2
	Agglomerative Hierarchical Clustering	[30], [42]	2
	Random Forest	[35], [43]	2
	Ranking SVM	[12]	1
	REPTree with Bootstrap Aggregating technique	[38]	1
	Ensemble Averaging Linear combination	[36]	1
	A clustering method	[6]	1
	A graph cluster algorithm	[3]	1
	Linear Regression	[3]	1
	The k-means clustering algorithm	[42]	1

	Expectation Maximization,X-means algorithm	[41]	1
	Clustering Labeling	[41]	1
Multi-label classification	MULAN (a multi-label classification scheme)	[42]	1

2.3.3. Identification of Retrieval

Table 7. Identification of Retrieval

Retrieval Concern	**Method**	**Reference**	**Occurrence**
Similarity Measurement	Cosine Similarity	[32], [4], [3], [29], [33], [5], [30], [6], [8], [7], [27], [26], [11], [12], [18], [34], [37], [19], [46], [17], [36], [16], [22], [10], [28], [21], [40]	27
	Dice Similarity	[32]	1
	Jaccard Similarity	[32]	1
	Secondary similarity measure	[11]	1
	Position Dependent Model (PDM)	[42]	1
	KL-divergence	[44]	1
	Character-Level N-Gram Model Similarity computation module	[33]	1
Time window (Time frames)		[32], [7], [43], [13], [10], [15], [43]	7
Group Centroids		[7], [43]	2
Score Cutoff Thresholds		[15], [37]	2

Fidelity loss function		[5]	1
Feature Reweighting	Cluster Shrinkage	[34]	1

2.3.4. Evaluation Metrics

Table 8. Evaluation Metrics

Metrics	Reference	Occurrence
Recall-rate@k	[3], [4], [32], [29], [33], [30], [6], [7], [45], [8], [27], [43], [34], [31], [16], [23], [24], [9], [13], [21], [22], [36], [40]	23
Precision	[3], [25], [15], [44], [37], [35], [19], [46], [17], [36], [39], [47]	12
MAP	[23], [24], [12], [22], [47], [39]	6
AUC	[39], [26], [38], [35]	4
Kappa	[39], [28], [26], [38]	4
MRR	[22], [9], [13], [21]	4
Feedback	[46], [17]	2
Likelihood	[17], [46]	2

2.4. Answers of Research Questions

RQ1 What are the existing methods to detect duplicated bug reports automatically?

Answer:

Attribute selection, Feature calculation, similarity calculation are the existing methods

RQ2 What are different categories of all automated duplicate bug reports detection methods?

Answer:

Because title and summary are the most fundamental attributes of bug reports in software projects, the results demonstrate that they are the most often used qualities in attribute selection. Although extra data, like as execution information, aids in the detection of duplicate bug

reports, obtaining it is costly due to the complexity of generating or receiving execution information.

For the feature weighting model and similarity calculation, the results indicate that TF-IDF and Cosine Similarity are dominant techniques used in automated duplicate bug reports detection methods. The feature weighing model can be divided into two ways, one is machine learning method, another is non-machine learning method.

When considering about applying machine learning methods, when the feature size becomes larger, the efficiency of classification become lower.

These techniques can be further classified into two groups for machine learning methods: unsupervised learning approach and supervised learning approach. There is no need for any training data in the unsupervised learning strategy. As a result, it can be applied to any batch of problem reports.

In primary investigations, the problem of duplicate detection has been addressed in a variety of methods. We divided them into three major groups: All three issues must be solved: the TOP N recommendation and ranking problem, the binary classification problem, and the decision-making approach.

RQ03 How are the methods evaluated?

Answer:

We identified all performance metrics which are used to evaluate various automated duplicate bug reports detection methods. There are precision, recall, F1 (F-measure, F-score), accuracy, AUC, kappa, recall-rate@k, MRR, MAP, Normalized Discounted Cumulative Gain, Cluster purity classification accuracy, Feedback, Likelihood, and EARate. Among them, the recall-rate@k is the most frequently used performance metric.

CHAPTER 3: PROPOSED METHODOLOGY AND IMPLEMENTATION

3.1. Data Set

3.1.1 Source

The Eclipse dataset is used in the ablation study. It contains 85,156 bug reports from October 10, 2001, to December 30, 2013. After completing all of the pre-processing methods outlined in Section 3.2, the dataset contains 85,027 bug reports, 70,629 of which are master reports and 14,398 of which are duplicate bug reports. In this example, all 70,629 master reports and 1000 duplicate reports are used at random. A bug report is a structured document with eleven fields, as shown below, although this research just uses the description field.

3.1.2. Attributes

3.1.2.1 Issue Id:

The bug tracking system assigns a unique number to each problem report.

3.1.2.2 Priority:

It's the significance and urgency with which a problem must be resolved.

3.1.2.3 Component:

It demonstrates where an issue arises in the system..

3.1.2.4 Duplicated issue:

It presents a duplicate report of the original report.

3.1.2.5 Title:

In a single line, state the problem.

3.1.2.6 Description:

This clarifies the issue much further.

3.1.2.7 Status:

It displays the bug's status as Open, Fixed, Closed, or Deferred.

3.1.2.8 Resolution:

It displays the bug report's current status, such as whether it's Fixed, Won't Fix, Invalid, or Duplicate.

3.1.2.9 Version:

It displays the programme version in which the bug was discovered.

3.1.2.10 Created time:

It shows when the bug report was created as well as the date and time.

3.1.2.11 Resolved time:

The date when the bug was repaired is displayed.

3.2 Pre-Processing

As previously stated, the bug reports are written in plain natural language. As a result, cleaning the data and converting it to a common format for future analysis necessitates a significant amount of pre-processing. Stop word removal, lemmatization, text cleaning, construction of a continuous Bag of Words, and tokenization are only a few of the procedures involved (CBoW).

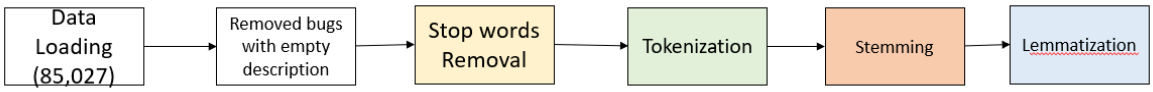


Figure 4 Data Pre Processing Flow

3.2.1 Text Cleaning:

Unnecessary sections of the bugs are deleted during text cleaning. In this phase, all invalid bug reports are completely erased. Invalid bug reports are ones with an empty description field or a set of fixed description patterns. Statement "Has been tagged as read-only" is an example. This indicates that the bug report is readonly. As a result, such bug reports go unaddressed, but their inclusion in the model may have an impact on the similarity measures. As a result, before modelling an efficient duplicate bug report detection system, all such reports must be eliminated.

3.2.2 Stop Word Removal:

Stop words include keywords like "the," "in," "a," and "that," among others, because they have little meaning or value and have a tendency to negatively impact the duplicate bug identification

process. As a result, the elimination of these stopwords is unavoidable. A change to lower case occurs after all of the stop words have been removed from the text.

3.2.3 Tokenization:

Tokenization is the breakdown of text into a small number of significant parts called "tokens" [64].The process of transforming a stream of words into a stream of tokens is known as tokenization. This is accomplished by deleting all extraneous information such as punctuation, capitalization, and numerals. We eliminated all the periods and other punctuation characters from the bug report's text. All uppercase letters are converted to lower case letters. Tokenization is applied to all bug reports, and the list of tokens for each report is used as an input in the phases that follow.

3.2.4 Lemmatization:

The technique of reducing a word to its root form while keeping its context is known as lemmatization; for example, "fix" becomes "fix," and "requires" becomes " require."

3.2.5 Stemming

Words can be written in a variety of grammatical forms, but the information pertaining to that word remains the same. As a result, stemming a word reveals its root form. All affixes are removed from tokens during stemming, leaving us with a list of stemmed tokens for all bug reports.

3.5.6. Bag of Words generation:

BoW is a textual data feature extraction technique. It's a text representation that shows the location of a word in a document. The power to multiply a text is portrayed as a jumble of words with no regard for grammar or even word order. This approach consists of two steps: determining vocabulary and vectoring words based on their existence in a document.

	Issue_id	Duplicated_issue	Title	Description
0	1	NaN	[usability, external, editors]	[project, contain, resource, release, project,...
1	2	NaN	[open, repository, resources, doesnt]	[open, repository, resource, default, editor, ...
2	3	NaN	[indicate, deletion]	[deletion, indicator, viewer, subtle, vision, ...
3	4	NaN	[better, message, catch, resource]	[synchronize, project, repository, different, ...
4	5	NaN	[isharingmanager, share, inconsistent]	[gettingsetting, manage, resource, methods, is...

Figure 5 Actual Data After Processing.

3.3 Proposed Methodology

3.3.1 LDA Topic Modelling

A statistical model that is unsupervised and allows unknown groups to examine a set of observations and explain why some areas of the data are similar is known as LDA. Representation of each document is done as a latent topics distribution, each of which is defined by a set of words. A document-topic matrix is built by using LDA on BOW. A distribution of probability of the topic is then provided by LDA within each document for every document. LDA starts with a key presumption, in which a document is analyzed to select a group of topics. Then, for each topic, it assigns a collection of words. It encompasses the six items listed below.

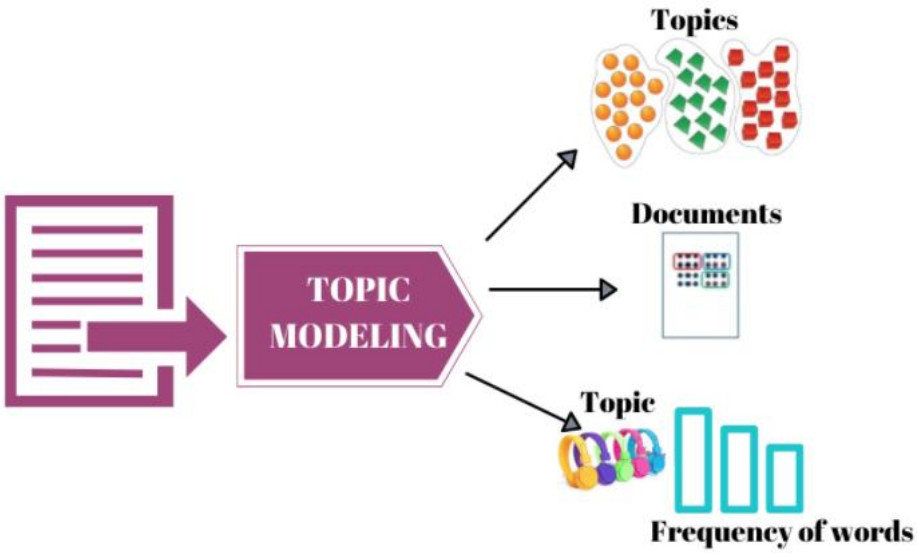


Figure 6 LDA Topic Modelling

3.3.1.1 Initialization:

Using t as a predetermined number of subjects.

3.3.1.2 Random topic assignment:

Each topic t in topics has been allocated a v in the document di at random. This ensures distributions of subject and word for all papers and themes.

3.3.1.3 Reassignment:

Increasing the randomness of the assignments by determining the meaning of each word v in the document di:

- Currently assigned total number of words in document di to topic t.
- Over all the documents that arise from the term v, the total number of assignments to topic t.

Finally, a new topic is given to the letter v.

3.3.1.4 Model updation:

The assumption made by LDA here is that the current word's allocation is incorrect but all other allocations are correct. Thus upgrades the allocation of that word.

3.3.1.5 Iteration:

4 The same process is repeated for all the assignments so that they are all correct. The assignments here are then later used to perform probability distribution of the k topics in documents di along with the words that were assigned to each subject.

5 The assumption here is that t topics are generated by LDA from a document di. Suppose the previous distribution is a *k x z* matrix, with δ as a variable of distribution of the word (δ is aggregation of words) inside the topic, $\delta_{ij} = \mathbf{P}(x_j \mid y_i)$ is that the probability of the word x_j within the i_{th} topic. This results in generation of a document topic matrix containing N topics, which is shown in Eq (1).

$$p(\alpha, y, x) = p(\alpha \mid \theta) \prod_{n=1}^{N} p(y_n \mid \alpha)(x_n|y_n, \delta), \qquad (1)$$

This represents x as the N-word vector, y as the N-dimensional subject vector, showing the document's topic distribution vector

$$p(y \mid \theta, \delta) = \int p(\alpha \mid \theta) \prod_{n=1}^{N} p(y_n \mid \alpha)(x_n|y_n, \delta)\, d\alpha, \qquad (2)$$

With *c* being the corpus, having m documents $p(c \mid \theta, \delta)$ is estimated, as

$$p(c|\theta, \delta) = \sum_{c = 1 \ldots m} p(x_c|\, \theta, \delta). \qquad (3)$$

Thus,

$$p(D|\theta, Z) = \prod_{d=1}^{M} \int p(\alpha_d|\theta)\left(\prod_{n=1}^{N_d} \sum_{d_n} p(\beta_{dn}|\alpha_d)\, p(\gamma_{dn}\,|\beta_{dn}, Z)\right) d\alpha_{d.} \qquad (4)$$

Now, maximization of parameters is performed by the LDA model on θ and δ in p(c | θ , δ) by using the expression

$$p(\alpha, y|x, \theta, \delta) = \frac{p(\alpha, y, x \mid \theta, \delta)}{p(y|\theta, \delta)} \qquad (5)$$

We are able to obtain *p*(*c* | θ , δ) θ and δ after a topic model is built. The trained topic model is used to infer the topic of the unlabeled text and estimated and possibly forecast the related topic distribution. One thing to note is that this is also the document's conditional probability distribution in the topic space.

5. Selection of Optimum Number of Topics in LDA:

We get the coherence score, which is a resultant of the mix of the internal measure UMass and the extrinsic measure UCI. It is used towards choosing the optimal number of topics, shown by below Eq. (6).

$$\text{coherence} = \sum_{i<j} \text{Score}\ (w_i, w_j) \qquad (6)$$

A Pointwise Mutual Information (PMI) is used in the UCI measure, as defined in Eq (7).

$$score_{UCI}(w_i, w_j) = log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} \qquad (7)$$

The probability of seeing ω_i and ω_j co-occurring in a totally random document can be represented as $P(\omega_i, \omega_j)$. $P(\omega)$ being the representation of the probability of seeing ω_i in a random document. Whereas the empirical conditional log-probability model is represented by the UMass, as shown in Eq (8).

$$log(w_i, w_j) = log \frac{P(w_i, w_j)}{P(w_j)} \qquad (8)$$

Counting the documents results in UMass, such as

$$score_{U\,MASS}(w_i, w_j) = log \frac{D(w_i w_j) + 1}{D(w_i)} \qquad (9)$$

where $D(\omega_i, \omega_j)$ is the count of documents containing both words the words . The total number of documents in the corpus, whereas $D(\omega_i)$ is the count of documents containing the word.

Once the topic model is created, the model, under discussion, builds clusters around the subjects.

6. Selection of Optimum Number of Topics in LDA:

The data set is compilation of master bug reports and duplicates. Our work results in isolation of the master bug reports from the duplicate bug reports initially so that the model can be tested on duplicate reports. The LDA model is trained using the The Bag of Words corpus obtained from the master reports. Each master report topic is computed along with the probability distributions of each document's theme. Once the topic distribution is examined in the master report, it is assigned to the cluster of highest topic distribution. This process is of cluster formation is show in an example.

Assume the LDA produces five topics.

- An analysis is performed on the Master Report$_i$ *(MR_i)* based on the above covered topics.
- MR_i is assigned to the 3rd cluster, since topic 3 has the largest probability among all the topics.

Each master report goes through this process until all of them are assigned to their proper clusters.

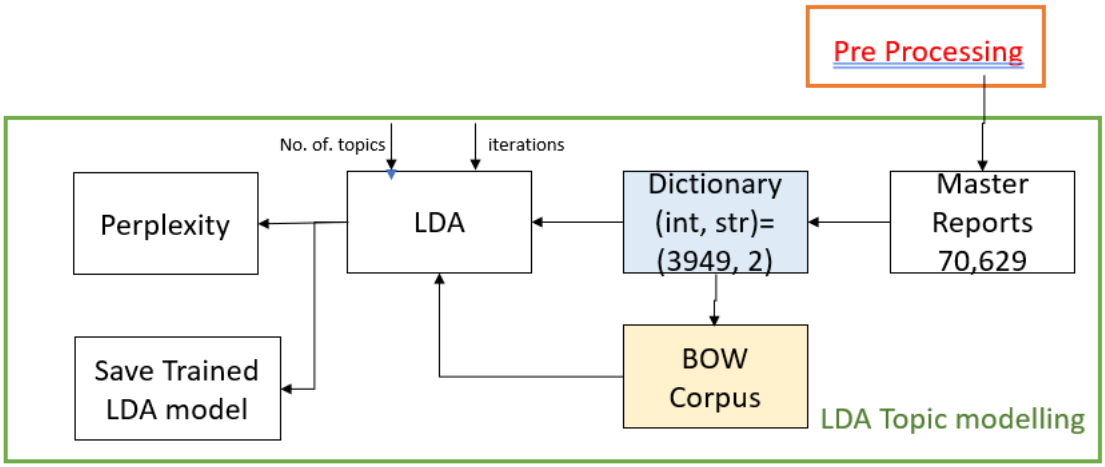


Figure 7 LDA Based Topic Modelling Flow

3.3.2 Classification

This stage contains Top-n cluster selection, W2V, FT, and GloVe text representation, and a measure of text similarity that combines similarity metrics of the Cosine and Euclidean nature.

3.3.2.1 Formation of Feature Vectors:

Various text to numeric representation algorithms are used to create feature vectors in a multimodality and single modality fashion.

3.3.2.2 Single-Modality Feature Vector:

Word2Vec (M1), FastText (M2), and GloVe are the three feature extractors used (M3)

3.3.2.3 Multi-Modality Feature Vector:

It is commonly known that multimodal approaches provide a wealth of information extraction and learning opportunities [65] [66] [67]. As a result, the suggested method takes use of multi-modality feature extraction by combining various feature vectors to improve performance. As a result, four multi-modal feature representations are proposed: M4 (FastText and GloVe), M5 (GloVe and Word2Vec), M6 (FastText and Word2Vec), and M7 (FastText, GloVe, and Word2Vec).

The fusing of numerous feature vectors is expressed in Equation (10).

$$\bar{v} = f_n(v_1{}^{R\epsilon D}, v_2{}^{R\epsilon D}), \quad (10)$$

where v_1 and v_2 are the vectors produced by single-modality feature extractors, $n \in 1,...4$, and D is the dimension of the vectors. The operations used to combine the feature vectors can be summarised as follows:

$$f_1(v_1, v_2) = v_1 \oplus v_2 \quad (11)$$

where $\oplus$ is a concatenation operation

$$f_2(v_1, v_2) = avg(v_1, v_2) \quad (12)$$

The point-wise average of the two feature vectors is obtained using this method. PCA can also be used to reduce dimensionality, improve information interpretability, and reduce data loss. It achieves this by progressively increasing the variance by introducing new unrelated variables. As a result, the concatenated outcome from Eq(11) is subjected to PCA,like in Eq. (13).

$$f_3(v_1, v_2) = PCA\ ((f_1(v_1, v_2))) \quad (13)$$

Similarly, PCA is used to reduce the feature vector provided by Eq. (12), as shown below.

$$f_4(v_1, v_2) = PCA\ (f_2(v_1, v_2)) \quad (14)$$

3.3.3 Top-N Recommendations:

This step's main purpose is to calculate an adequate measure of similarity between the

submitted bug report and the master report repository. The duplicate report's corelated master report is fetched from the clusters when the Top-N clusters are selected. Every master report in the Top-n clusters is compared to the duplicate report. As a result, the similarities of the Cosine and Euclidean nature of both feature vectors are estimated using the equation below to create feature vectors for each master report and duplicate report from the Top-N clusters.

. $$Similarity = w1 \times Cosine\ Similarity + w2 \times Euclidean\ Similarity$$

3.3.4.1 Cosine Similarity:

As demonstrated in below Eq, cosine similarity indicates how similar two vectors in an inner product space are in Eq. (15).

$$sin_{cos}(d_1, d_2) = \frac{d_1 . d_2}{\|d_1\|\|d_2\|} = \frac{\sum_{i=1}^{n}(d_{1i} d_{2i})}{\sqrt{\sum_{i=1}^{n}(d_{1i}^2)}\sqrt{\sum_{i=1}^{n}(d_{2i}^2)}} \qquad (15)$$

where d_{1i} and d_{2i} are the vectors of documents d_1 and d_2 in i_{th} dimension topic space. It analyses the Cosine angle between two multidimensional space projected vectors in a significant way. The Cosine value is a number that varies from 0 to 1. The Cosine value decreases as the angle increases.

3.3.4.2 Euclidean Similarity:

In Euclidean space, it is a straight-line distance between two points and is given by below Equation, as

$$Distance_{euc}(d_1, d_2) = \sqrt{\sum_{i=1}^{n}(d_{1i} - d_{2i})^2} \qquad (16)$$

where d_{1i} and d_{2i} are the document vectors d_1 and d_2 in Euclidean space ϵ R_n. The similarity score of the Euclidean nature between the two documents d_1 and d_2 can be computed from this via below equation as

$$\boldsymbol{sim_{euc}(d_1, d_2) = \frac{1}{1 + Distance_{euc}}} \qquad \boldsymbol{(17)}$$

3.4 Implementation

3.4.1 Computing Environment

We used Google Colaboratory for the complete implementation. Colaboratory or Colab is a Google product. Colab allows anybody to write and execute python code through the Google browser

3.4.2 Experimentation

The below steps were performed in order to run our pipeline and achieve our results.

1. Relevant libraries were imported.
2. Data was read.
3. Data pre-processing steps were performed as mentioned in section B-1.
4. Duplicate and master reports were separated.
5. Master reports are trained on LDA Model. When the model is trained, coherence score is calculated to detect the best possible number of topics.
6. 10 was picked out as optimum number of topics. Master reports were then pushed into these clusters based on topic modelling.
7. For feature selection, all these clusters were then trained on Models M1(Word2Vec), M2(FastText) and M3(GloVe).
8. M4, M5, M6 and M7 are made by fusing M3, M2 and M1 by techniques mentioned in section C.
9. Experimentations are performed to detect the best fusion model, for Top - 2.5K recommendations.
10. Afterwards, experimentations are performed on the models M4, M3, M2 and M1 for different samples starting from 10, 100, 500, 1000, 2000, and 25000 samples
11. Performance evaluation is made by recall rate

3.4.3 Evaluation Criteria

The measure provided by Equation(18) is used to assess this research.

$$RecallRate = \frac{N_{true}}{N_{total}} \qquad (18)$$

where N_{true} is the amount of duplicate reports that can locate their master report appropriately, and N_{total} is the total number of reports that are duplicates.

CHAPTER 4: RESULTS AND DISCUSSION

To begin, training was performed for the master reports on the LDA model for 1000 iterations and 20 passes using all the techniques outlined in Section II-D. For determining the optimal number of topics, an analysis and a coherence score were used. The highest coherence was achieved for 10 topics. As a result, ten topics were chosen as the optimal number for this study.

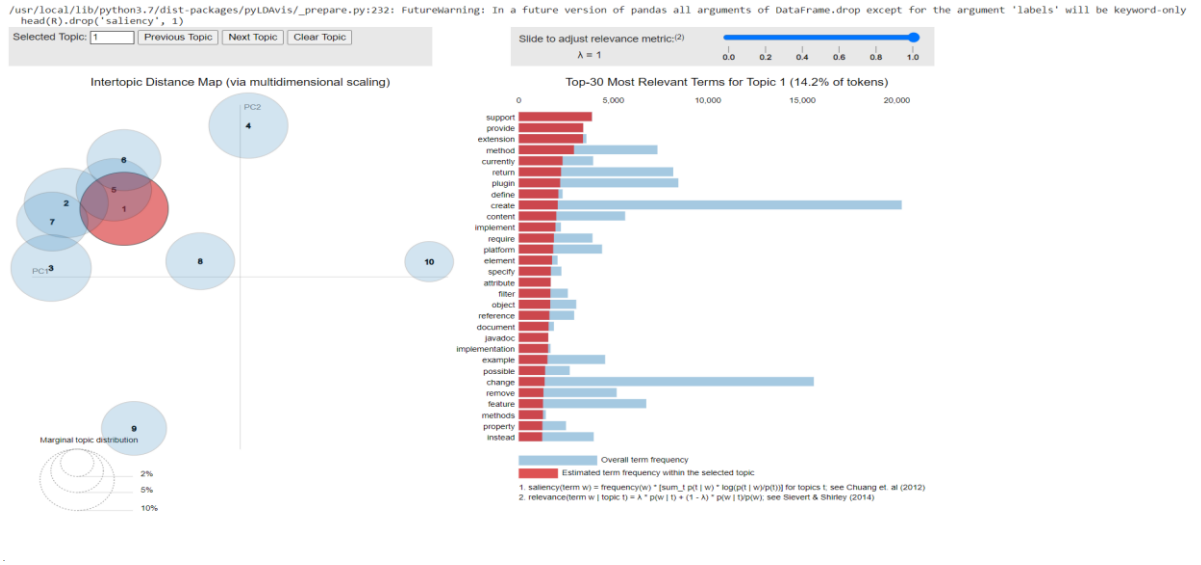


Figure 8 LDA Based Topic Modelling

In order to determine the best fusion model among our four fusion techniques namely F1 as concatenation of the vectors F2 as average of the vectors, F3 as Dimensionality reduction using PCA on concatenation of the vectors and F4 as Dimensionality reduction using PCA on average of the vectors, experimentation was performed for top 2.5K recommendation. As per the results mentioned in table I. Model M4 performed the best for all fusion techniques F1, F2, F3 and F4 and among these with fusion F4 and RR% 71.5% it performed the best.

It is to be noted that M5 performed better than M6 and M7. And for all models M4, M3, M2 and M1 Fusion technique of Dimensionality reduction using PCA on average of the vectors referred as F4 performed the best. Hence to carry out any further experimentation we took M4 and F4 as reference and did further experimentations by varying weights

Table 9 Performance Analysis W.R.T Recal Rate, For Top 2.5K, W1= 0.5, W2 = 0.5

	F1	F2	F3	F4
M4	**65.8**	**67.5**	**68.5**	**71.5**
M5	63.0	64.0	66.0	64.5
M6	64.5	61.5	61.5	65.0
M7	64.7	60.0	66.5	67.1

Taking this as reference, further experimentation is carried out on models M4, M3, M2 and M1 for Top 10-N: 10, 100, 500, 1000, 2000 and 2500K by varying the weights w1 and w2. The comparisons are done on single modality models M3, M2, and M1, as well as the best multimodal technique

M4 that was chosen as a consequence of an early analysis, as shown in Table1. When weights were modified, Figure 2-9 shows the trend of models M4, M3, M2, and M1. It can be seen that M4 performed the best. It's also clear that as the number of samples grows larger, the model's performance improves.

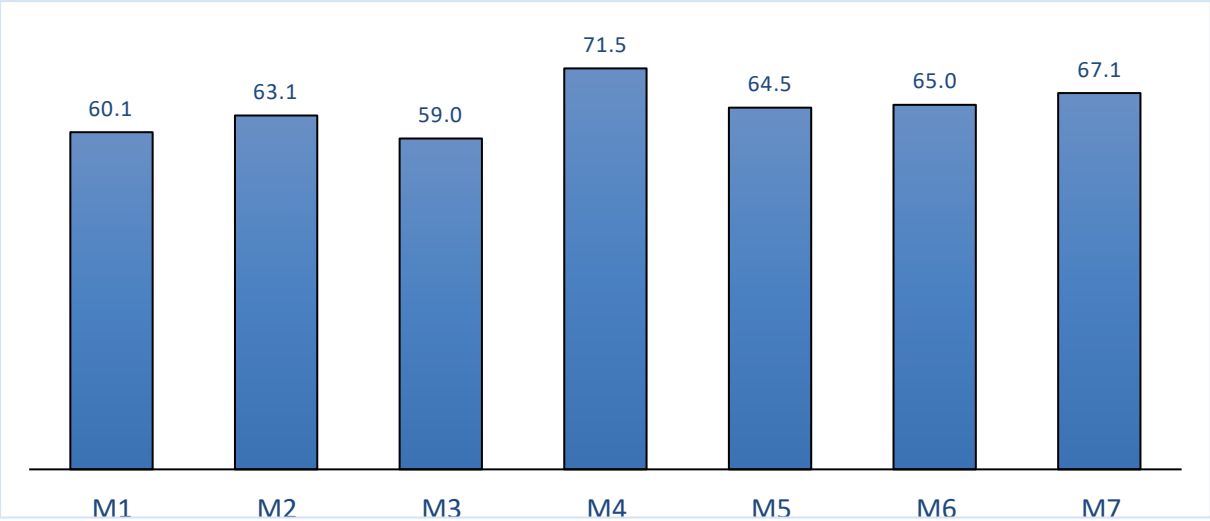


Figure 9. RR Across all Models for 2.5k samples for w1=0.5, w2=0.5

Fig 2-9 give a very detailed account of the trend that were acquired as a result of running all the pipelines by varying the weights. Here M1 and M2 utilizes continuous Bag of words (CBoW) as backbone and trained for 1000 iterations M3 performed better when it is trained for 2000 epochs. These Figures also present a detailed comparison of the best multimodal technique M4 with single modality models M3, M2 and M1. It is to be noted that M4 performed the best.

It is to be noted that M2 performed the best among single modality models M3, M2 and M1 for all the weights except when w1 = 0.9 and w2=0.1 where M1 performed better than M2 with recall rate percent 60%.

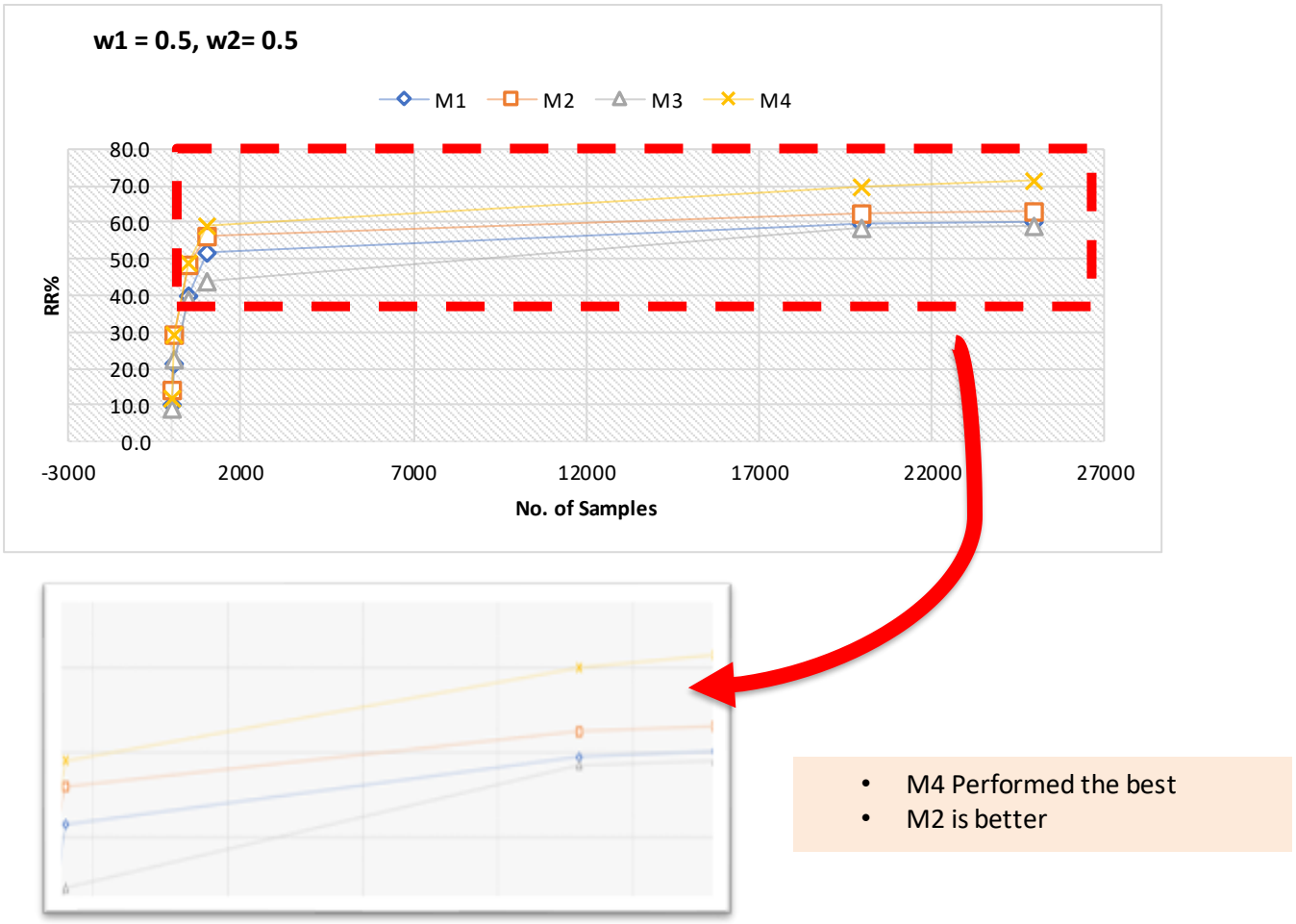
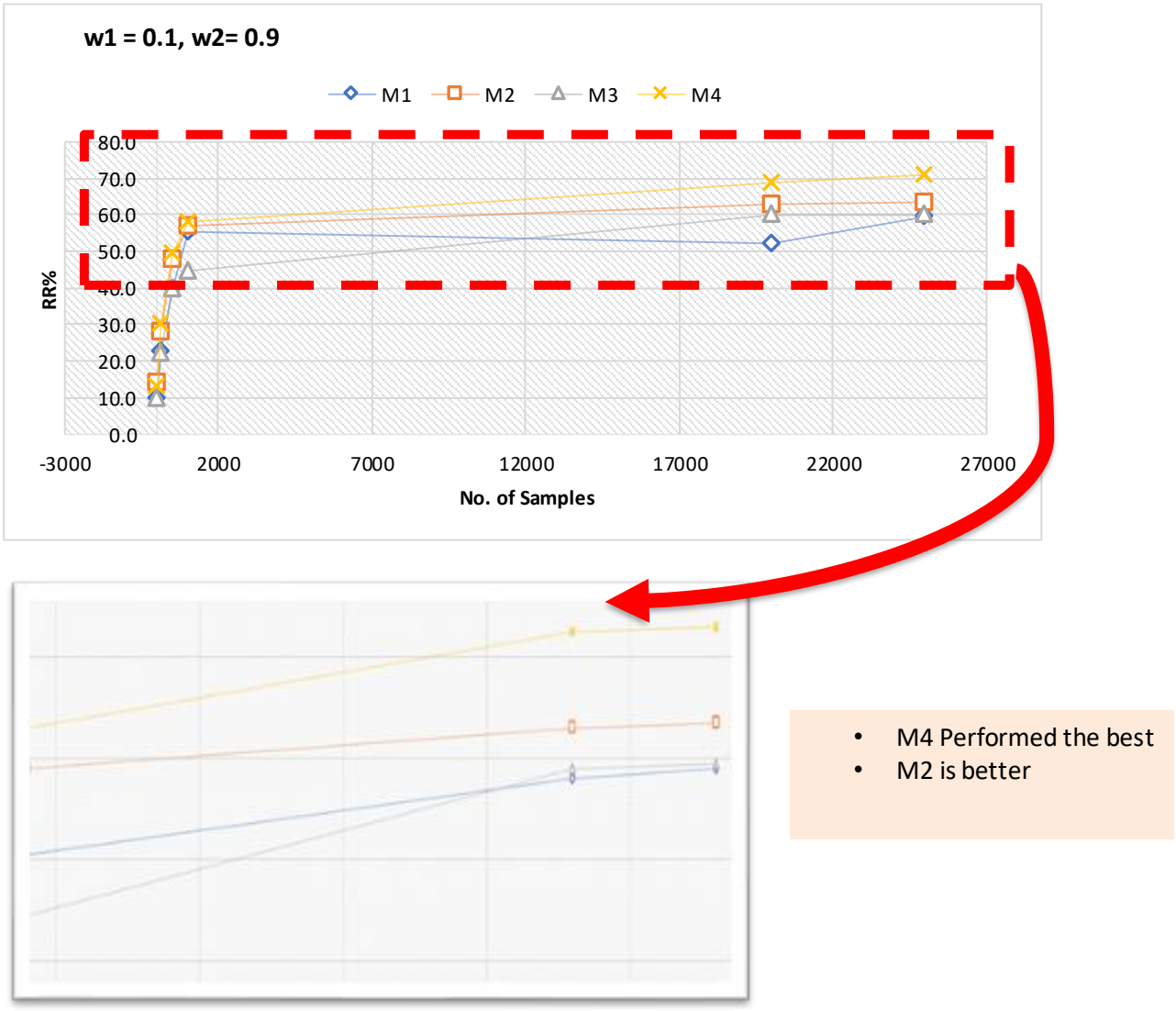


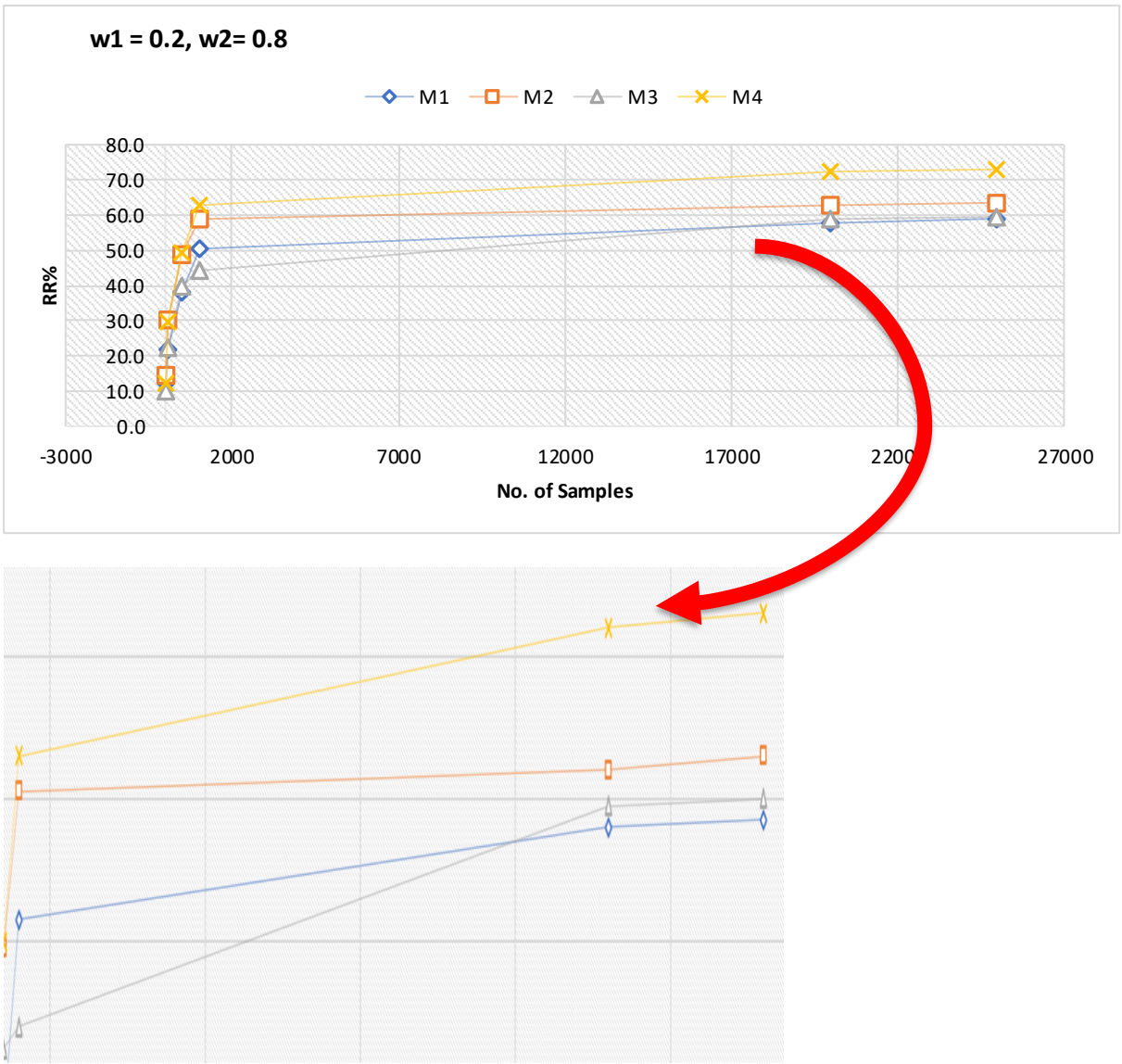
Figure 10. RR% Comparison of M1,M2,M3 & M4 for w1=0.5, w2=0.5

Figure 11. RR Comparison of M1,M2,M3 & M4 for w1=0.1, w2=0.9



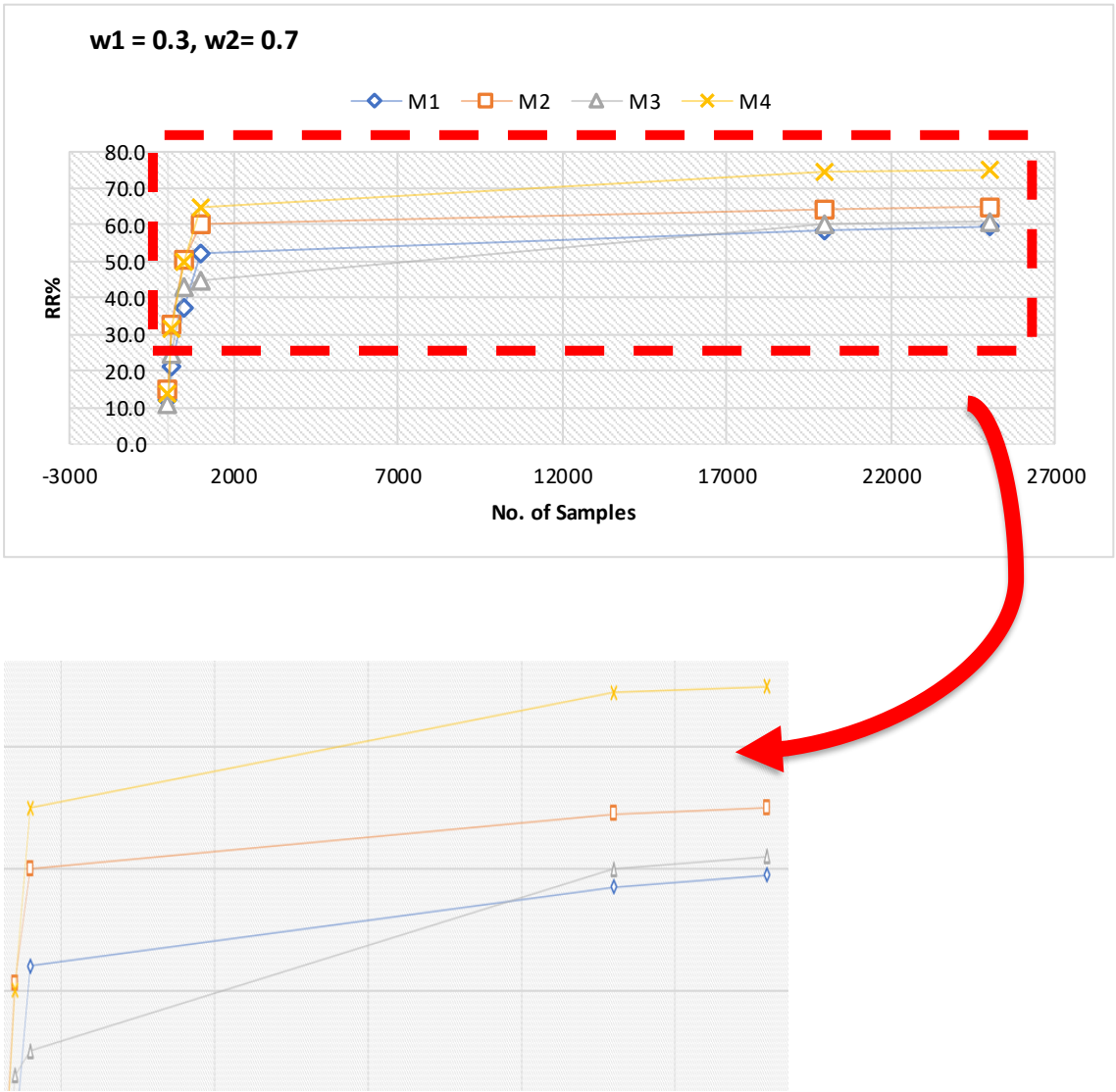
It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 12. RR Comparison of M1,M2,M3 & M4 for w1=0.2, w2=0.8



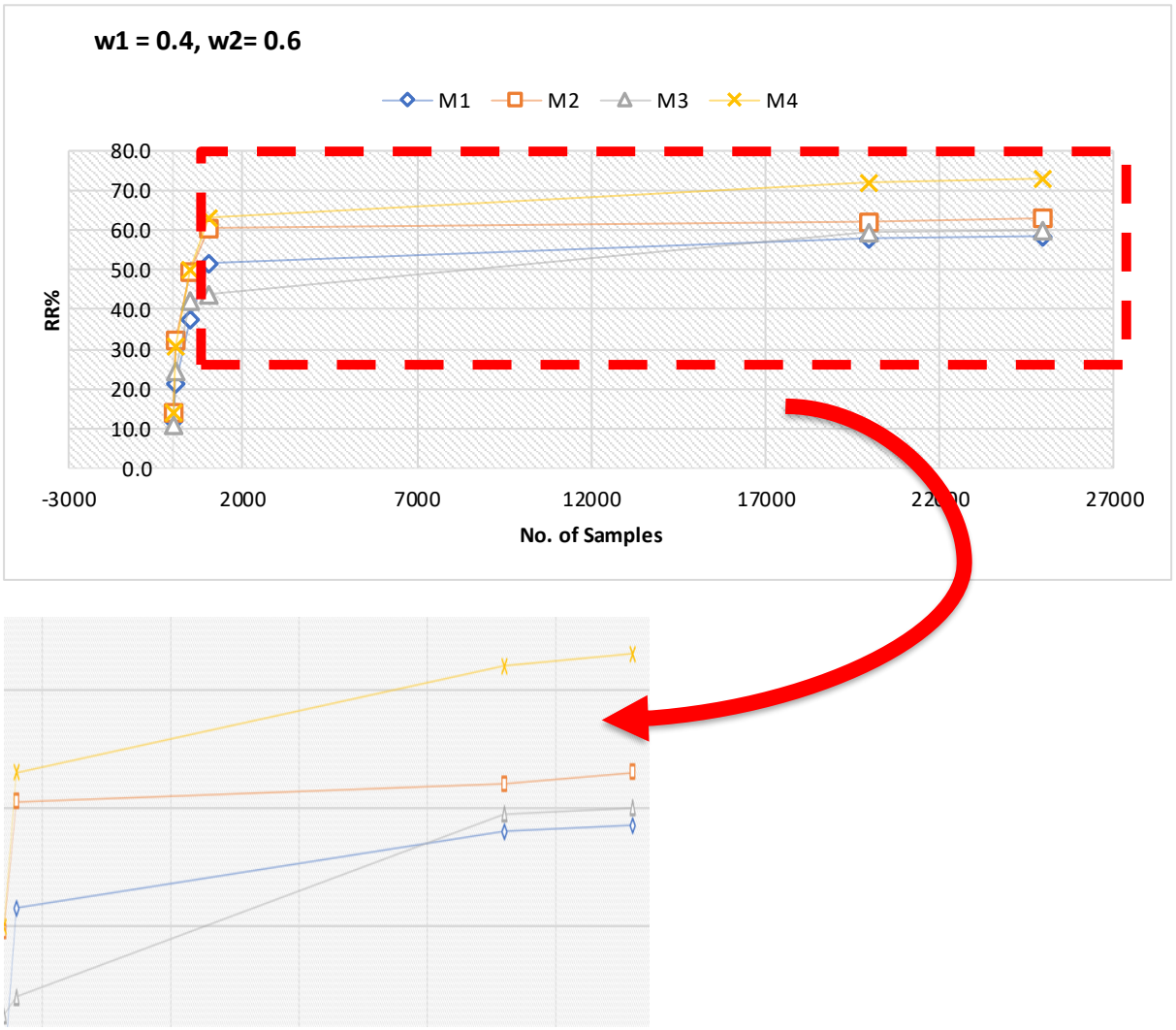
It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 13 RR Comparison of M1,M2,M3 & M4 for w1=0.3, w2=0.7



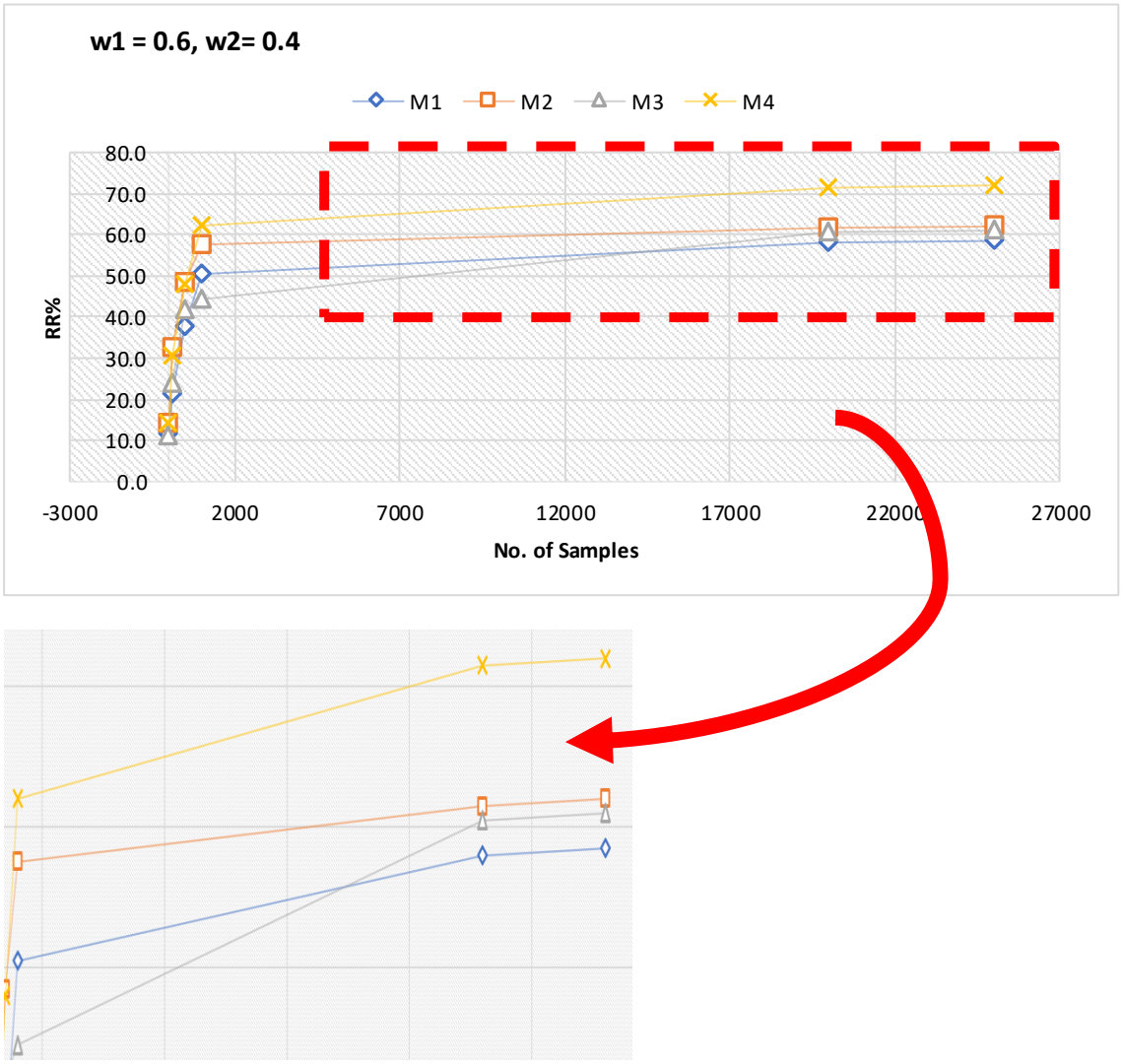
It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 14 RR Comparison of M1,M2,M3 & M4 for w1=0.4, w2=0.6



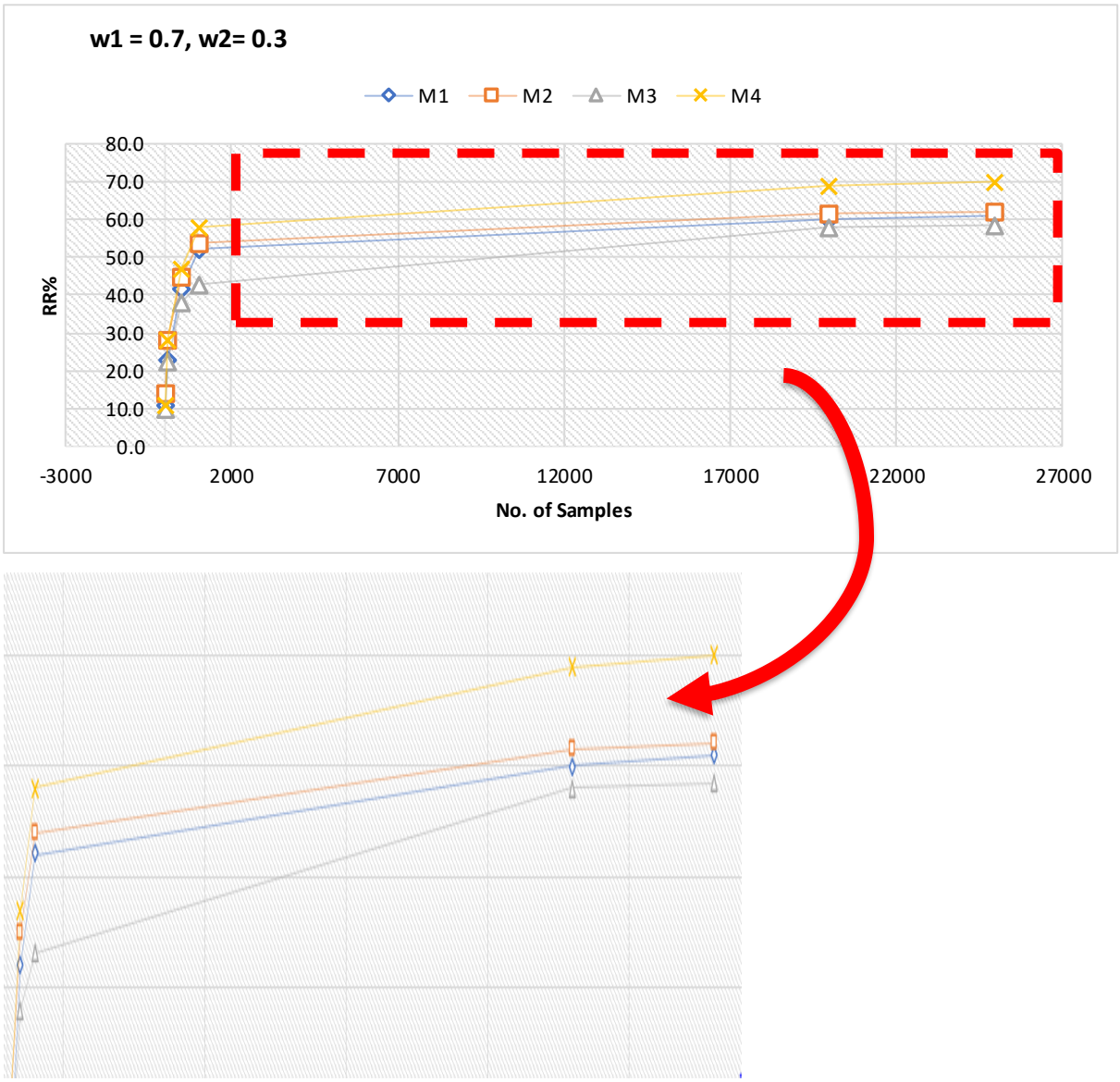
It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 15. RR Comparison of M1,M2,M3 & M4 for w1=0.6, w2=0.4



It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 16. RR Comparison of M1,M2,M3 & M4 for w1=0.7, w2=0.3



It can be seen that M4 performed the best and M2 is better as compared to M1 and M3

Figure 17. RR Comparison of M1,M2,M3 & M4 for w1=0.8, w2=0.2

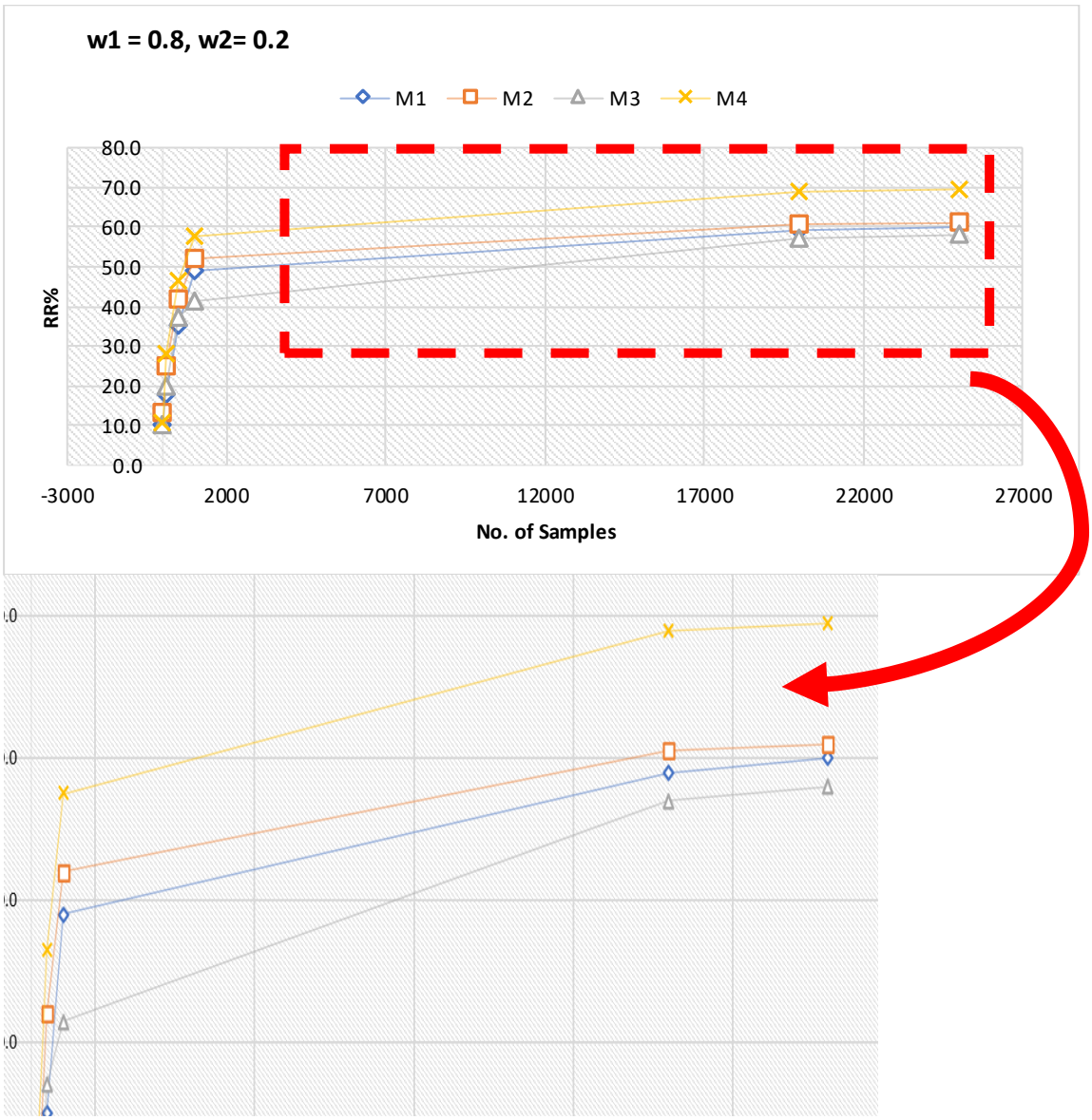


Figure 18. RR Comparison of M1,M2,M3 & M4 for w1=0.9, w2=0.1

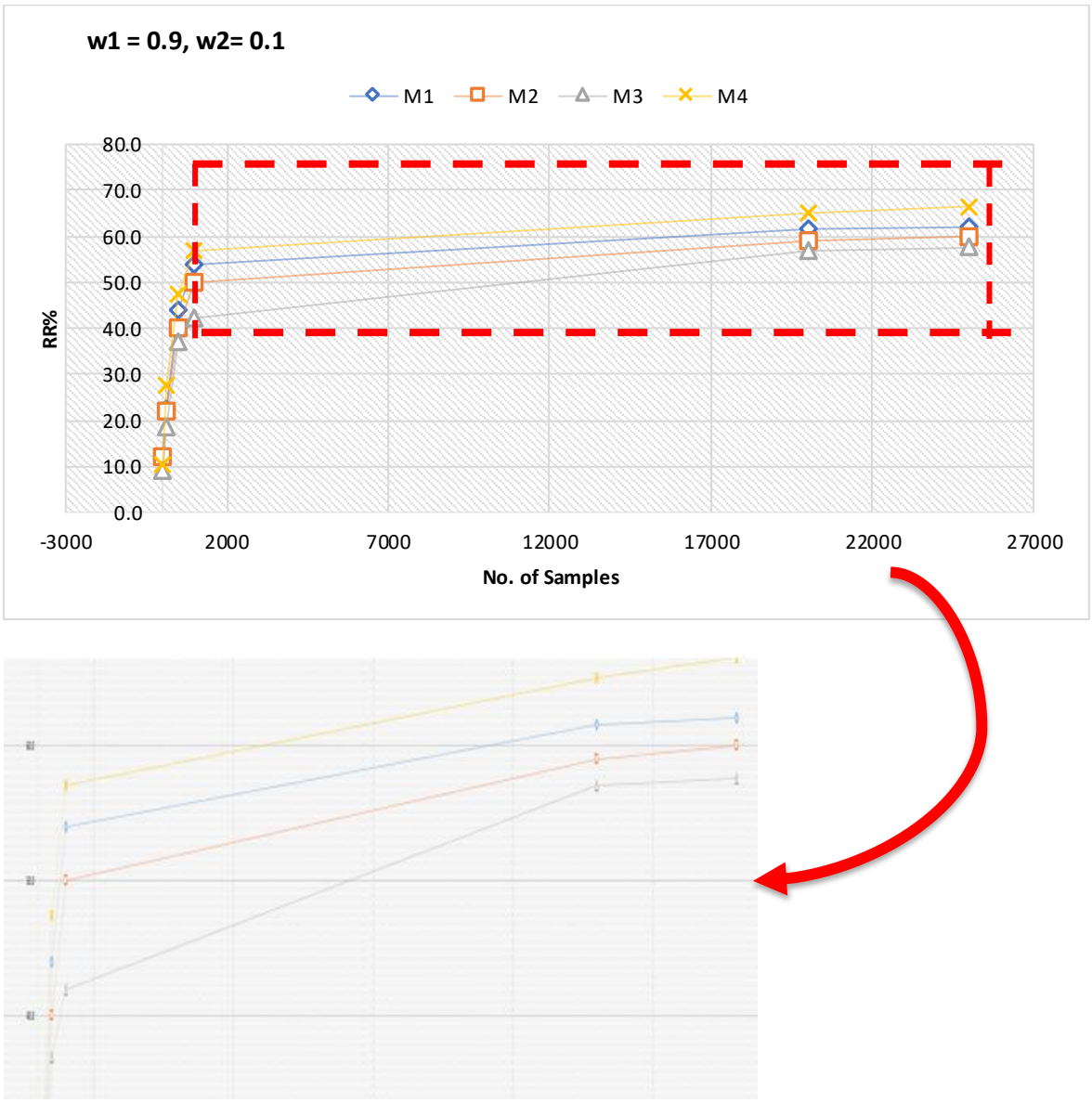


Figure 15-16 corroborates the findings of M4 performance across all weights for different sample size. Increase in performance has been seen with increase in sample size. When w_1=0.3 and w_2=0.7 with RR percent of 75%, the proposed two-tier strategy with M4 performed the best at iteration.

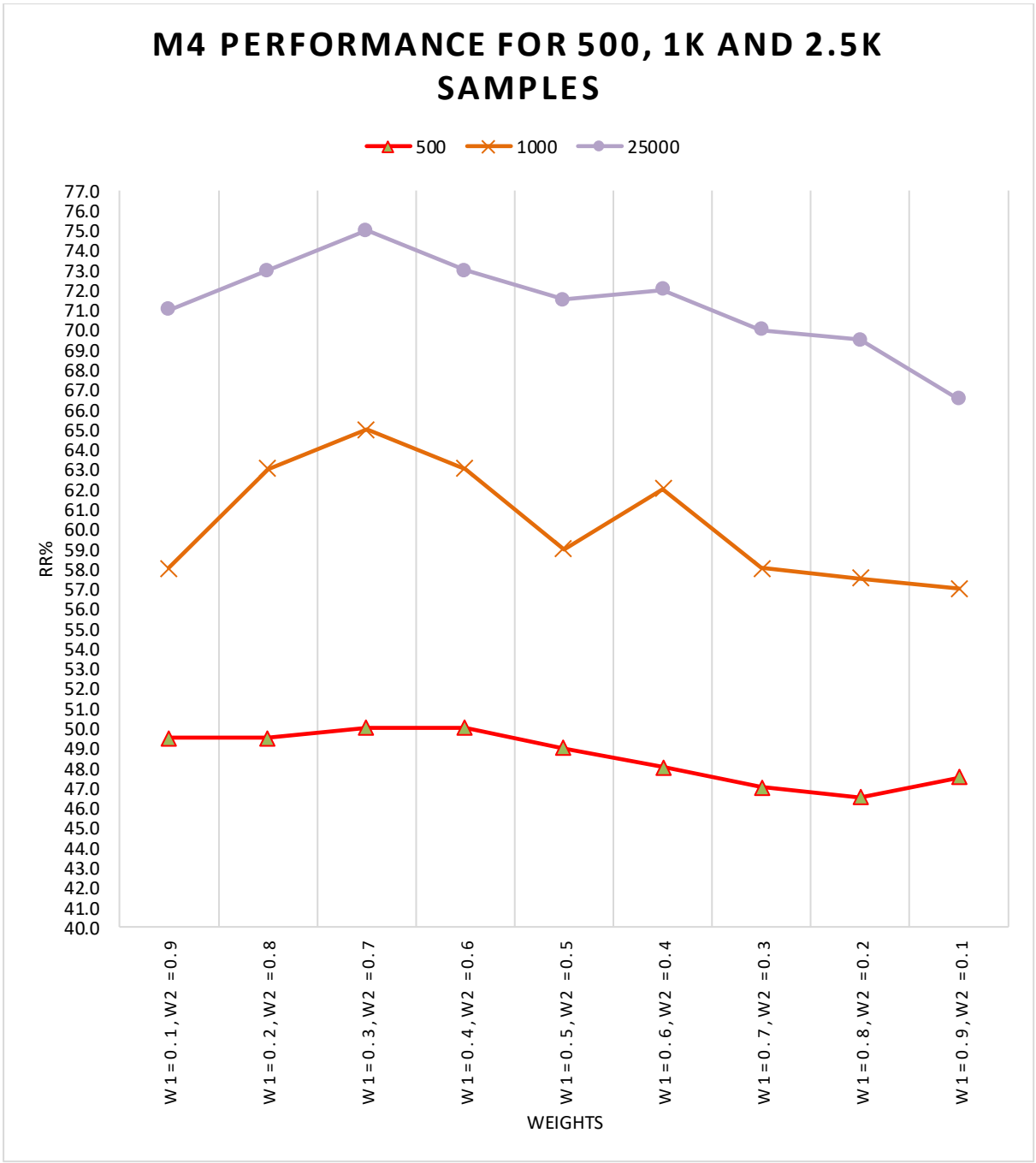


Figure 19. M4 Performance

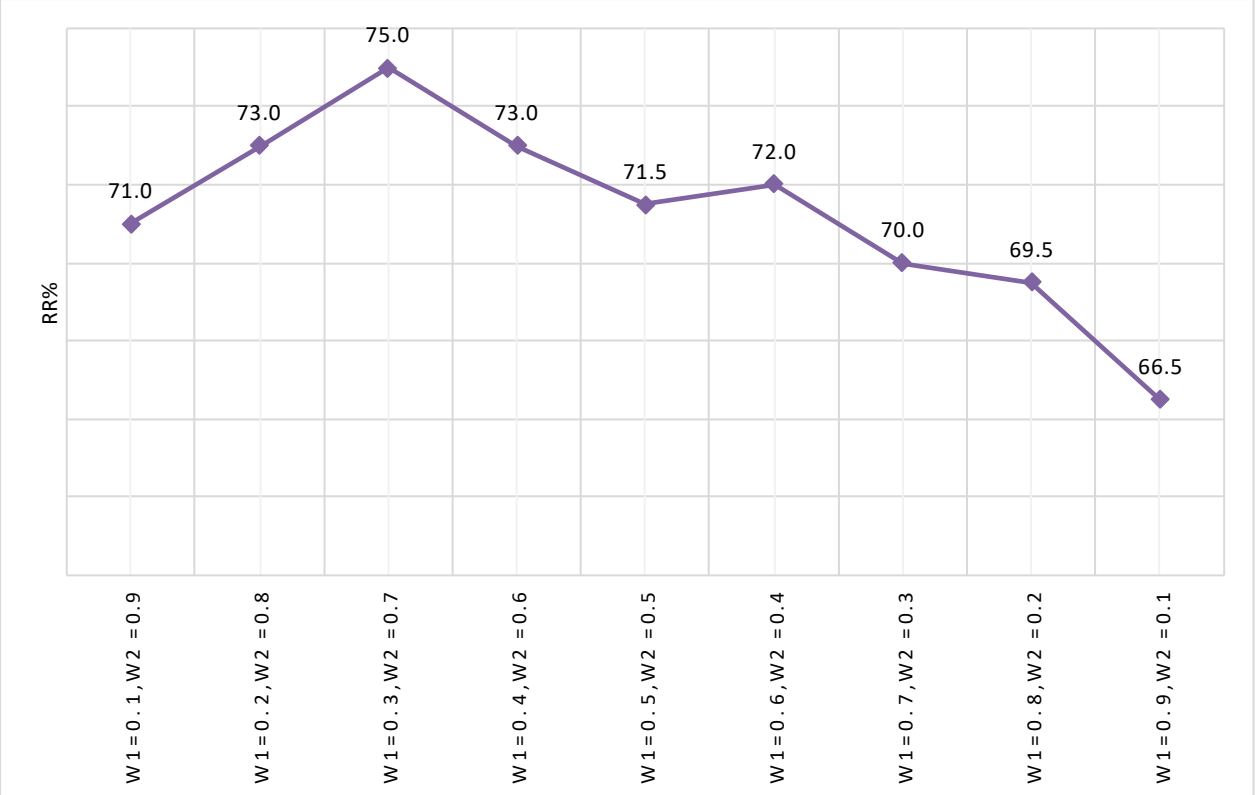


Figure 20. M4 Trend for 2.5K Samples

A decline in the trend has been seen when weightage of w1 has been increased and w2 is decreased which determines that for M4 whenever more weights are assigned to cosine similarity the model M4 has performed better

CHAPTER 5: CONCLUSION

This research proposes a two-tier method based on topic based clustering done by LDA approach, multimodal representation of text using W2V, FT, GloVe and a unified text similarity measure utilizing Cosine and Euclidean similarities to solve this challenge. The Eclipse dataset, which contains over 80,000 bug reports and includes both master and duplicate reports, is used to validate the suggested method. This investigation focuses primarily on the report descriptions in order to identify duplication. For Top-N proposals, the recommended two-tier technique has achieved a 75% recall rate, which is higher than the traditional one-on-one classification model.

CHAPTER 6: FUTURE WORK AND RECOMMENDATIONS

The LDA-based topic modelling and strategy for classification which is recommended outperforms the traditional one-on-one similarity-based classification method across all models. In the future, it will be used to investigate improved similarity assessments and representation methodologies of documents

References

[1] R. S. Pressman, Software engineering: a practitioner's approach. Palgrave Macmillan, 2005..

[2] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," Found. Empir. Softw. Eng. Leg. Victor R Basili, vol. 426, p. 37, 2005..

[3] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in Proceedings of the 29th international conference on Software Engineering, 2007, pp. 499–510.

[4] N. Serrano and I. Ciordia, "Bugzilla, ITracker, and other bug trackers," IEEE Softw., vol. 22, no. 2, pp. 11–13, 2005..

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, 2005, pp. 35–39., vol. pp. 361–370.

[6] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in Proceedings of the 28th international conference on Software engineering, 2006, pp. 361–370., vol. pp. 35–39.

[7] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on, 2008, pp. 52–61., Vols. 7-15.

[8] Y. C. Cavalcanti, P. A. da M. S. Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, "The bug report duplication problem: an exploratory study," Softw. Qual. J., vol. 21, no. 1, pp. 39–66, 2013.

[9] Y. C. Cavalcanti, E. S. de Almeida, C. E. A. da Cunha, D. Lucredio, and S. R. de Lemos Meira, "An initial study on the bug report duplication problem," in Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on, 2010, pp. 264–267.

[10] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful… really?," in Software maintenance, 2008. ICSM 2008. IEEE international conference on, 2008, pp. 337–345.

[11] C. R. Reis and R. P. de Mattos Fortes, "An overview of the software engineering process and tools in the Mozilla project," in Proceedings of the Open Source Software Development Workshop, 2002, pp. 155–175.

[12] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in Software Engineering Conference (APSEC), 2010 17th Asia Pacific, 2010, pp. 366–374.

[13] F. Naumann and M. Herschel, "An introduction to duplicate detection," Synth. Lect. Data Manag., vol. 2, no. 1, pp. 1–87, 2010..

[14] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," National Institute of Standards and Technology, RTI Project, vol. 7007, no. 011, pp. 429–489, 2002.

[15] J. Sutherland, "Business objects in corporate information systems," ACM Computing Surveys (CSUR), vol. 27, no. 2, pp. 274–276, 1995.

[16] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? a two-phase recommendation model," IEEE Transactions on Software Engineering, vol. 39, no. 11, pp. 1597–1610, 2013.

[17] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?," IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 618–643, 2010.

[18] F. Elberzhager, J. Münch, and V. T. N. Nha, "A systematic mapping study on the combination of static and dynamic quality assurance techniques," Inf. Softw. Technol., vol. 54, no. 1, pp. 1–15, 2012.

[19] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: A systematic mapping study," Inf. Softw. Technol., vol. 56, no. 10, pp. 1200–1218, 2014.

[20] M. Unterkalmsteiner, T. Gorschek, A. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement—a systematic literature review," IEEE Trans. Softw. Eng., vol. 38, no. 2, pp. 398–424, 2012.

[21] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” Inf. Softw. Technol., vol. 64, pp. 1–18, 2015.

[22] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, “An approach to detecting duplicate bug reports using natural language and execution information,” in Software Engineering, 2008. ICSE’08. ACM/IEEE 30th International Conference on, 2008, pp. 461–470..

[23] Y. Song, X. Wang, T. Xie, L. Zhang, and H. Mei, “JDF: detecting duplicate bug reports in Jazz,” in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, 2010, pp. 315–316.

[24] H. Gu, L. Zhao, and C. Shu, “Analysis of duplicate issue reports for issue tracking system,” in Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on, 2011, pp. 86–91..

[25] T. Prifti, S. Banerjee, and B. Cukic, “Detecting bug duplicate reports through local references,” in Proceedings of the 7th International Conference on Predictive Models in Software Engineering, 2011, p. 8.

[26] N. Kaushik and L. Tahvildari, “A comparative study of the performance of IR models on duplicate bug detection,” in Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on, 2012, pp. 159–168..

[27] J. Zhou and H. Zhang, “Learning to rank duplicate bug reports,” in Proceedings of the 21st ACM international conference on Information and knowledge management, 2012, pp. 852–861.

[28] S. Banerjee, Z. Syed, J. Helmick, and B. Cukic, “A fusion approach for classifying duplicate problem reports,” in Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on, 2013, pp. 208–217.

[29] N. Tomasev, G. Leban, and D. Mladenic, “Exploiting hubs for self-adaptive secondary re-ranking in bug report duplicate detection,” in Information Technology Interfaces (ITI), Proceedings of the ITI 2013 35th International Conference on, 2013, pp. 131–136..

[30] K. Liu, H. B. K. Tan, and H. Zhang, “Has this bug been reported?,” in Reverse Engineering (WCRE), 2013 20th Working Conference on, 2013, pp. 82–91..

[31] J. Lerch and M. Mezini, “Finding duplicates of your yet unwritten bug report,” in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, 2013, pp. 69–78.

[32] I. Chawla and S. K. Singh, “Performance evaluation of vsm and lsi models to determine bug reports similarity,” in Contemporary Computing (IC3), 2013 Sixth International Conference on, 2013, pp. 375–380..

[33] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu, “Search-based duplicate defect detection: an industrial experience,” in Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, 2013, pp. 173–182.

[34] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, “Enhancements for duplication detection in bug reports with manifold correlation features,” J. Syst. Softw., vol. 121, pp. 223–233, 2016..

[35] H. Rocha, G. De Oliveira, H. Marques-Neto, and M. T. Valente, “NextBug: a Bugzilla extension for recommending similar bugs,” J. Softw. Eng. Res. Dev., vol. 3, no. 1, p. 3, 2015..

[36] F. Thung, P. S. Kochhar, and D. Lo, “DupFinder: integrated tool support for duplicate bug report detection,” in Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 871–874.

[37] A. Tsuruda, Y. Manabe, and M. Aritsugi, “Can we detect bug report duplication with unfinished bug reports?,” in Software Engineering Conference (APSEC), 2015 Asia-Pacific, 2015, pp. 151–158..

[38] C. Jingliang, M. Zhe, and S. Jun, “A data-driven approach based on LDA for identifying duplicate bug report,” in Intelligent Systems (IS), 2016 IEEE 8th International Conference on, 2016, pp. 686–691.

[39] K. K. Sabor, A. Hamou-Lhadj, and A. Larsson, “DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports,” in Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on, 2017, pp. 240–250..

[40] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, “Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports,” in Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on, 2016, pp. 127–137.

[41] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, “Towards more accurate retrieval of duplicate bug reports,” in Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on, 2011, pp. 253–262.

[42] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, “Duplicate bug report detection with a combination of information retrieval and topic modeling,” in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2.

[43] Y. Tian, C. Sun, and D. Lo, “Improved duplicate bug report identification,” in Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on, 2012, pp. 385–390..

[44] A. Alipour, A. Hindle, and E. Stroulia, “A contextual approach towards more accurate duplicate bug report detection,” in Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 183–192.

[45] C.-Z. Yang, H.-H. Du, S.-S. Wu, and X. Chen, “Duplication detection for software bug reports based on bm25 term weighting,” in Technologies and Applications of Artificial Intelligence (TAAI), 2012 Conference on, 2012, pp. 33–38..

[46] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, “Detecting duplicate bug reports with software engineering domain knowledge,” J. Softw. Evol. Process, vol. 29, no. 3, 2017..

[47] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, “A discriminative model approach for accurate duplicate bug report retrieval,” in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, 2010, pp. 45–54..

[48] Q. Wu and Q. Wang, “Natural language processing based detection of duplicate defect patterns,” in Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, 2010, pp. 220–225..

[49] M.-J. Lin and C.-Z. Yang, “An Improved Discriminative Model for Duplication Detection on Bug Reports with Cluster Weighting,” in Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual, 2014, pp. 117–122.

[50] P. N. Minh, “An Approach to Detecting Duplicate Bug Reports using N-gram Features and Cluster Chrinkage Technique,” Int. J. Sci. Res. Publ., vol. 4, no. 5, 2014..

[51] A. Lazar, S. Ritchey, and B. Sharif, “Improving the accuracy of duplicate bug report detection using textual similarity measures,” in Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 308–311..

[52] J. Zou, L. Xu, M. Yang, X. Zhang, J. Zeng, and S. Hirokawa, “Automated Duplicate Bug Report Detection Using Multi-Factor Analysis,” IEICE Trans. Inf. Syst., vol. 99, no. 7, pp. 1762–1775, 2016.

[53] R. P. Gopalan and A. Krishna, “Duplicate bug report detection using clustering,” in Software Engineering Conference (ASWEC), 2014 23rd Australian, 2014, pp. 104–109..

[54] N. Klein, C. S. Corley, and N. A. Kraft, “New features for duplicate bug detection,” in Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 324–327..

[55] A. Hindle, A. Alipour, and E. Stroulia, “A contextual approach towards more accurate duplicate bug report detection and ranking,” Empir. Softw. Eng., vol. 21, no. 2, pp. 368–410, 2016.

[56] Thangarajah Akilan, Dhruvit Shah, Nishi Patel, Rinkal Mehta, "Fast Detection of Duplicate Bug Reports using LDA-based Topic Modeling and Classification" in 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC).

[57] N. Limsettho, H. Hata, A. Monden, and K. Matsumoto, "Automatic unsupervised bug report categorization," in Empirical Software Engineering in Practice (IWESEP), 2014 6th International Workshop on, 2014, pp. 7–12.

[58] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel, "ReBucket: a method for clustering duplicate crash reports based on call stack similarity," in Proceedings of the 34th International Conference on Software Engineering, 2012, pp. 1084–1093..

[59] S. Banerjee, Z. Syed, J. Helmick, M. Culp, K. Ryan, and B. Cukic, "Automated triaging of very large bug repositories," Inf. Softw. Technol., vol. 89, pp. 1–13, 2017.

[60] T. Zhang and B. Lee, "A Novel Technique for Duplicate Detection and Classification of Bug Reports," IEICE Trans. Inf. Syst., vol. 97, no. 7, pp. 1756–1768, 2014.

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

[62] H. Rocha, M. T. Valente, H. Marques-Neto, and G. C. Murphy, "An empirical study on recommendations of similar bugs," in Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, 2016, vol. 1, pp. 46–56..

[63] L. Feng, L. Song, C. Sha, and X. Gong, "Practical duplicate bug reports detection in a large web-based development community," in Asia-Pacific Web Conference, 2013, pp. 709–720.

[64] B. Patel, H. Balvantrai Patel, M. Khanvilkar, N. Rajendrakumar Patel, and T. Akilan, "ES2ISL: an advancement in speech to sign language translation using 3D avatar animator," in 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE).

[65] T. Akilan, Q. J. Wu, A. Safaei, and W. Jiang, "A late fusion approach for harnessing multi-cnn model high-level features," in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 566–571, IEEE, 2017.

[66] T. Akilan, Q. J. Wu, and H. Zhang, "Effect of fusing features from multiple dcnn architectures in image classification," IET Image Processing, vol. 12, no. 7, pp. 1102–1110, 2018.

[67] Y. Yang, J. Q. Wu, X. Feng, and A. Thangarajah, "Recomputation of dense layers for the performance improvement of dcnn," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.