Automated Summarizing of Bug reports to Speed-up Software Development/Maintenance Process

Author

Muhammad Irtaza Nawaz Tarar

FALL 2017 - MS-17 (CSE) 00000203705

Supervisor

Dr. Wasi Haider Butt

Co-Supervisor

Dr. Moazzam Khattak

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JAN 2020

Automated Summarizing of Bug reports to Speed-up Software Development/Maintenance Process

Author

Muhammad Irtaza Nawaz Tarar

FALL 2017 - MS-17 (CSE) 00000203705

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

MS Computer Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature:-_____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

JAN 2020

DECLARATION

I certify that this research work titled *"Automated Summarizing of Bug reports to Speed-up Software Development/Maintenance Process"* is my own work under the supervision of Dr. Wasi Haider Butt and co-supervised by Dr. Muazzam Khattak. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

_____

Signature of Student

Muhammad Irtaza Nawaz Tarar

FALL 2017 - MS-17 (CSE) 00000203705

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

Muhammad Irtaza Nawaz Tarar

FALL 2017 - MS-17 (CSE) 00000203705

_____

Signature of Supervisor

Dr. Wasi Haider Butt

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

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

I would like to express my special thanks to my supervisor **Dr. Wasi Haider Butt** and co-supervisor **Dr. Muazzam Khattak** for his generous help throughout my thesis, and for being available even for the pettiest of issues. My thanks for a meticulous evaluation of the thesis, and guidance on how to improve it in the best way possible.

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

It is indeed a privilege to thank my Mother, my father **Nawaz Tarar** and my elder brother **Sajid Tarar** for their constant encouragement throughout my degree and research period. The sense of belief that they instilled in me has helped me sail through this journey. I would like to thank my Family & friends, especially **Mubashir Ali** who has rendered valuable assistance to my study.

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

Dedicated to my exceptional parents, excellent siblings and my best friend whose tremendous support and cooperation led me to this wonderful accomplishment. I am truly indebted to you all.

ABSTRACT

Software's development process can be optimized by using the knowledge about past information about same kind of product or problem. During development process, software's bug repository can provide a great deal of easiness for development team. It can be a rich source of information for developers and other members of development team. Bug reports can provide a great deal of assistance for developers during the process of development. But due to the large size of bug repositories, it is sometimes difficult to take advantage of these artifacts in the available time. One way of helping developers to provide summaries of these reports and provide relevant details only. Once it's decided that this is the required report then one can study the details. We analyzed the previous approaches use for this purpose and realized that there is need of improvement in this research. We used an extractive summarization approach using the unsupervised learning method for this purpose and developed a novel framework to get better results than previous a state of the art systems. As text mining technology advances, many substantial approaches have been proposed to generate optimized summaries for bug reports. In this paper, we have proposed an extractive based methodology for the generation of summaries of bug reports by using the sentence embedding. We used supervised learning technique to generate the summaries. In our proposed methodology the similarity between sentences is calculated by using sentence embedding. After preprocessing, the sentences are converted to vectors of real numbers by sentence embedding. K-mean cluster is used to cluster these sentences. Then we have to select one sentence per cluster. Sentence ranking is used to rank sentences per information they contain and select high rank sentences for summarization. We achieved improved rouge-1 and rouge-2 results than the previous state of the art systems for the bug report summary generation.

Keywords: Summarization, Natural Language Processing, Machine Learning, Software Artifacts, Bug reports

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

This chapter contains a brief introduction of the research performed. The background study is elaborated in **Section 1.1**. The problem statement is specified in **Section 1.2**. **Section 1.3** includes the proposed methodology and **Section 1.4** provides a brief overview to our research contribution. Lastly, the thesis organization is stated in **Section 1.5**.

1.1. Background

Software development cycle has many stages and these stages have different kind of artifacts with them are associated to them. One of these artifacts is software bug reports [1]. But a bug is not despite its name a little animal in the context of Software engineering or software development, but it is something else. We can define Bug related to software [3] as:

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

Software bug consists of the bugs faced by development teams during development. So during the progress of projects, when bugs occur in the software then the person who found the bug would report in the form of document & send it to people in charge of fixing these bugs, error or failures [2]. Software bug resolution consists of different steps as shown in the Figure.1 below:

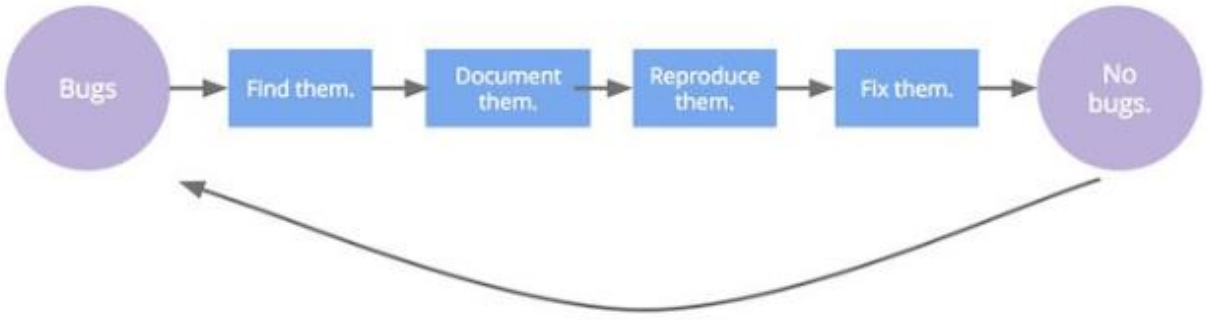


Figure 1. Software bug resolution cycle

Bugs found will be documented and forwarded to the relevant teams or persons. Those teams will analyze these bugs and then fix them. Bug report will contain the information about this complete process started from bug detection to bug resolution. These bug reports will consist of conversation of different members of the team, about how to resolve the bugs and how to

perform this action in short time and budget. These are conversational in nature. An example of a bug report is shown in **Figure.2** below:

BUG REPORT

Comment 1:

The releng test have been DNF for the last couple builds. Running locally we occasionally see crashes in the BrowserTests so we suspect these crashes to be the cause of the DNF.

Comment 2:

Created an attachment (id=135853) [details]

Comment 3:

Pushing this fix for RC2, as it's too risky to make this change for RC1 now. For RC1 we'll comment out the crashing tests so that the other tests can run properly. We have NOT seen this bug affect the usability of Eclipse, only the running of the SWT test suite.

Comment 4:

The problem is that the strategy we use to release pools periodically (in readAndDispatch, createWidget, etc.) to be able to run the test suites, releases the main thread pool created be Device and stored in a thread local dictionary. The pool was not removed from the thread local dictionary and got reused the next time a display was created. The new strategy is to only release the main thread pool when the device count goes to zero.

Comment 5:

Obviously, something this low level needs as much testing as it can get. Can we get a test build run to verify all tests pass/no crashes/no OOM/etc.? Also, you should be doing the two day test pass on a version of SWT that includes the change.

Comment 6:

Created an attachment (id=136083) [details] new fix

Comment 7:

The new fix is simpler, we just make sure the pool in the thread dictionary is always valid, that way when another display is created it will not use a released pool. There was a further problem where the pool was released too early. If we are in call in, we cannot released the top of the stack pool.

Comment 8:

The tests that where showing this problem have been put back (just in HEAD). Fixed > 20090515

Figure 2. Sample Bug Report

The software bug resolution is a cyclic process and it is one of crucial part of software development cycle and is considered among the challenging phases in a software development life cycle [2]. The process of reporting bugs involve conversations of the development team members about the suggestion and handling of the bug. In ideal case these bug reports should be summarized in the end for future guidance but projects time and money constraints doesn't allow this. So if we want to optimize or automate this process of bug resolution, we have to consider the most important factor involved which is natural language [4].

Natural language processing is the field of artificial intelligence which is basically concerned with automatic analysis of plain natural language [5].There are several techniques in the natural language processing e.g. sentence splitting, tokenization, POS tagging [9]. Sentence splitting is basically used to split complex structured sentences into short and easy to understandable sentences [51], [52]. Tokenization is the processing of breaking up the complex long text into units. These small chucks or units are called tokens [49]. Part of speech tagging, commonly known as POS tagging is basically the process of marking or tagging the input words on the basis of part of speech i.e. noun, pronoun, verb, adverb etc. [8]. Entity identification chunking and extraction is all under named-entity recognition natural language technique. It is sub-part of information extraction in which named elements are extracted from plain text [9].

1.2. Problem Statement

Software's development process can be optimized by using the knowledge about past information about same kind of product or problem. During development process, software's bug repository can provide a great deal of easiness for development team. It can be a rich source of information for developers and other members of development team. For example, a developer can see the bug repository to learn how changes were made to software in the past .He can also consult to understand the reported bugs in more details. So when a developer needs to see the bug repository, often he ends up getting a large number of reports either as a result of some search recommender engine [11], [12].In fact, developer needs only few reports that he should follow. Sometimes a developer can have idea about report by reading the title but sometimes he has to read the whole report to determine that if this report is relevant or not. These reports can be very lengthy because it includes the conversations between development team members or other

stakeholders. This can be very hectic process to read the lengthy reports every time he needs to consult bug repository. For instance, a developer is trying to get duplicate reports for a bug #564332 from Mozilla system by using a duplicate report recommender [12]; it gets a report of a total 237 sentences and 5,125 words in the top six bug reports on the recommendation list. There is a dire need to develop an intelligent framework capable of automatically summarizing the bug reports.

1.3. Proposed Methodology

In this thesis, we have proposed solution is bases on Natural Language Processing (NLP). We have explored the existing literature to highlight the standards for bug report summarization.

We analyzed the previous approaches use for this purpose and realized that there is need of improvement in this research. We used an extractive summarization approach using the unsupervised learning method for this purpose and developed a novel framework to get better results than previous a state of the art systems.

Following figure shows the proposed process at abstract level. Basically, what we want to do is simply explained in the following **Figure.3**.

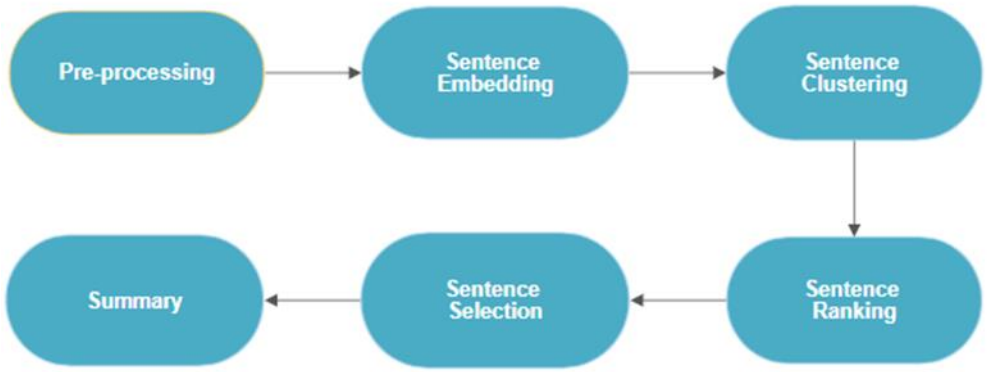


Figure 3. Brief Introduction of the Research Study

The entire research is done in a systematic way. Flow of the research is shown in **Figure. 4.** First of all, we identify the problem, then we propose a solution to the identified problem. Then, we carry out a comprehensive systematic literature review which becomes the foundation of the proposed solution. Researches related to the proposed solution are analyzed and compared.

The proposed work includes a fully automated approach to generate the summaries for bug reports. The proposed methodology has been compared with the previous methodologies to compare the improvement.

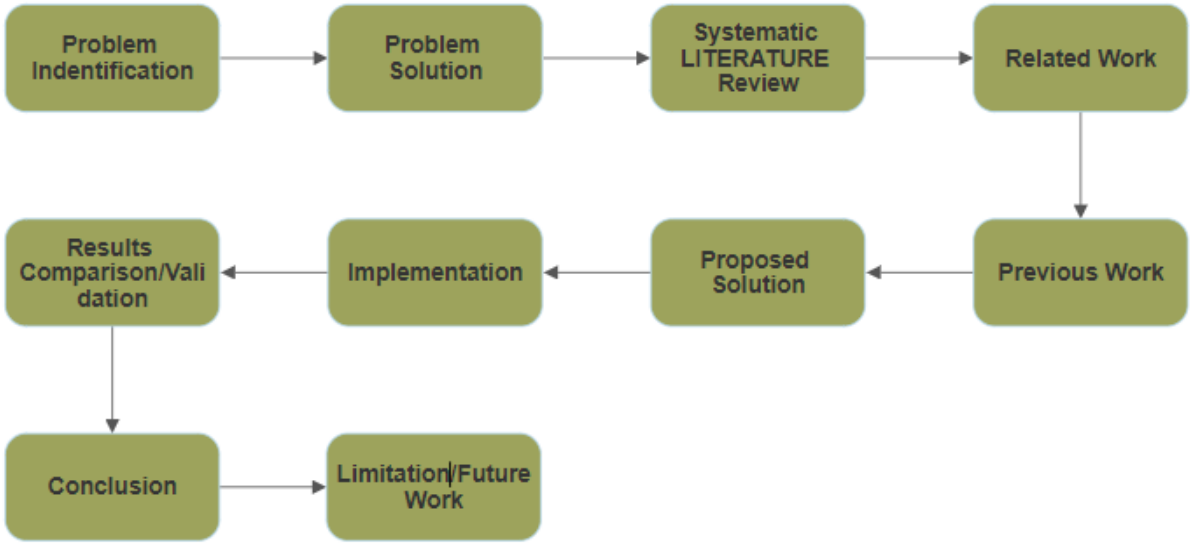


Figure 4. Research Flow

1.4. Research Contribution

The main contributions made by our research work is to save time, cost and other resource in software bug resolution process during software development cycle.

Below are listed the contributions concluded by the proposed approach:

- We have performed a systematic literature review on automated summarization of bug reports. Through this literature review, we identified different techniques used in past for this purpose from researches reported in detail in Chapter 2. Similarly, our systematic literature review has drawn the NLP algorithms used. The concluded results show that our study would provide an advantage to the researchers in future it is the very first systematic literature review carried out on automated bug report summarization. It will help practitioners in this field to overview the results to explore more investigate maturity of this process.
- We have identified analyzed and employed techniques and tools provided by Natural Language Processing for automation of summarization of bug reports

- We have formalized a practical approach that uses techniques offered by Natural language processing.
- We have exploited text mining for the determination and development of a novel framework more specifically rational expression for the transformation rules on basis of formal language theory.
- We have implemented the defined methodology to the standard data set and compared the results with previous state of the art systems to validate the improvements.
- In order to design and develop our tool, Python has been utilized.

1.5. Thesis Organization

Organization of the thesis is represented in **Figure. 5**

Chapter 1: Introduction offers a brief introduction containing the background study, problem statement, research contribution and thesis organization. Chapter 2: Literature Review provides the detailed literature review highlighting the work done in the domain of automated bug report summarization. The systematic literature review is composed of three main sections. First section is review protocol which gives details on the methodology using which the literature review is carried out. Section two offers details on research works, whereas, section three highlights the research gaps that we encountered. Chapter 3: Proposed Methodology covers the details of proposed methodology used for identification of problem. Chapter 4: Implementation presents the detailed implementation regarding the proposed tool. Chapter 5: Validation provides the validation performed for our proposed methodology using nine important case studies. Chapter 6: Discussion and Limitation contains a brief discussion on entire work performed along with limitations to our research. Chapter 7: Conclusion and Future Work concludes the research and recommends a future work for the research.

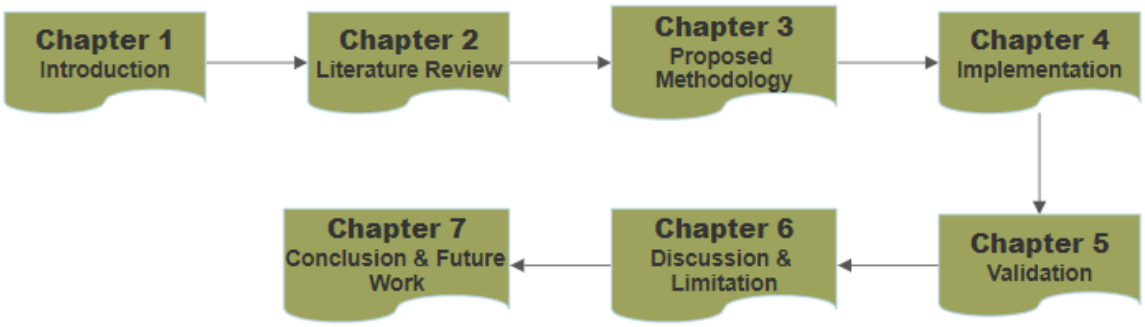


Figure 5. Thesis outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

Natural language Processing techniques have been proved very helpful in optimizing the software development process. It has improved the accuracy and speed of different steps of development process. Summarization of software artifacts is one of application of natural language processing techniques to help the developers or testers. Summarization tools and techniques have been applied to many software artifacts in the past like source code, discussions and bug reports. We present a systematic literature review of the natural language processing techniques applied for the summarization of bug reports. Bug reports are very important for development process because these have valuable knowledge of the problems and their resolution. By summarizing bug reports, a lot of developer's time can be saved during bug triaging when developers are looking for the similar problems from the past. Bug report summarization is done by various methods and techniques and it have helped the developer to save their time and better understanding of the problem at hand. This survey of the past techniques used for the summarization of bug reports will provide useful and wide background knowledge of this research field to the future researchers. This chapter contains the systematic literature review performed for our research. **Section 2.1** presents introduction to the SLR. Research methodology of literature review is explained in **Section 2.2**. The review protocol mechanism is explained in **Section 2.2** and **2.3**. The results, acquired using review protocol mechanism, are presented in **Section 2.4**. Research gaps are presented in the **Section 2.5**. Discussion and limitations are discussed in **Section 2.6**.

2.1. Introduction

Now days, with the increasing amount of data generated electronically on daily basis, there is greater need of generating summary of information provided. Many specialists commit themselves to concentrate the summarization techniques and many researchers have done work in this area. Software's development process can be optimized by using the knowledge about past information about same kind of product. Bertram et al. [28] conducted a study to prove that bug reports are becoming more and more important to software development industry as they provide coordination and communication for involved parties. During development process, software's bug repository can provide a great deal of easiness for development team. It can be a rich source of information for developers and other members of development team. For example, a developer can see the bug

repository to learn how changes were made to software in the past .He can also consult to understand the reported bugs in more details. So when developers need to see the bug repository, often they get a large number of defect reports from some recommender or search engine ([11], [12]).In fact, developer needs only few reports that he should follow. Sometimes a developer can have idea about report by reading the title but sometimes he has to read the whole report to determine that if this report is relevant or not. These reports can be very lengthy because it includes the conversations between development team members or other stakeholders. This can be very hectic process to read the lengthy reports every time he needs to consult bug repository. As bug reports are similar to conversations and G. Murray and G. Carenini [33] have summarize the spoken and written conversations in their study. Further, Lawrie et al. [34] illustrated the value of bug report in software engineering. Many research studies have been conducted in this field. Some researchers have also considered the technical nature of bug reports and they have considered it as an important feature. XiaoyinWang et al. [35] have provided the mechanism of extracting the technical expression s from the bug reports. Some other studies have provided the mechanism of finding the duplicate bug reports using natural language processing to help developers in the process of software development[36][38][39]. He Jiang et al. [40] have provided the research about the importance of authorship feature in bug reports. It can help to improve the bug report summarization. A. Podgurski et al. [41]have provide the way to classify the failure/bug reports to prioritize the reports. Haiduc has generated the summaries of code that contains mostly used terms in classes or methods to describe the method or class [46].Sridhar has proposed the natural language processing technique to generate comments for java method by using the internal structure of method and its statements [47]. Morino has produced the summaries of java classes to understand the purpose of classes by using the internal structure of classes [48]. Basically there are two categories in which we can divide the summarization techniques: Abstractive and Extractive. In extractive, first of all it will prioritize the sentences on the basis of given criteria and then it will select a reasonable amount of sentences from a set of existing sentences to generate summary. An abstractive summarization approach will build the internal semantic representation of sentences and then applies NLP techniques to generate summary [37]. We will concentrate on the extractive approach in this paper as it provides the good summaries in low cost than abstractive approaches and it has shown good results in other domains as well. Many generic extractive summarization techniques have been used for bug reports in previous literature. Usually generic

summarization approaches do not make any assumptions, about the domain knowledge. But it can be very useful in order to produce accurate results if use these domain information by defining the most important information in the document. For example, summarization of conversation-based data. Extractive summarization approaches have been applied to telephonic conversations and emails and meeting conversations [42] [43] [44].Several researchers have explored the bug reports for its conversational nature and its usefulness during development stage of software projects. Sandusky and gasser found that bug repositories are fundamental location for the distribution of responsibilities in bug resolution in software projects [45]. Different paths have been followed to generate extractive summaries of various software artifacts. Many researchers have applied different tools and techniques to produce summaries for bug reports to reduce the time delay during bug triaging. But there is lack of some systematic study that can give the overall understanding to future researcher that which technique is better for summarization. We have developed some research questions for conducting this systematic study of literature:

- How many significant efforts are made to summarize the bug reports from 2000 to 2019?
- What algorithms or specific techniques have been used for this purpose during 2010 – 2019 researches?
- What specific tools have been used during 2000 – 2019 researches?
- What metrics have been used for evaluation of these techniques during 2010 – 2019 researches?

2.2. Research Methodology

Kitchenham presented Systematic Literature Review in 2004 [10], which is used as a guideline for performing this research. He had explained all procedures to do a systematic literature review (SLR). It is a best way to systematically analyze and evaluate all existing researches relevant to our topic and research questions. Therefore, our research involves five levels these are:

- Identification of Problem
- Development of Review protocol

- Defining of Inclusion and exclusion criteria
- Search process using selected databases
- Quality assessment of selected research
- Data extraction and synthesis.

2.3. Methodology/Development of Review Protocol

2.3.1. Inclusion and exclusion criteria

For development of review protocol, first of all we have defined some constraints to make sure that results of research are relevant and accurate. For this purpose we have defined five constraints and we will include or exclude the research based in these five constraints:

1) We will select the research that includes the automated summarization of bug reports and exclude the research papers that are irrelevant to our research topic and our research questions.

2) Only research conducted between 2000 and 2019 will be included in our systematic study and previous researches will be excluded.

3) We will use mostly these four scientific databases for the inclusion of our research work: IEEE, Springer, Elsevier and ACM.

4) For all the research work that is part of more than one database are merge to avoid the redundancy in our results.

5) For all research work that have some defined results and these results are sustained by concrete evidences, will be included

2.3.2. Search process and selection of keywords

For the search process for the systematic studies of our selected research topic, we have done a selection of keywords shown below, to find the research paper in the four scientific databases IEEE, ACM, Springer and Elsevier having high impact factor journals and conference

proceedings; Bug, Defect, Report, Summarization, Natural Language Processing, Extractive summarization, Abstractive summarization.

2.3.3. Execution of Search String:

Key words have been used with the number of filters to find the research papers related to our topic and criteria. Search process has been done by using the four selected scientific databases by using filters like research publications between 2008-2019 and by using the AND/OR operators to find the publication containing the one or a combination of keywords. Moreover Table.01 contains the execution of search strings with the filters or conditions. After the execution of search strings we have selected a total 6,183 papers that were seem to related to our research topic and then further we have analyze and scrutinize these research papers based on our further inclusion criteria by filtering on the basis of their title, abstract and general study and detailed study of publications. Further process is illustrated in Figure.6.

Table 1. Execution of Search String

Sr. No.	**Key Words/Search Terms**	**Operators**	**Number of Research Papers**			
			IEEE	**Springer**	**ACM**	**Elsevier**
01	Bug Report and Summarization	AND	18	321	23	6
		OR	6,851	7,658	22,129	6,158
02	Defect Report and Summarization	AND	6	589	11	8
		OR	9,443	8,556	18,986	5,012
03	Extractive summarization and Bug	AND	5	19	10	14
		OR	7,053	789	18,993	11,143
04	Bug and abstractive Summarization	AND	0	8	14	0
		OR	7,546	659	21,723	19,004
05	Summary and Natural Language Processing	AND	893	10931	156	954
		OR	98,511	78,198	94,159	78,259
06	Bug NLP	AND	21	401	14	96
		OR	15,892	19,258	4,241	5,127

Further we have used an advance search to further refine the results from the defined four scientific. The steps of our research process using the defined key words from four databases is shown in the Error! Reference source not found..

1) We identified the several "search terms' in the selected four scientific databases. Then after scrutinizing these and 7,640 search results selected according to the rejection and selection criteria.

2) 3,713 studies were rejected on the basis of their Title, according to the exclusion and inclusion criteria.

3) 2,963 studies were rejected on the basis of evaluation of their Abstract, according to the exclusion and inclusion criteria.

4) 964 studies were selected to perform general study and then further, 871 research works were discarded according to the exclusion and selection criteria based on general study.

5) Then after the thorough study of 93 studies and 62 researches were discarded according to the criteria.

6) At the last, 31 researches were selected having completely agreement with our rejection and selection criteria.

2.3.4. Quality checking

Quality cehcking is done by the checking that the facts evaluation of study is grounded on the tangible evidences and theoretic understanding without some unclear declarations. Also quality was ensured by using the trusted sources and to improve the accuracy and qulaity of our search results four quality scientific databases, Springer, ACM, IEEE and Elsevier were selected.

2.3.5. Data Extraction and synthesis

We have find the different algorithms and tools used to summarize the bug reports, by reviewing the researches we get after the execution of search process described in section 2.3.We have perform the systematic review of data from the selected researches to extract and analyze the data. First of all we record the Bibliographic data from the researches and it includes the Author, Title, publication year, publication type etc. Then we extracted the technical data from the researches by identifying that what algorithms have been used for summarization of bug reports in these researches. Then we recorded the summary of techniques used for this purpose. After that we have

analyze the results of these summarization techniques. We also analyze that what performance metrics have been used to check the quality of results and what was the nature/kind of summarization techniques used for the summarization of bug reports. Results of Data extraction and analysis have been shown in the next section.

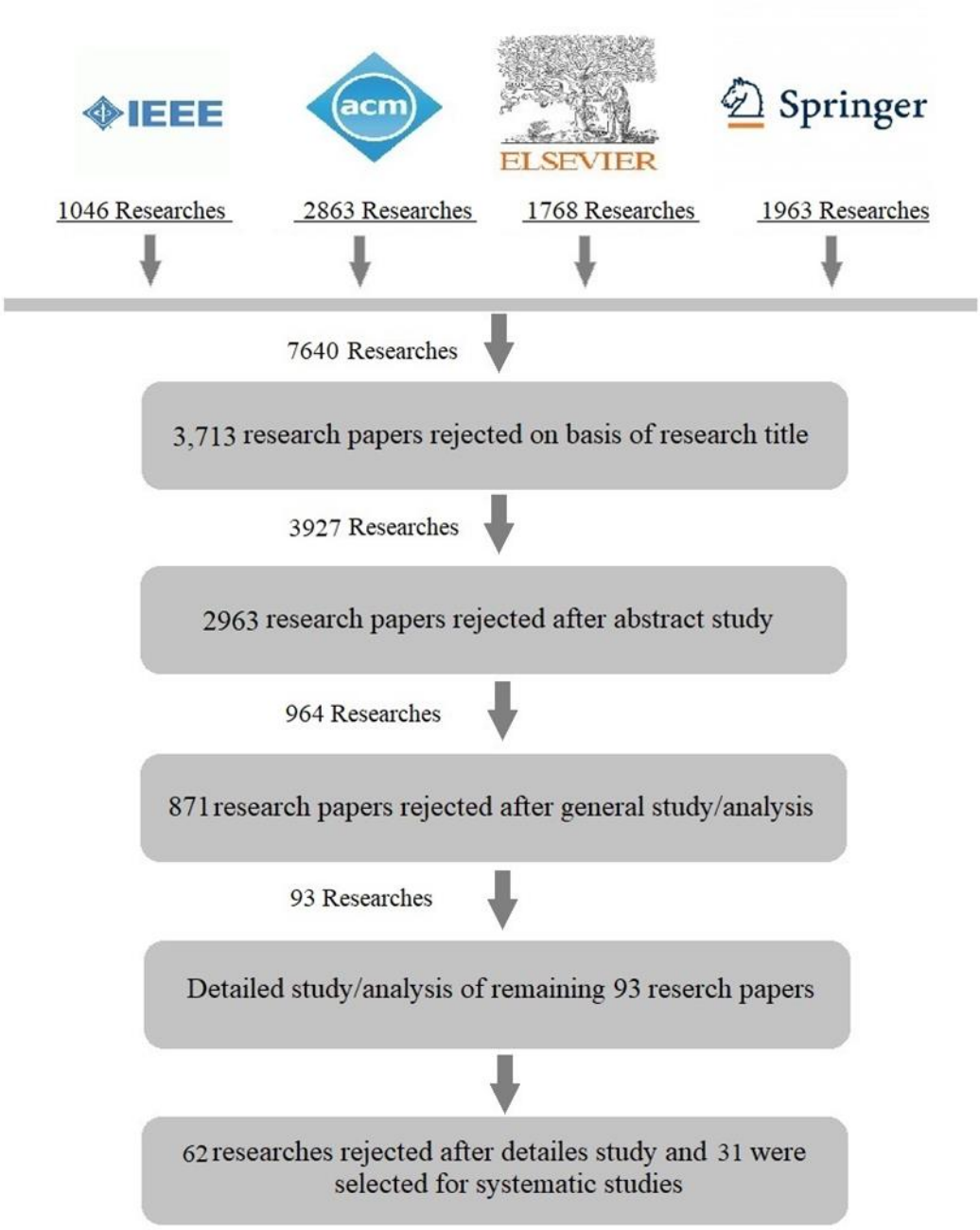


Figure 6. Search Process

2.4. Results

We have identified the various kinds of algorithms by the systematic studies of our selected researches and these have been shown in the Table.2. Different tools were identified during the study for process of summarizing bug reports and have been shown in the Table.3. Performance metrics are used for the calculating the accuracy of different techniques used in the machine learning. We have identified the performance metrics used for the summarization process shown in Table.4. All of our selected researches have used the extractive summarization approach for the summarization of bug reports and none have used abstractive summarization. [13] [14] [16] have used the supervised learning techniques and other have used the unsupervised techniques for this purpose. Rastkar [13] [14] proposed the method to summarize the reports on the basis of their conversational nature. Their methodology involves a BRC (Bug report Corpus) summarizer. They used the twenty four features to rank the sentences of conversations and then uses a logistic regression to prioritize the sentences to be used in the summary. Senthil Mani et al. [15] have used the noise reduction module with three classes; code, question and investigative. After that they applied the four unsupervised learning algorithms; Centroid [29], Grass Hopper [30], Maximum Marginal Relevance [22] and Diverse Rank [31] to find the best sentences for the summary. Cheng-Zen YANG et al. [16] have used the noise reduction module same as [15] but with two new classes; anthropogenic and procedural information. And then they applied the logistic regression to find the sentences to be included in summary. Lotufo [17] provides a novel model and his proposed methodology includes the summarization of discussion threads of bug reports. In first step he used the network model and applied it to the discussion thread and then he applied PageRank method to find the relation between sentences. In addition to relation between the sentences, he also considers the similarity measure between the sentences and the topic or description of bug reports. If similarity measure is high then sentence will have more priority to be considered for summary. Xiaochen Li et al. [18] have used the Deepsum algorithm which used the unsupervised network training to compute the scores for sentences and then sentences with best scores are included in the summary. Jiang [19] has used the approach known as PRST (PageRank Summarization Technique). He also used the additional information for generation of summaries from the relevant duplicate bug reports. Hi Jiang et.al [22] introduced new eleven attributes with a method crowded-

attribute (CA) and then used logistic regression to compute the scores for sentences for the summary on the base of these new eleven attributes.

Table 2. Identified Algorithms

Sr. No.	Algorithms	References
1	Logistic Regression	[33],[34] ,[36] ,[22] ,[19] ,[25] ,[33]
2	Centroid	[15],[22] ,[29]
3	Page Rank	[17],[20] ,[21] ,[19]
4	Grass Hopper	[15],[22] ,[30]
5	Deep sum	[18]
6	Maximum Marginal Relevance	[15],[22]
7	Diverse Rank	[15],[22],[31]

Table 3. Identified Tools

Sr. No.	Tools	References
1	Porter Stemmer	[16],[17] ,[18] ,[19] ,[20]
2	NLP Sanford Parser	[15],[16]
3	Stanford Tokenizer	[16]
4	Lingpipe Tokenizer	[19]
5	Island Grammar Parser	[24]
6	Infozilla	[18],[23],[32]
7	Fuzzy Java code Parser	[23]
8	Snow Ball Stemmer	[21]

Table 4. Identified Performance Metrics

Sr. No.	Performance Metrics	References
1	Precision	[13],[14],[15],[16],[17],[18],[19],[21],[22],[26],[36]
2	Rouge	[18],[27]
3	Recall	[13],[14],[15],[16],[17],[18],[19],[21],[22],[26],[36]
4	F-measure	[13],[14],[15],[16],[18],[19],[21],[22],[26]
5	Accuracy	[26]
6	AUROC	[13],[14],[33]
7	Pyramid Precision	[13],[14],[15],[17],[18],[19],[22]

2.5. Research Gaps

We have conduct rigorous research about the automated bug report summarization and identified the different algorithm used for this purpose in the past. We use a proper systematic literature review protocol to find the previously used techniques and algorithms. But we have found that this research area needs a lot of improvements. Also, our proposed methodology gives very good results in this summarization of bug reports and it has been never used before. Therefore we have proposed this methodology to improve the results and quality of system generated summaries.

2.6. Discussion and Limitations

Automation has revolutionized the modern day processes by reducing the time delays and introduced the great accuracy. Natural language processing has been proven great beneficial in software development processes and has helped the software developer with in many fields. One such field is the summarization of software artifacts to save the time of developers needed to analyze these artifacts. Many efforts have been made to summarize the different software artifacts like source code, mailing lists, developers' discussion or bug reports. This paper investigates the

efforts made for the summarization of bug reports and xx researches were selected from 2000 to 2019 for the systematic study. Different algorithms and tools were identified and research results and their performance metrics were analyzed and recorded for the systematic mapping. Finally, it is concluded that good results have been achieved by the efforts made so far but still there is need for improvement in techniques and more sophisticated tools are needed to increase the accuracy of results to cope with the requirements of real time systems. Although we have followed a proper research methodology but still there is a chance that we may have missed some of researches in other databases are missed, because we have used only four scientific databases to ensure the quality of our research. And may be our related studies may have different titles or content and so we could have missed these researches on the basis of title rejection criteria. Although we used the advanced search to check the abstract and other parts of research too, but still there is a minor chance that some fraction of researches is missed.

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

This chapter contains details of the proposed methodology. **Section 3.1** discusses the targeted core concepts explanation, **Section 3.2** provides the detailed proposed methodology.

3.1. Core Concepts Explanation

3.1.1 NLP

Natural Language processing usually referred as NLP is an Artificial Intelligence Branch. This field is related to helping computers to interpret the human language, understand it and then able to manipulate it [49]. Basically, Natural Language processing is actually a subfield of Computer Science, Artificial Intelligence and Linguistics and it is related to the interaction between human's languages and computer. It's about how to teach computers that how to analyze and process the natural language. This field can be further breakdown in speech recognition, natural language understanding, natural language summarization and natural language generation etc. Natural language processing is used to help the developers to manage and organize the work knowledge to perform their tasks like summarization, named entity relationship, translation, information retrieval or relationship extraction, speech recognition and topic segmentation etc.[50]. This field helps computers and create automated systems that can understand and analyze a human languages like Arabic, Latin or English etc.

3.1.2. Tokenization

Tokenization is basically a dividing task that is used to divide a lengthy text in to smaller parts known as tokens. As natural language processing is used for building automated systems like text classification, automated Chabot, language translations and sentiment analysis etc. It is important to understand and analyze the patterns occurring in the text to achieve the before mentioned objectives. So tokenization plays a vital role in finding these kind of patterns and also tokenization is considered as base step for lemmatization and stemming [68]. Stemming and lemmatization will be discussed later in the report. We can understand the stemming and lemmatization as the cleaning steps for text using the Natural Language Processing (NLP). Tokenization is described as splitting the text in to tokens. Tokens can be defined as words in the sentence or sentences in a paragraph.

In this research, we have done tokenization on sentence level by using a function that uses "PunktSentenceTokenizer" from the nltk.tokenize.punkt module. This tokenizer is trained

already and so it knows very well how to mark the end and beginning of sentence considering the characters and punctuation.

3.1.3. Stemming

The process of stemming can be defined as a kind of normalization process performed for words [50]. Normalization is described as a technique in which we can convert the set of words in to sequence so that we can shorten its lookup time. Those words that have same meaning and have some variation due to context can be normalized. Simply we can define the stemming as a root word for many variants of that same word. For example, “kick” is a root word for its different variants like “kicks, kicking, kicked” and many others. With the help of same process, we can use stemming to find the root word for different variations of a word.

For example, we can find root word in the two sentences given below:

“He will be eating a burger.”

“He is going to eat a burger.”

In both sentences, we can see that meaning is same because action is going to be happened in the near future. We as humans can easily understands that both sentences have same meaning but machine will take both sentences as different. So it will hard for a machine to convert it into the same data. If we did not provide the same data to machine, it will be not possible for machine to interpret that these sentences have same meaning. And machine will be fail to predict this similarity in the meaning. So here is the point when we will use the process of stemming to categorize the same data in single category by finding the root for these same meaning words.

From the aforementioned explanation, we can conclude that stemming is one of most important step in the preprocessing of textual data before the starting the data processing. English language contains the different variations for a single word. That can cause ambiguity and redundancy. Process of stemming can remove this redundancy and ambiguity. And as a result we can use the data for much accurate training or processing.

3.1.4. Lemmatization

Process of Lemmatization is to find the lemma of a word according to its meaning. This process is usually refers to analyzing the words in morphological way and it can remove the inflectional endings [69]. It helps to return the base or dictionary form of word, usually known as lemma. Before staring the processing of data, stemming is performed and lemmatization as well.

Sometimes these two processes, stemming and lemmatization is considered as same and it can be confusing for some. But stemming and lemmatization have diffenrces. Lemmatization is usually preferred over stemming due to a reason mentioned below.

Stemming works by returning the root by cutting the suffix of words. But Lemmatization on the contrary performs more detailed operation. In addition to cutting the words to their roots it considered the morphological analysis of the words. Lemmatization gives the lemma which is the base form of all its inflectional forms. Lemmatization use the in-depth linguistic knowledge to perform the creation of dictionaries and returns the proper form of the word. Simply we can stemming is a general pre-processing step and lemmatization is an intelligent pre-processing step. Lemmatization can also save memory by forming better machine learning features and reducing the density of a word by returning the words to their base. It reduces the text ambiguity and gives clean representation of data. Cleaner the data, the more intelligent and accurate your machine learning model and will reduce the computational cost.

3.1.5. Stop Words

Process of converting the data to a form that a computer can understand is known as a preprocessing stage. This stage has many steps like stemming and lemmatization as we discussed before. Another major issue with raw data is useless data that we don't need for the processing or training. To resolve this issue, we performed a process of removing the useless data. This useless data is known as Stop words in natural Language Processing [6]. Stop words are the commonly used words like "the", "a", "an", "in" and we don't need these words during the processing or training because these words are not going to help in building the training model and will cost useless processing/computing power. Processing time and memory is very valuable in case of language processing, so we cannot let this useless data to increase the processing time and taking up extra memory. Some examples have been given in Table.5 below:

Table 05. Stop words explained

Text with Stop Words	Stop Words Removed
Listening can be exhausting?	Listening, Exhausting
I am reading and I like it	Reading, Like

3.1.6. Clustering

Clustering is an interesting process of Machine learning and it is used to group the data points. If we are given a set of different data points, clustering can be used to classify these data points in a specific number of groups known as clusters [54]. The data points of a single cluster should have similar features or properties and data points of different clusters will have dissimilarities in features or properties. Clustering is basically an unsupervised learning process because in this case ground truth is not available to compare the output of the results of algorithm to the true labels, to evaluate the algorithm's performance .This process is very common and widely used technique for the statistical analysis of data in many fields. Clustering is used in the field of data science to get some valuable insights from the data. We can define clustering as:

"Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data."

3.1.7. RNN

Recurrent Neural Network also referred as RNN are a type of neural network in which output of the previous step is fed as input to the next or current step. As we know that the basic or traditional neural networks work differently because in this case inputs and outputs doesn't depend on each other. But when we have to predict the next word of a sentence or next sentence of a paragraph, the previous state (sentence or word) has to be remembered. So Recurrent Neural Networks (RNN) came into existence and it proposed the solution for this problem by introducing the hidden layer. So most important concept related to the Recurrent Neural Network (RNN) is the hidden state, which has capability to remember the information about previous state in a sequence [58].

Recurrent Neural Network has ability to memorize and it has mmemeory that can remember all the previous information about any calculations. In these neural networks, same paramenters are used for each inout like it performs the similar task on all hidden layers or inputs to get the output. This helps to reduce the complexity of the parameters. We can understand the working principal of RNN by this exampe (Figure. 7).

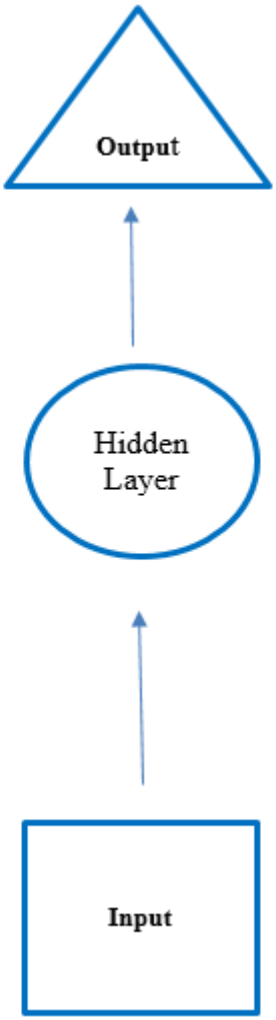


Figure 7. RNN Working Principal

Suppose ,we have a deep network and it has one inout, one output layer and three hidden layers. Now we know that like other neural networks, these hidden layers will have their own weights and biases. So suppose that layer 1, layer 2, layer 3 has weights and biases (w1,b1), (w2,b2) and (w3,b3) respectively as shown in Figure. 8. This means that these layers donot memorize the prevous outputs and are independent of each other.

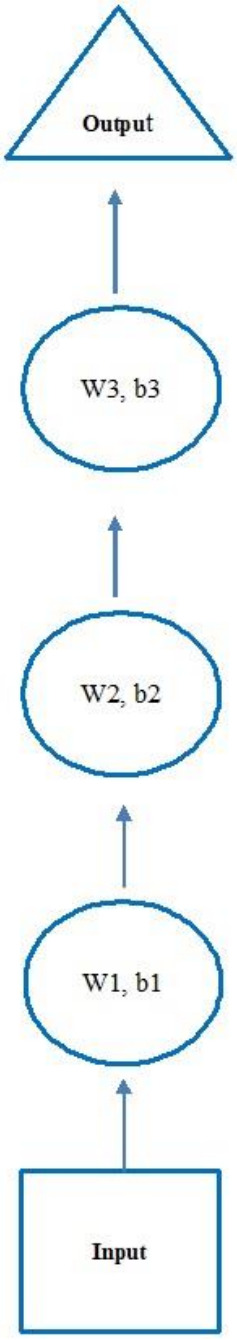


Figure 8. RNN Weights and Biases

Now recurrent neural Network have different strategy and it will convert these independent activations in to dependent ones by inserting the same biases and weights to all the layers. This will do the two things; first it will decrease the complexity also and second it will make possible to memorize the previous output/state by providing the output as an input to the next hidden

layer as shown in the Figure. 9. So these three hidden layers will be joined in to a single recurrent layer as all weights and bias of the layers will be same.

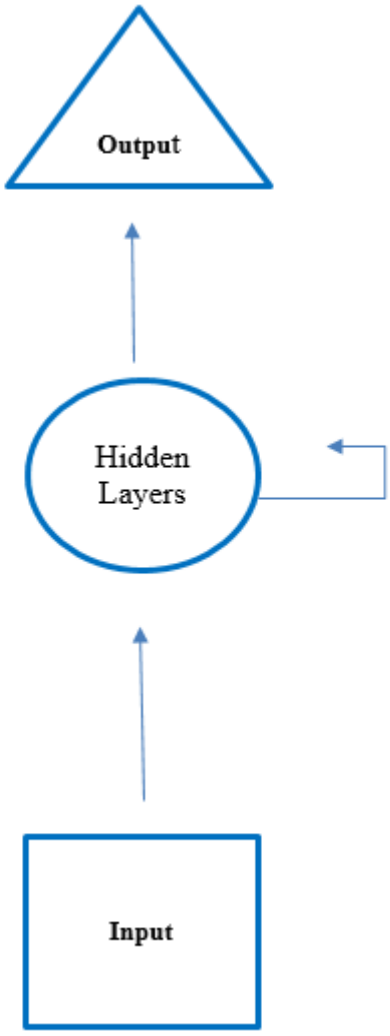


Figure 9. RNN Feedback Nature

Here is the formula for the calculation of Current state:

$$h_t = f(h_{t-1}, x_t)$$

Where

h_t represents the current state

h_{t-1} represents the previous state

x_t represents the input state

And the Formula for output calculation:

$$y_t = W_{hy} h_t$$

Y_t represents the output

W_{hy} represents the weight at output layer

Formula for applying Activation Function (tanh):

$$\mathbf{h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)}$$

W_{hh} represents the weight at recurrent neuron

W_{xh} represents the weight at input neuron

3.1.8. GRU

Vanishing gradient problem in machine learning is basically a difficulty in training the neural networks with the backpropagation and gradient-based learning methods. These methods, weights of neural network will be receiving an updated proportion to the partial derivative of the error function with respect to the current weight with the each iteration of the training. But issue is in some cases is that, gradient can be vanish-ably small and stopping the weights from changing its value. Or it can even stop the neural networks from further training in worst case. This problem of vanishing-exploding gradient problem is usually faced during the training of a basic Recurrent Neural Network. So to resolve this problem, many variations were developed. One of these variations is Long Short Term Memory Network (LSTM) [59] and another is Gated Recurrent Unit Network (GRU) [60]. Both these variations used for solution of vanishing-exploding gradient problem are equally effective. Difference between LTSM and GRU is that GRU has only three gates and doesn't no have an internal cell state unlike LTSM. The information is incorporated in a hidden state in case of GRU which is stored in the internal cell state in case of LTSM. And this information will be feed in to next GRU. Different gates of GRU [60] has been described below:

Update Gate (z): This decides that how much knowledge needs to be passed along in next stage from past stage. Update Gate is shown below:

Reset Gate (r): It will decide that how much information to forget from past. Reset Gate is shown bel

Current Memory Gate (h_t): It is often incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future. Current memory gate is shown below:

GRU's basic work flow is similar to the basic RNN, main difference is internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.

3.2. Proposed Solution

In our proposed methodology the similarity between sentences is calculated by using sentence embedding. After preprocessing, the sentences are converted to vectors of real numbers by sentence embedding. K-mean cluster is used to cluster these sentences. Then we have to select one sentence per cluster. Sentence ranking is used to rank sentences per information they contain and select high rank sentences for summarization. Python is used to develop the system. Figure.10 shows the top level flow of proposed system.

3.2.1 Preprocessing

NTK [63] is used to work on natural language. NLTK is a platform used to work on human language in python. It has many libraries for classification, tokenization, stemming etc. It also contains many corpora, almost over 50 like WordNet etc. Another beautiful thing about it is NLTK has a large community support. Natural Language Toolkit is a set of libraries written python for the statistical and symbolic natural language processing (NLP) for English language and it is usually known as NLTK. NLTK contains sample data and graphical demonstrations. It is intended to support the different research fields in Natural language processing (NLP) and other fields related to the machine learning, cognitive science, Artificial intelligence, Information retrieval and empirical linguistics. It has been successfully used as an individual study tool, and as a platform for prototyping and building research systems. NLTK can perform classification, tokenization, and stemming, tagging, parsing, and semantic reasoning functionalities.

First step is to combine all sentences from different documents like text file or .csv file. Tokenize the string data to list sentences is the next process to move to next step.

3.2.2 Sentence Embedding

Sentences are then converted to fixed length dense vectors. One approach is to use word2vec method to generate vector representation for every word in sentences. Then we can take average sum of these vectors to calculate sentence embedding. Another approach is to use sentence vector. Skip-thought [64] is used to calculate sentence embedding. It is works as encoder-decoder model. Skip-thought used RNN [58] encoder with GRU [60] activations and an RNN decoder with a conditional GRU.

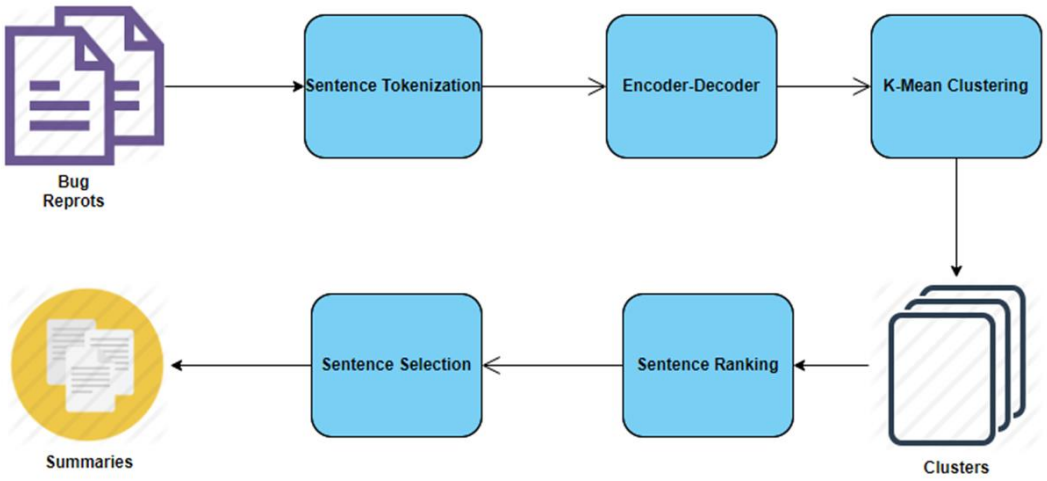


Figure 10. Methodology Flow Diagram

We implemented and considered the skip-thoughts as framework of encoder-decoder model. Encoder-Decoder model is being widely used in the field of neural machine translation. An encoder will map the words of a sentence to the sentence vector. Then a decoder will be used to generate the surroundings of that sentence. We have used the encoder with the RNN with GRU activations and decoder with the conditional GRU. GRU has shown equally good performance like LTSM and it is simpler.

```
('encoded is:', array([[ 0.00189015, -0.00062589,  0.03172658, ..., -0.00820192,
         0.0023868 ,  0.00570701],
       [ 0.00658135, -0.01066917, -0.01804201, ...,  0.03118046,
         0.00139447,  0.01604929],
       [-0.00747948, -0.01266284, -0.01512253, ..., -0.01106962,
         0.00288723, -0.00303695],
       ...,
       [ 0.00576437, -0.01023412,  0.0003492 , ..., -0.00305528,
         0.00268128, -0.00043554],
       [-0.01420968, -0.00232877,  0.00229206, ...,  0.04353614,
         0.00247602,  0.00019447],
       [ 0.00795791, -0.00086122,  0.01546293, ..., -0.01676783,
        -0.00593759,  0.00178594]], dtype=float32))
```

Figure 11. Vector representation of sentences

The encoder encodes the sentence into vector. Given a sentence, the decoder generates the surrounding sentences of that input. Consider we have 3 sentences S_{i-1}, S_i, S_{i+1}. If we give S_i to decoder, the output of decoder will be S_{i-1}, S_{i+1}. Pre trained model are available for English sentences. As focus of this paper is also on English language so there is no need to train the model. These vectors of sentences are then passed to next step. Figure. 11 shows the vectored sentences.

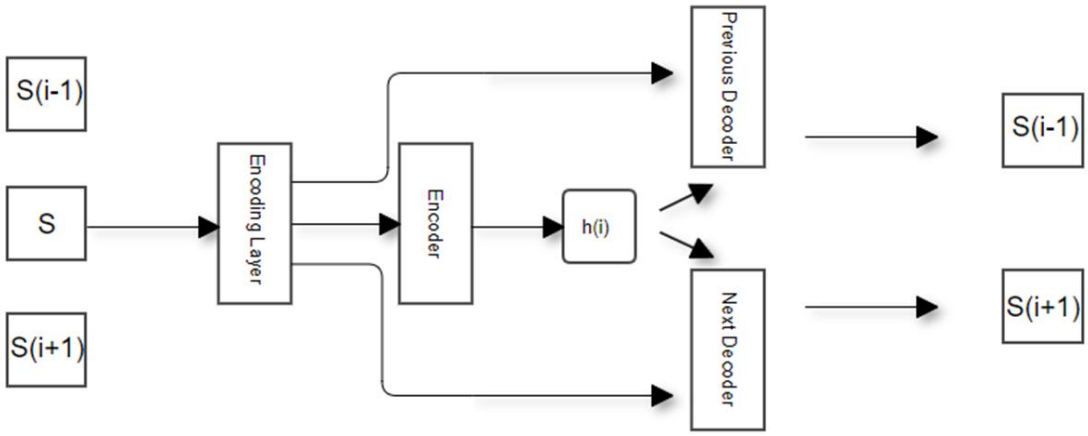


Figure 12. Encoder-Decoder Model

3.2.3 Clustering based on Similarity

As we have vectors of sentences. These vectors contain a list of features (numbers). By comparing these features we can calculate clusters of similar sentences. K mean clusters are used for this purpose. K-mean cluster fit best in our scenario as we have numbers and fixed

number of clusters [57]. This number depends upon the size of input document (number of sentences). Clusters are different for different size of data. Figure. 13 show the clusters of sentences.

```
{0: [9], 1: [13, 21, 22], 2: [4, 11, 12, 14, 24, 25, 27, 28, 36], 3: [0, 5, 6, 20,
35, 39], 4: [2, 16, 38, 40, 43], 5: [26, 30, 42], 6: [7, 8, 10, 29], 7: [41, 48],
8: [1, 3, 15, 37], 9: [17, 18], 10: [34, 44, 46], 11: [23, 45, 47], 12: [33], 13:
[19, 32], 14: [31]}
```

Figure 13. Clusters of sentences

Clusters are in following format:

```
{cluster0: sentences, cluster1: sentences}
```

The sentences are array having 1 or many sentences. Each sentence is represented by a number. This number is the order number of that specific sentence in document.

For K-mean clustering, first of all we will select the number of clusters or groups to be used and the will initialize the randomly center points of these clusters [56]. We should take a quick look, in order to figure out the classes' number to use and try to identify the different distinct classes. K-mean will categorize the data in to specified groups of similarity. And Euclidian distance is used to measure the similarity between data points. Basic flow of this algorithm is mentioned below:

In first step, center points or means will be initialized randomly.

Then each data point will be categorized to its closest center point or mean and then these means will be updated according to the average of the points that are grouped in previous iteration.

This process will be repeated for a specified number of iterations and then we will have our clusters.

The center points are called means because they are the average of the points categorized in that cluster. These can be initialized randomly in an intuitive way.

The above algorithm in pseudocode:

"Initialize k means with random values

For a given number of iterations:

Iterate through items:

Find the mean closest to the item

Assign item to mean

Update mean ”

3.2.4 Sentence Ranking

Sentences are ranked based on the information they carry. Tf-idf is most commonly used term weighting method [67]. This assign a high weight to a term or word if it is occurring frequently in the document but rarely occurred in the whole document collection. On the other hand, if one term or word is occurring in nearly all the documents of a collection then it will have less discriminatory power and it will be given a low weight. Usually these low weight term are the stop words in a document. So in order to calculate the tf-idf , we need to know the two things; first is that how many times a word occur in the document which is known as term frequency and second one is that in how many documents this term appears known as document frequency. Then we will take the inverse of the document frequency and we have both components to calculate the weight by multiplying tf by idf.

In practice, the inverse document frequency is calculated as the logarithm (base 10) of the quotient of the total number of documents (N) and the document frequency in order to scale the values.

$$idf = \log\left(\frac{N}{df}\right)$$

Selecting one by one cluster and looping through it. Every sentence within a cluster is process to remove stop words and stemmed. Finally find the weight of every sentence by the following formula

$$Sw = \sum_{i}^{n} tfidf(wi)$$

Where

‘Sw’ is the weight of sentence

‘n’ is the total number of words in Sw

‘wi’ is the i_{th} word of sentence

The same method repeated for every cluster. Now we have all sentences along with their weight.

3.2.5 Sentence Selection

From every cluster only one sentence is selected having high weight from other sentences of same cluster. By getting only one sentence per cluster solve the problem of redundancy. The sentences are then re arranged in their original sequence.

Chapter 4

Implementation

CHAPTER 4: IMPLEMENTATION

The chapter elaborates the implementation details of our proposed methodology for the automated summarization of bug reports. **Section 4.1** discusses the details about the standard data-set we have used for our research. **Section 4.2** gives details about the experimentation details.

4.1. Data-Set

We implemented proposed algorithm with the machine learning library Theano. We implemented our methodology on a standard data set known as Summary Data Set (SDS). This data set contains 36 bug reports and this bug report corpus was created by Sarah Rastkar et al [13] and has been used widely in this research field. It contains bug reports from four open-source Software projects which includes Eclipse Platform, Mozilla, Genome and KDE. These reports have different length in terms of sentences or comments. Sixty nine percent reports have five to fourteen comments and remaining thirty one percent have fifteen to twenty five comments.

Table 6. Data-Set Details

Data-Set	No. of Bug Reports	Average Comment Count	Average Comments Size	Total Sentences
SDS	36	6.5	9.5	2361

Total number of sentences are 2361 in these reports. Table.6 shows the details about SDS data set. Each bug report in this data set was annotated by three annotators and they wrote an abstractive summary for each report. These abstractive summaries known as Gold standard summaries (GSS) are used for evaluation of our results. Generated summaries are 25% of the original bug report.

BUG REPORT

Comment 1:

The releng test have been DNF for the last couple builds. Running locally we occasionally see crashes in the BrowserTests so we suspect these crashes to be the cause of the DNF.

Comment 2:

Created an attachment (id=135853) [details]

Comment 3:

Pushing this fix for RC2, as it's too risky to make this change for RC1 now. For RC1 we'll comment out the crashing tests so that the other tests can run properly. We have NOT seen this bug affect the usability of Eclipse, only the running of the SWT test suite.

Comment 4:

The problem is that the strategy we use to release pools periodically (in readAndDispatch, createWidget, etc.) to be able to run the test suites, releases the main thread pool created be Device and stored in a thread local dictionary. The pool was not removed from the thread local dictionary and got reused the next time a display was created. The new strategy is to only release the main thread pool when the device count goes to zero.

Comment 5:

Obviously, something this low level needs as much testing as it can get. Can we get a test build run to verify all tests pass/no crashes/no OOM/etc.? Also, you should be doing the two day test pass on a version of SWT that includes the change.

Comment 6:

Created an attachment (id=136083) [details] new fix

Comment 7:

The new fix is simpler, we just make sure the pool in the thread dictionary is always valid, that way when another display is created it will not use a released pool. There was a further problem where the pool was released too early. If we are in call in, we cannot released the top of the stack pool.

Comment 8:

The tests that where showing this problem have been put back (just in HEAD). Fixed > 20090515

Figure 14. Bug Report Structure

4.2. Experimentation

We implemented proposed algorithm with the machine learning library Theano. Theano is a library written in Python and it is used for fast numerical computations that can be

run on the CPU or GPU. Theano can be used directly to create Deep Learning models or wrapper libraries to simplify the process greatly and it is a key foundational library for Deep Learning in Python. We have run our algorithm on corei5 with RAM of 8 GB and GPU of 2 GB.

First of all we took the data from text files and then performed the proposed algorithm. We used following libraries for or algorithm implementation: NLTK, Skipthought, Numpy, Sklearn for clustering, SentenceRanking for tf-idf, RankSentence to select top sentences. NLTK has been used for the tokenization of the bug report at sentences level. Then these tokenized sentences will be given as input to the skipthought module, which will use the encoder-decoder method to convert he sentences in to vectors. Then these vectors will be used for clustering. Cluster's sentences will be then ranked by using tf-idf weighting technique. Then we will select the top sentences from each clusters. It will generate the summary.

We generated the summaries for 36 bug reports. An example is shown below:

System Summary

The releng test have been DNF for the last couple builds. Running locally we occasionally see crashes in the BrowserTests so we suspect these crashes to be the cause of the DNF.

For RC1 we'll comment out the crashing tests so that the other tests can run properly.

We have NOT seen this bug affect the usability of Eclipse, only the running of the SWT test suite.

The problem is that the strategy we use to release pools periodically (in readAndDispatch, createWidget, etc.) to be able to run the test suites, releases the main thread pool created be Device and stored in a thread local dictionary.

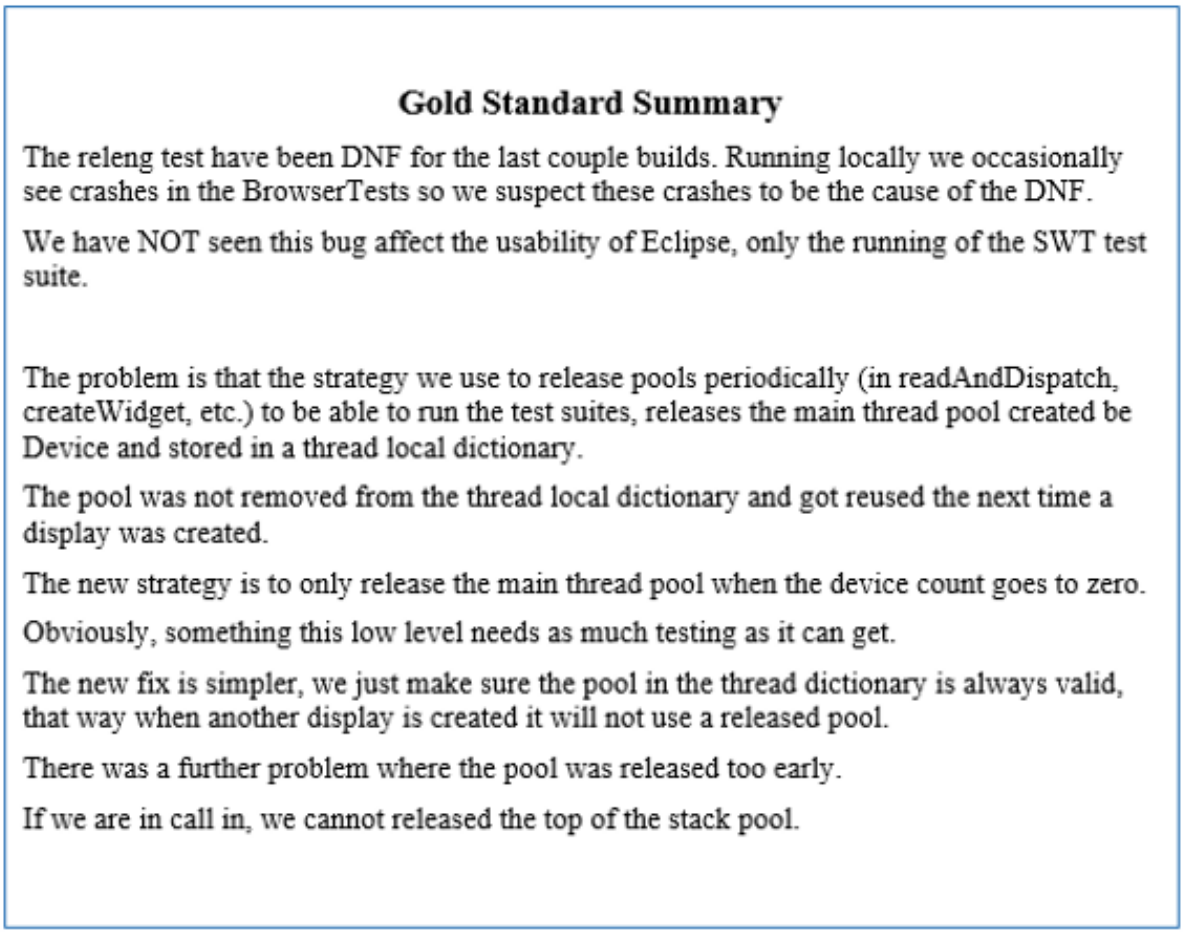
The pool was not removed from the thread local dictionary and got reused the next time a display was created.

Fixed > 20090515

Figure 15. System Generated Summary Sample

These system generated summaries were used to compare with the gold standard summaries provided with the original data-set. These gold standard summaries were annotated by the experienced annotators [14]. First they wrote abstractive summaries for each report and then

they linked each sentence of the abstractive summary with the sentences of bug reports. Gold standard summaries are consisted of these linked sentences of the original bug reports. Now we have three gold summaries of each bug report. So to achieve consensus, we take these sentences in to our final gold standard summary which are present in at least two annotated summaries. Below is an example of the gold standard summary:



Gold Standard Summary

The releng test have been DNF for the last couple builds. Running locally we occasionally see crashes in the BrowserTests so we suspect these crashes to be the cause of the DNF.

We have NOT seen this bug affect the usability of Eclipse, only the running of the SWT test suite.

The problem is that the strategy we use to release pools periodically (in readAndDispatch, createWidget, etc.) to be able to run the test suites, releases the main thread pool created be Device and stored in a thread local dictionary.

The pool was not removed from the thread local dictionary and got reused the next time a display was created.

The new strategy is to only release the main thread pool when the device count goes to zero.

Obviously, something this low level needs as much testing as it can get.

The new fix is simpler, we just make sure the pool in the thread dictionary is always valid, that way when another display is created it will not use a released pool.

There was a further problem where the pool was released too early.

If we are in call in, we cannot released the top of the stack pool.

Figure 16. Gold Standard Summary Sample

Main coding layout for implementation is given below:

“from nltk.tokenize import sent_tokenize

import skipthoughts

import numpy as np

from sklearn.cluster import KMeans

import collections

```
from sentenceRanking import tfIdf
from rankSentence import rankPreprocessor
model = skipthoughts.load_model()
encoder = skipthoughts.Encoder(model)
def task (fileName):
    BugReport_file = open('BUGREPORTs/'+fileName,'r')
    BugRep =BugReport_file.read().decode("UTF-8")
    BugRep =BugRep.replace('\n','')
    summaryFile = open('summaries/system/'+fileName,'w+')
    sentences = sent_tokenize(BugRep)
    encoded =  encoder.encode(sentences)
    print('encoded is:',encoded)
    n_clusters = np.int64(np.ceil(len(sentences)**0.30))
    if len(sentences)<100:
        n_clusters  = np.int64(np.ceil(len(sentences)*0.25))
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans = kmeans.fit(encoded)
    clusters = collections.defaultdict(list)

    for i, label in enumerate(kmeans.labels_):
        clusters[label].append(i)
    clusters = dict(clusters)
    print(clusters)
    #tf idf matrix function  is
    tfidf_matrix = tfIdf(BugRep)
    summary_indexes =  rankPreprocessor(sentences,tfidf_matrix,clusters)
    summary = '\n'.join(' '+sentences[id] for id in summary_indexes).encode('utf-8').strip()
```

```
    summaryFile.write(summary)
for i in range(3):
    if i==0:
        continue
    fileName = '00'+str(i)+'.txt'
    if i>9:
        fileName = '0'+str(i)+'.txt'
    task(fileName)”
```

Chapter 5

Results and Evaluation

CHAPTER 5: Results and Evaluation

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

5.1. Evaluation Metric

Different evaluation metrics have been used for the evaluation of the summary generating algorithms in the past [66]. But most commonly used metric used for the evaluation of summaries is ROUGE. It is widely accepted because it is an automated evaluation method.

5.1.1. ROUGE

ROUGE is a combination of different metrics or variations of a metric, used for the evaluation of automatic text summarization and it stands for "Recall oriented Understudy for Gisting Evaluation" [65]. It compares the automatically generated summary with the reference summary usually known as gold standard summary. For better understanding for example we have a system summary or a summary produced by machine given below:

"The cat was running fast"

And below is given reference Summary or gold standard summary:

"The cat was running very fast"

So if we count the words in system summary ,it gives a count of 5 words and reference summary or gold standard summary gives a word count of 6 words. So rouge will find the overlapping between both system and reference summary. But it doesn't give enough idea about the quality. So Rouge recall ad precision is measured using the overlap for each summary. So Rouge recall would be the how much coverage of reference summary is done in the system summary. If we are just considering the Rouge-1 then recall would be:

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

In case of above given example Recall rouge would be:

Recall = 5/6 = 0.83

This means 83% of reference summary has been captured by the system summary.But only recall is not enough and it doesn't tell us about the other side of the story. It doesn't give the idea about how much right information is the system summary contains. So we will find the precision, which can be defined as:

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

In above mentioned example, precision would be:

Precision = 5/5 =1

That means system summary contains 100% right information and doesn't consist of useless information .Although in practice this doesn't happens. But reality is that recall and precision don't give the complete idea more than often. So F-measure is always used for the measurement of ROUGE.ROUGE has different variations like ROUGE-N, ROUGE-S, and ROUGE-L and difference is their granularity level of texts being compared between the system summaries and reference summaries.

- ROUGE-N: "it measures unigram, bigram, trigram and higher order n-gram overlap like ROUGE-1, ROUGE-2 or ROUGE-3 etc."
- ROUGE-L: "it measures longest matching sequence of words"
- ROUGE-S: "it is also known as skip-gram concurrence and it measures the overlap of word pairs that can have a maximum of two gaps in between words."

We used a rouge evaluation package 1.5.5 [62] and below we have shown an example of output in the figure.

We used R-1 and R-2 and measured the F-measure as it gives the all over idea about the performance.

Rouge evaluation package 1.5.5 is used for the rouge evaluation:

==========

"from pyrouge import Rouge155

r = Rouge155()

r.system_dir = 'summaries/system'

r.model_dir = 'summaries/user'

r.system_filename_pattern = '(\d+).txt'

r.model_filename_pattern = '(\d+).txt'

output = r.convert_and_evaluate()

print(output)

output_dict = r.output_to_dict(output)”

==========

```
---------------------------------------------
1 ROUGE-1 Average_R: 0.60000 (95%-conf.int. 0.60000 - 0.60000)
1 ROUGE-1 Average_P: 0.46584 (95%-conf.int. 0.46584 - 0.46584)
1 ROUGE-1 Average_F: 0.52448 (95%-conf.int. 0.52448 - 0.52448)
---------------------------------------------
1 ROUGE-2 Average_R: 0.33065 (95%-conf.int. 0.33065 - 0.33065)
1 ROUGE-2 Average_P: 0.25625 (95%-conf.int. 0.25625 - 0.25625)
1 ROUGE-2 Average_F: 0.28873 (95%-conf.int. 0.28873 - 0.28873)
---------------------------------------------
1 ROUGE-3 Average_R: 0.27642 (95%-conf.int. 0.27642 - 0.27642)
1 ROUGE-3 Average_P: 0.21384 (95%-conf.int. 0.21384 - 0.21384)
1 ROUGE-3 Average_F: 0.24114 (95%-conf.int. 0.24114 - 0.24114)
---------------------------------------------
1 ROUGE-4 Average_R: 0.22951 (95%-conf.int. 0.22951 - 0.22951)
1 ROUGE-4 Average_P: 0.17722 (95%-conf.int. 0.17722 - 0.17722)
1 ROUGE-4 Average_F: 0.20000 (95%-conf.int. 0.20000 - 0.20000)
---------------------------------------------
1 ROUGE-L Average_R: 0.60000 (95%-conf.int. 0.60000 - 0.60000)
1 ROUGE-L Average_P: 0.46584 (95%-conf.int. 0.46584 - 0.46584)
1 ROUGE-L Average_F: 0.52448 (95%-conf.int. 0.52448 - 0.52448)
---------------------------------------------
1 ROUGE-W-1.2 Average_R: 0.23280 (95%-conf.int. 0.23280 - 0.23280)
1 ROUGE-W-1.2 Average_P: 0.29610 (95%-conf.int. 0.29610 - 0.29610)
1 ROUGE-W-1.2 Average_F: 0.26066 (95%-conf.int. 0.26066 - 0.26066)
---------------------------------------------
1 ROUGE-S* Average_R: 0.31768 (95%-conf.int. 0.31768 - 0.31768)
1 ROUGE-S* Average_P: 0.19115 (95%-conf.int. 0.19115 - 0.19115)
1 ROUGE-S* Average_F: 0.23868 (95%-conf.int. 0.23868 - 0.23868)
---------------------------------------------
1 ROUGE-SU* Average_R: 0.32207 (95%-conf.int. 0.32207 - 0.32207)
1 ROUGE-SU* Average_P: 0.19448 (95%-conf.int. 0.19448 - 0.19448)
1 ROUGE-SU* Average_F: 0.24252 (95%-conf.int. 0.24252 - 0.24252)
```

Figure 17. ROUGE Evaluation Package output

5.2. Results Evaluation and comparison:

After the implementation of our algorithm, we evaluated the experimentation by using widely used and accepted summarization evaluation package "Recall-Oriented Understudy for Gisting Evaluation" or ROUGE. This metric compare the automatically generated summary with the human written reference summary or Gold summary [65]. ROUGE tries to asses that if the automatically generated summary covers the most of important information present in the reference summary or Gold summary. ROUGE-1 and ROUGE-2 are very useful in emulation of human evaluation procedure [66].

Table 7. Results

Report Number	ROUGE-1	ROUGE-2
1	0.39	0.14
2	0.75	0.13
3	0.57	0.28
4	0.75	0.25
5	0.64	0.19
6	0.75	0.22
7	0.67	0.27
8	0.72	0.21
9	0.71	0.29
10	0.52	0.13
11	0.41	0.18
12	0.57	0.23
13	0.62	0.25
14	0.69	0.22
15	0.41	0.17
16	0.4	0.17
17	0.58	0.19
18	0.52	0.26
19	0.51	0.22
20	0.69	0.28
21	0.57	0.26
22	0.32	0.16
23	0.71	0.21
24	0.54	0.23
25	0.52	0.18
26	0.58	0.19
27	0.56	0.29
28	0.68	0.32
29	0.71	0.25

30	0.5	0.19
31	0.61	0.23
32	0.62	0.21
33	0.52	0.16
34	0.5	0.21
35	0.54	0.16
36	0.7	0.26
Average	**0.584722222**	**0.216388889**

The reason we didn't used ROUGE-1 alone but with the combination of ROUGE-2 due to the reason that it can tell us about the fluency of the system summaries or translation. As we know that if we follow the word orderings of the reference summary more closely, then our summary is actually more fluent.

Table 8. Results & Comparison

Algorithms	**R-1**	**R-2**
BRC [13] [14]	0.521	0.140
Centroid [15] [22]	0.471	0.126
MMR [15] [22]	0.498	0.145
Grasshopper [15] [22]	0.505	0.135
Diverse Rank [15] [22]	0.500	0.139
PageRank [17]	0.525	0.153
Deep Sum [18]	0.563	0.177
Sentence Embedding	0.584	0.216

That's why, we calculated the ROUGE-1 and ROUGE-2 for the evaluation and comparison of our summaries with previous work in this field, by using Rouge-1.5.5 [62]. We calculated the ROUGE-1 and ROUGE-2 for the 36 reports and then we took average of these results to achieve the final reading for our algorithm as shown in Fig.18.

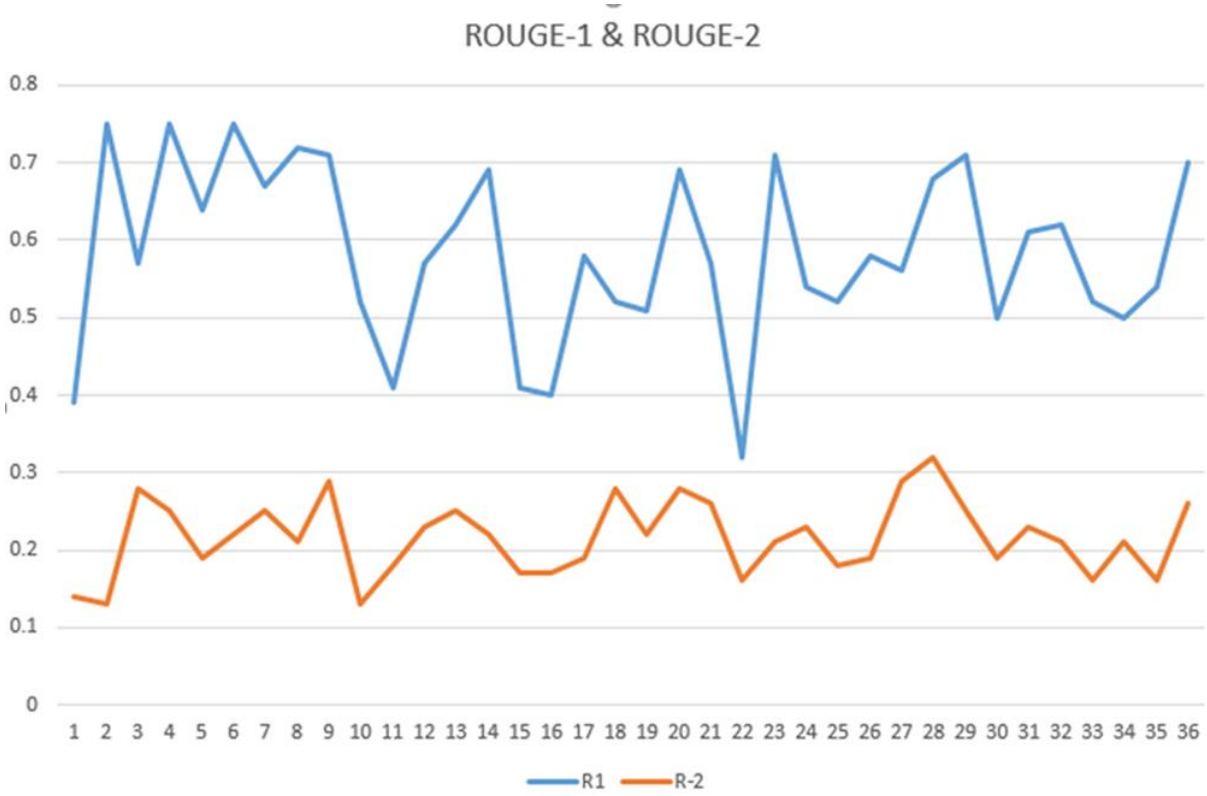


Figure 18. Graphical Representation of individual results

Then we compared our result with the previous algorithms used for the automated bug report summarization. Table.7 shows the results of this experiment. Results shows that our algorithm has shown significant improvement in rouge-1 and rouge-2 metrics on SDS Data set [13]. Fig.20 shows the graphical representation of comparison of our algorithm with algorithms used previously for bug report summarization

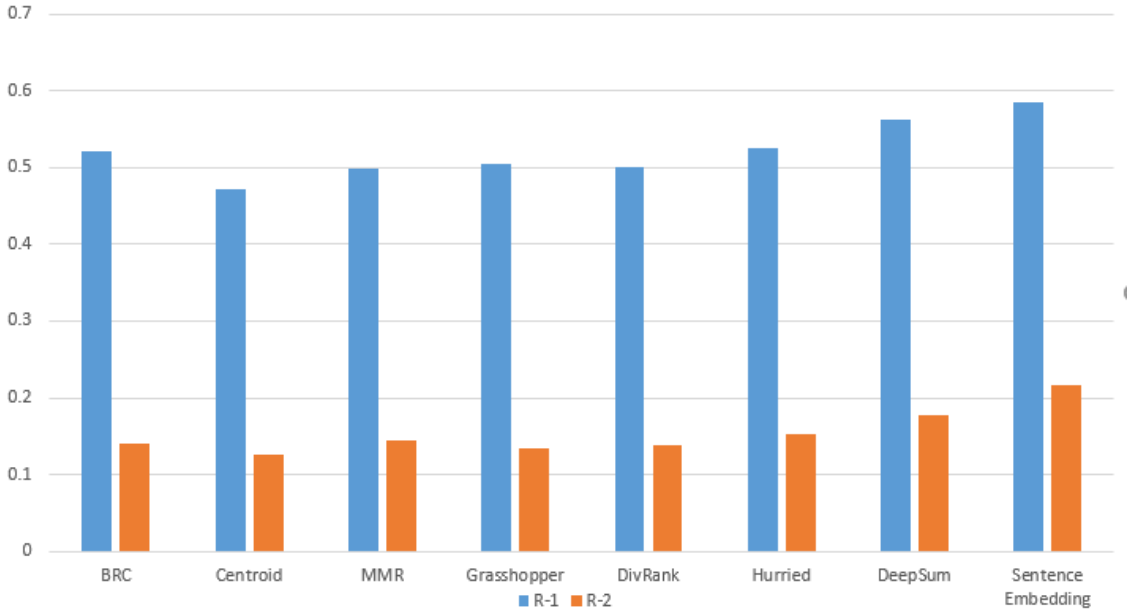


Figure 19. Graphical representation of Results

Chapter 6

Discussion and Limitation

CHAPTER 6: DISCUSSION AND LIMITATION

This chapter deals with an overview of research by highlighting research gap covered in **Section 6.1** whereas Limitations to research are mentioned in section **6.2**

6.1. Discussion

Automation has revolutionized the modern day processes by reducing the time delays and introduced the great accuracy. Natural language processing has been proven great beneficial in software development processes and has helped the software developer with in many fields. One such field is the summarization of software artifacts to save the time of developers needed to analyze these artifacts. We have optimized this process by using sentence embedding through skip thought vectors. We compared our results with the previous bug report summary generation algorithms and achieved improved results in term of rouge- and rouge-2 metric.

6.2. Limitations

As this approach is an extractive unsupervised learning in the field of automated bug report summarization and still it needs the improvement in the results. Also this approach doesn't give the abstractive summaries and abstractive summaries are more helpful if generated rightly. Also his approach doesn't consider the importance of a person who is commenting in the conversation. Like more experienced person's comment should have given more weight to include it in the final summary.

Chapter 7

Conclusion and Future Work

CHAPTER 7: CONCLUSION AND FUTURE WORK

This chapter deals with an overview of research conclusion covered in **Section 7.1** whereas future work is mentioned in **Section 7.2.**

7.1. Conclusion

Software Automation field is progressing day by day. Software development demand is increasing on daily basis as the modern age is moving towards the automation of all daily life process. So this demands the fast and optimized software development process. One of the biggest hurdle in the fast delivery of the software is the number of bugs faced during the development process. In case of large projects, number of issues arise in the development process are enormous. So if a team has to resolve the same bug each time with same effort, it would be time wasting and money as well. So to help the developers and to make the bug resolution fast. Developers can take advantage of historical knowledge about the bug if it has been faced before and in no time they can resolve the bug with the previously used resolution methods. For this purpose bug repositories are used. But as bug repositories would be growing with the time and it is difficult for the developers to find the needed bug report. It would be a very time consuming task to read all of the bug reports to find their concerning report. For the solution of this problem, automation of bug report summarization was introduced in 2010 first time. Many researchers have worked on this process and achieved good results. But there was still need of improvements in this process. So we proposed a novel framework for the automated summarization of bug reports to speed-up the software development process to cope with the modern days demands. We proposed an extractive based supervised approach for this purpose and achieved improvements in the ROUGE-1 and ROUGGE-2 results as compared to previously used techniques. But still there is a need of improvement in this area. We have discussed about suggestions in the next part about what can be improved and possible future directions.

7.2. Future Work

Future work includes improving and extending this approach in order to support the bug resolution in better way. We proposed an extractive summarization approach and it generates the extractive summaries for bug reports. An abstractive approach can be used in the future as abstractive summaries are more human alike. A proper GUI tool can be generated and

provided for the public access to use this tool. Also many important features can be identified in the bug reports like importance of persons involved in the conversation and use this to include most important sentences in the summary.

References

[1] J. D. Strate and P. A. Laplante, "A Literature Review of Research in Software Defect Reporting," in *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 444-454, June 2013. doi: 10.1109/TR.2013.2259204

[2] A. A. Rahman and N. Hasim, "Defect Management Life Cycle Process for Software Quality Improvement," 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS), Kota Kinabalu, 2015, pp. 241-244. doi:10.1109/AIMS.2015.47

[3] B. S. Rawal and A. K. Tsetse, "Analysis of bugs in Google security research project database," 2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS), Trivandrum, 2015, pp. 116-121. doi: 10.1109/RAICS.2015.7488399

[4] Yu, L., Ramaswamy, S., & Nair, A. (2011). Using bug reports as a software quality measure. ICIQ.

[5] A. Gelbukh, "Natural language processing," Fifth International Conference on Hybrid Intelligent Systems (HIS'05), Rio de Janeiro, Brazil, 2005, pp. 1 pp.-. doi: 10.1109/ICHIS.2005.79

[6] P. Kłosowski, "Deep Learning for Natural Language Processing and Language Modelling," 2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), Poznan, 2018, pp. 223-228. doi: 10.23919/SPA.2018.8563389

[7] Y. Tian and D. Lo, "A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, 2015, pp. 570-574. doi: 10.1109/SANER.2015.7081879

[8] Cheng Juan, "Research and implementation English Morphological Analysis and Part-of-Speech tagging," 2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT), Shenzhen, 2010, pp. 496-499. doi: 10.1109/EDT.2010.5496438

[9] Ankita and K. A. Abdul Nazeer, "Part-of-speech Tagging and Named Entity Recognition Using Improved Hidden Markov Model and Bloom Filter," 2018 International Conference on Computing, Power and Communication Technologies (GUCON), Greater Noida, Uttar Pradesh, India, 2018, pp. 1072-1077. doi: 10.1109/GUCON.2018.8674901

[10] Kitchenham, Barbara. "Procedures for performing systematic reviews." Keele, UK, Keele University33.2004 (2004): 1-26.

[11] D. Cubrani c and G.C. Murphy, “Hipikat: Recommending Pertinent Software Development Artifacts,” Proc. 25th Int’l Conf. Software Eng. (ICSE ’03), pp. 408-418, 2003.

[12] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, “Towards More Accurate Retrieval of Duplicate Bug Reports,” Proc. 26th Int’l Conf. Automated Software Eng. (ASE ’11), pp. 253-262, 2011.

[13] Sarah Rastkar , Gail C. Murphy, Gabriel Murray(2014), “Automatic Summarization of Bug Reports “ , IEEE Transactions on Software Engineering Volume 40 Issue 4, Pages 366-380.

[14] Sarah Rastkar , Gail C. Murphy, Gabriel Murray(2010), “Summarizing Software Artifacts:A Case Study of Bug Reports” ICSE '10 Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, Pages 505-514.

[15] Senthil Mani, Rose Catherine, Vibha Singhal Sinha, Avinava Dubey(2012), “AUSUM: approach for unsupervised bug report summarization” , FSE '12 Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, Article No. 11.

[16] Cheng-Zen YANG, Cheng-Min AO, Yu-Han CHUNG(2018), “Towards an Improvement of Bug Report Summarization Using Two-Layer Semantic Information”, IEICE Transactions on Information and Systems. VOL.E101–D, NO.7, Pages 1743-1750

[17] R. Lotufo, Z. Malik, and K. Czarnecki (2012) “Modeling the ‘Hurried’ Bug Report Reading Process to Summarize Bug Reports,” Proceedings; 28th IEEE International Conference on Software Maintenance (ICSM '12), pp.430-439.

[18] Xiaochen Li, He Jiang, Dong Liu, Zhilei Re, Ge Li (2018), “Unsupervised Deep Bug Report Summarization” ICPC '18 Proceedings of the 26th Conference on Program Comprehension, Pages 144-155.

[19] H. Jiang, N. Nazar, J. Zhang, T. Zhang, and Z. Ren(2017), “PRST: A PageRank-Based Summarization Technique for Summarizing Bug Reports with Duplicates,” International Journal of Software Engineering and Knowledge Engineering, Vol. 27, No. 06, Pages:869-896.

[20] Shamima Yeasmin, Chanchal K. Roy, Kevin A. Schneider (2014), “Interactive Visualization of Bug Reports using Topic Evolution and Extractive Summaries” ICSME

'14 Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Pages 421-425

[21] Isabella Ferreira, Elder Cirilo, Vinicius Vieira, Fernando Mour˜ao (2016), "Bug Report Summarization: An Evaluation of Ranking Techniques" X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pages: 101-110.

[22] He Jiang , Xiaochen Li , Zhilei Ren, Jifeng Xuan, Zhi Jin (2019), "Toward Better Summarizing Bug Reports With Crowdsourcing Elicited Attributes" IEEE Transactions on Reliability, Volume 68(1), pages: 2-22

[23] Nicolas Bettenburg, Rahul Premraj(2008) "Extracting Structural Information from Bug Reports" MSR '08 Proceedings of the 2008 international working conference on Mining software repositories, Pages 27-30

[24] Luca Ponzanelli, Andrea Mocci, Michele Lanza(2015), "Summarizing Complex Development Artifacts by Mining Heterogeneous Data"MSR '15 Proceedings of the 12th Working Conference on Mining Software Repositories, Pages 401-405.

[25] Gabriel Murray, Giuseppe Carenini (2008), "Summarizing Spoken and Written Conversations" EMNLP '08 Proceedings of the Conference on Empirical Methods in Natural Language Processing, Pages 773-782

[26] Yuan Tian, David Lo(2015), "A Comparative Study on the Effectiveness of Part-of-Speech Tagging Techniques on Bug Reports", IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 570-574.

[27] Kukkar A., Mohana R. (2019), "An Optimization Technique for Unsupervised Automatic Extractive Bug Report Summarization". International Conference on Innovative Computing and Communications, Pages:1-11.

[28] D. Bertram, A. Voida, S. Greenberg, and R. Walker,(2010) "Communication,Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams," Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '10), pp. 291-300, 2010.

[29] Dragomir R. Radev, Hongyan Jing, Malgorzata Stys, and Daniel Tam.(2004) "Centroid-based summarization of multiple documents." Information Processing and Management, 40:919–938, November 2004.

[30] Xiaojin Zhu, Andrew B. Goldberg, Jurgen Van, and Gael David Andrzejewski(2007). "Improving diversity in ranking using absorbing random walks." In Physics Laboratory â˘A ¸S University of Washington, pages 97–104, 2007.

[31] Qiaozhu Mei, Jian Guo, and Dragomir Radev. Divrank:(2010) “the interplay of prestige and diversity in information networks.” In Proc. 16th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD ’10, pages 1009–1018, 2010

[32] Nicolas Bettenburg, Rahul Premraj, Sunghun Kim, and Thomas Zimmermann. (2008). “Extracting structural information from bug reports”. In Proceedings of the International Working Conference on Mining Software Repositories (MSR’08). ACM, 27–30.

[33] G. Murray and G. Carenini,(2008) “Summarizing Spoken and Written Conversations,” Proc. 2008 Conference on Empirical Methods
in IEICE TRANS. INF. & SYST., VOL.E101–D, NO.7 JULY 2018 Natural Language Processing (EMNLP ’08), pp.773–
782, 2008.

[34] Lawrie, Dawn J. and Dave W. Binkley.(2018) “On the Value of Bug Reports for Retrieval-Based Bug Localization.” 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME) (2018): 524-528.

[35] Xiaoyin Wang, David Lo, Jing Jiang, Lu Zhang, Hong Mei (2009)“Extracting paraphrases of technical terms from noisy parallel software corpora” In Proceeding ACLShort '09 Proceedings of the ACL-IJCNLP 2009 Conference Short Papers Pages 197-200

[36] Per Runeson , Magnus Alexandersson, Oskar Nyholm (2007) “Detection of Duplicate Defect Reports Using Natural Language Processing” In Proceeding ICSE '07 Proceedings of the 29th international conference on Software Engineering. Pages 499-510.

[37] A. Nenkova and K. McKeown,(2011) “Automatic Summarization,” Foundations and Trends in Information Retrieval, vol. 5, no. 2/3, pp. 103-233, 2011.

[38] Xiaoyin Wang , Lu Zhang, John Anvik, Jiasu Sun (2008) “An approach to detecting duplicate bug reports using natural language and execution information” In Proceeding ICSE '08 Proceedings of the 30th international conference on Software engineering Pages 461-470.

[39] Yang, Cheng-Zen et al. (2012)“Duplication Detection for Software Bug Reports Based on BM25 Term Weighting.” 2012 Conference on Technologies and Applications of Artificial Intelligence (2012): 33-38.

[40] Jiang, H., Zhang, J., Ma, H. et al.(2017) “Mining authorship characteristics in bug repositories” Sci. China Inf. Sci. (2017) 60: 012107.

[41] A. Podgurski et al. (2003), “Automated support for classifying software failure reports” In Proceedings “25th International Conference on Software Engineering, 2003. Proceedings.”

[42] K. Zechner, “Automatic Summarization of Open-Domain Multiparty Dialogues in Diverse Genres,” Computational Linguistics, vol. 28, no. 4, pp. 447-485, 2002.

[43] X. Zhu and G. Penn, “Summarization of Spontaneous Conversations,” Proc. Ninth Int’l Conf. Spoken Language Processing (Interspeech ’06-ICSLP), pp. 1531-1534, 2006.

[44] O. Rambow, L. Shrestha, J. Chen, and C. Lauridsen,“Summarizing Email Threads,” Proc. Human Language Technology Conf. North Am. Chapter of the Assoc. for Computational Linguistics (HLT-NAACL ’04), 2004.

[45] R.J. Sandusky and L. Gasser, “Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management,” Proc. Int’l ACM SIGGROUP Conf. Supporting Group Work (GROUP ’05), pp. 187-196, 2005.

[46] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, “On the Use of Automated Text Summarization Techniques for Summarizing Source Code,” Proc. 17th Working Conf. Reverse Eng. (WCRE ’10), pp. 35-44, 2010.

[47] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay- Shanker, “Towards Automatically Generating Summary Comments for Java Methods,” Proc. 25th Int’l Conf. Automated Software Eng. (ASE ’10), pp. 43-52, 2010.

[48] L. Moreno, J. Aponte, G. Sridhara, A Marcus, L. Pollock, and K. Vijay-Shanker, “Automatic Generation of Natural Language Summaries for Java Classes,” Proc. IEEE 21st

[49] A Chowdhury, G. G. (2003). Natural language processing. Annual review of information science and technology, 37(1), 51-89.

[50] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." Journal of Machine Learning Research 12.Aug (2011): 2493-2537.

[51] Vogel, Stephan. "PESA: Phrase pair extraction as sentence splitting." Proc. of the Machine Translation Summit. Vol. 10. 2005.

[52] Collados, José Camacho. "Splitting Complex Sentences for Natural Language Processing Applications: Building a Simplified Spanish Corpus." Procedia-Social and Behavioral Sciences 95 (2013): 464-472.

[53] Y. Zhang, M. Chen and L. Liu, "A review on text mining," 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2015, pp. 681-685. doi: 10.1109/ICSESS.2015.7339149

[54] E. A. Calvillo, A. Padilla, J. Muñoz, J. Ponce and J. T. Fernandez, "Searching research papers using clustering and text mining," CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing, Cholula, 2013, pp. 78-81. doi: 10.1109/CONIELECOMP.2013.6525763

[55] Li Xinwu, "Research on text clustering algorithm based on improved K-means," 2010 International Conference On Computer Design and Applications, Qinhuangdao, 2010, pp. V4-573-V4-576. doi: 10.1109/ICCDA.2010.5540727

[56] Fasheng Liu and Lu Xiong, "Survey on text clustering algorithm -Research present situation of text clustering algorithm," 2011 IEEE 2nd International Conference on Software Engineering and Service Science, Beijing, 2011, pp. 196-199. doi: 10.1109/ICSESS.2011.5982288

[57] J. Lin, X. Li and Y. Jiao, "Text Categorization Research Based on Cluster Idea," 2010 Second International Workshop on Education Technology and Computer Science, Wuhan, 2010, pp. 483-486. doi: 10.1109/ETCS.2010.413

[58] Z. Shi, M. Shi and C. Li, "The prediction of character based on recurrent neural network language model," 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), Wuhan, 2017, pp. 613-616. doi: 10.1109/ICIS.2017.7960065

[59] Yu Wang, "A new concept using LSTM Neural Networks for dynamic system identification," 2017 American Control Conference (ACC), Seattle, WA, 2017, pp. 5324-5329. doi: 10.23919/ACC.2017.7963782

[60] R. Dey and F. M. Salemt, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, 2017, pp. 1597-1600. doi: 10.1109/MWSCAS.2017.8053243

[61] Z. Tang, Y. Shi, D. Wang, Y. Feng and S. Zhang, "Memory visualization for gated recurrent neural networks in speech recognition," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, 2017, pp. 2736-2740. doi: 10.1109/ICASSP.2017.7952654

[62] https://github.com/summanlp/evaluation/tree/master/ROUGE-RELEASE-1.5.5#start-of-content

[63] NLTK; Available from: https://www.nltk.org/.

[64] Kiros, R.a.Z., Yukun and Salakhutdinov, Ruslan and Zemel, Richard S and Torralba, Antonio and Urtasun, Raquel and Fidler, Sanja, Skip-Thought Vectors. arXiv preprint arXiv:1506.06726, 2015.

[65] Lin, Chin-Yew. (2004). ROUGE: A Package for Automatic Evaluation of summaries. Proceedings of the ACL Workshop: Text Summarization Braches Out 2004. 10.

[66] Karolina Owczarzak, John M Conroy, Hoa Trang Dang, and Ani Nenkova. 2012. An assessment of the accuracy of automatic evaluation in summarization. In Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization. ACL, 1–9.

[67] P. Bafna, D. Pramod and A. Vaidya, "Document clustering: TF-IDF approach," 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, 2016, pp. 61-66. doi: 10.1109/ICEEOT.2016.7754750

[68] F. M. Barcala, J. Vilares, M. A. Alonso, J. Grana and M. Vilares, "Tokenization and proper noun recognition for information retrieval," Proceedings. 13th International Workshop on Database and Expert Systems Applications, Aix-en-Provence, France, 2002, pp. 246-250. doi: 10.1109/DEXA.2002.1045906

[69] P. Han, S. Shen, D. Wang and Y. Liu, "The influence of word normalization in English document clustering," 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Zhangjiajie, 2012, pp. 116-120. doi: 10.1109/CSAE.2012.6272740