International Journal of Gender, Science and Technology

..ISSN: 2040-0748

UGC Care Group I Journal

Vol-12 Issue-02 Nov 2023

AUTOMATIC KEYWORD AND SENTENCE-BASED TEXT SUMMARIZATION FOR SOFTWARE BUG REPORTS

Medoji Laxman Chari, Gandhe Tharun Kumar, Kyamaji Shiva Kumar Rao, Jorrigala Sri Ram,
Ms.B. Koteswari(Assistant professor),
Department of CSE,
MALLA REDDY INSTITUTE OF TECHNOLOGY AND SCIENCE, Telangana, Hyderabad.

ABSTRACT

The purpose of text summarization is to extract useful information from texts in an effective manner. The goal of the suggested unsupervised method is to synthesize comprehensive and diverse information from bug reports, which are software artifacts. The suggested method extracts relevant and meaningful keywords and keyphrases using Rapid Automatic Keyword Extraction and the term frequency-inverse document frequency technique. Sentence extraction makes use of fuzzy C-means clustering to glean those with cluster memberships above a certain threshold. To choose which sentences to utilize, a rule-engine is used. The clustering approach selects keywords and phrases from the retrieved material, and the rules are built using the domain knowledge. Using Apache bug reports, the suggested technique produces a summary that is both cohesive and logical. The retrieved summary is enhanced using hierarchical clustering for redundancy removal and re-ranking of the resulting summary. The freshly built Apache project Bug Report Corpus (APBRC) and the old Bug Report Corpus (BRC) are used to assess the suggested method. Precision, recall, F-score, and pyramid precision are some of the performance indicators used to compare the outcomes.

I. INTRODUCTION

There is a plethora of data accessible online now from a variety of sources. It is a tedious and time-consuming operation to read whole text texts and extract essential information from the massive amounts of accessible data. Text summary is used to automatically get pertinent information in a concise manner. It takes human intelligence to glean useful information from text in order to provide an accurate summary of a written document. Document summation[1,2], essay or news summary[3,4], and e-mail summarization[5,6] are only a few of the many applications of automatic text summarization. This is in addition to the many open source projects that use software repositories like Jira and Bugzilla to handle a large number of problem reports [7]-[9]. Automated operations such as finding duplicate bug reports [10], fixing defects [12], and triaging bug reports [13]–[15] have been carried out to manage a wide amount of reports. Software engineers and testers must read through lengthy defect reports, often consisting of hundreds of words, in order to do their responsibilities. Since bug report history is not just a general text summary, it requires particular topic expertise for a tester or developer to comprehend. The authors of this study zeroed in on one of software projects' most prized artifacts—a summary of defect reports. Title, one-line description, BugID, comprehensive description, comments from several authors, and more are all included here. Finding and repairing the software project's issue is made easier with this knowledge. Because it is a painstaking endeavor to read and comprehend a whole bug report, bug summary is a new area of study that aims to assist many developers in improving the process of fixing bugs.

There are two types of algorithmic summarization that have been discussed in the literature: abstractive and extractive. By preserving the original context, abstractive summarization alters the semantic representation, word order, and natural language of a document. Using deep learning methods, a huge stride forward is made in this field. When compared to other methods in the literature, some researchers have achieved very high levels of accuracy using Convolution neural networks (CNN)[16], Recurrent neural networks (RNN)[17], Reinforcement learning, and Generative Adversarial networks (GAN) [18]. Since deep learning is a supervised technique, and typical golden summaries aren't accessible in every area, the lack of training data is a big drawback to using these approaches. In contrast, extractive summarization builds a compressed summary by selecting phrases from the source material that have the same syntax and sequence. A number of

methods for automatically summarizing bug reports have been suggested in the literature, including supervised [19]-[21] and unsupervised [22], [23] methods. Rastkar et al. [19] conducted research using a supervised technique and built the Bug Report Corpus (BRC) comprised of 36 bug reports from different open source projects. A total of twenty-four characteristics were computed for every bug report. These features were grouped into four groups: lexical, participant, length, and structure. Annotators trained a logistic regression classifier on a corpus of golden summaries that were created by hand. A low f-score of 40%, a recall of 35%, and a pyramid accuracy of 66% are all shown in the findings. In order to enhance the outcomes of [19], Jiang et.al. put out an alternative method called PRST[20]. The authors built a new corpus called the Modified Bug Report Corpus (MBRC) by considering 36 bug reports from the BRC corpus [19] and its duplicate bug reports. The likelihood of each phrase was calculated using a logistic regression classifier after the page-rank algorithm had assessed the textual similarity among them. A summary was generated by merging the data and then selecting phrases with a high likelihood value. With the same recall and f-score values, accuracy and pyramid precision showed a small improvement.

III. PRELIMINARY CONCEPTS

In this part, we will go over some of the main ideas behind bug report summary. Fast automated keyword extraction, fuzzy clustering, hierarchical clustering, and pre-processing of textual data from bug reports are all part of it.

(A) Pre-processing of Text

The collection of raw bug reports has to be pre-processed in order to provide a summary. Segmentation, tokenization, stop word removal, punctuation removal, and stemming are some of the standard pre-processing procedures that are used.

Segmentation: This method extracts individual sentences from bug reports using a delimiter. The retrieved text is then saved in the same sequence as the first bug report. Tokenization is a sentence-splitting technique that deconstructs text into its component words, symbols, and phrases.

The first stage is to eliminate stop words from the textual data. These are words like "the," "a," "and," and "this" that are used too often and do not provide any semantic value. Exclamation points, question marks, and other punctuation marks are deleted in this process.

By stripping words of their suffixes and prefixes, stemming returns them to their original form. One example is the diminutive of the word "presentation," which is the basic form of the word alone.

Document term matrix (DTM) is the structural representation that results from converting text data through all these phases. In other words, it shows how often each phrase appears in a document.

B) RAKE, or rapid automated keyword extraction

The Rake method is domain-agnostic and used for keyword extraction. It then sorts the textual input into possible keywords, which are groups of words that appear in the text. It finds potential keywords by counting how often certain content terms appear in a prospective keyword. Rake creates a word array from the textual material in order to extract keywords. The next step is to divide this word array into groups of related words using phrase delimiters and stop words. Each potential keyword is given a fixed place in the text and is defined as a series of adjacent words. To see how often different words in a proposed keyword appear together, we build a matrix of word co-occurrence. Potential

II. MOTIVATION

The software system's most prized artifact is the bug report. Description, id, title, and a few preset fields are all part of it, as are a plethora of comments written by different developers. Prior research indicated that out of 275 total bug reports, 200 were sourced from the Mozilla open source project. Each bug report contains hundreds of phrases, some of which are duplicates, making it a difficult process for any developer or tester to read and understand them all by hand. Software testers may save time and effort by using a new method for summarizing issue reports. In order to produce high-quality bug summaries, we are aiming to develop a fresh, new method of automated text summarizing that can identify the domain expertise in bug reports. The summary is made up of prime phrases, which include descriptions and code snippets with '{', '}', '{tmp field}','sql', '\', 'public static', and '='. The code snippets from the description and comments make up the most important portion of the problem report, yet Tf-idf, unigram, bigram, and centric algorithms cannot capture them, as indicated in literature [21]. This study fills a gap in the literature by using the RAKE approach to extract hitherto unexplored code snippets from the textual data of bug reports [19]-[21].

words for inclusion in a document are sometimes referred to as a sequence of informative words, or content words. After all potential keywords have been found, a score is given to each one. The overall score of any potential keyword is equal to the sum of its content word scores [30, 31]. The procedure for rating each term is shown in the following image:

• The first step is to determine how often each content word appears in a certain text by calculating its frequency, or freq(CW).

Deg (CW) is the degree of a word that is computed after the frequency (CW). The degree is calculated by counting the total number of words in the content word's candidate keywords.

Last but not least, we calculate the degree-to-frequency ratio of content words, which is represented by

$$Deg\,(CW) \Big/ Freq\,(CW) \qquad (1)$$

Table 2 and the text of bug Id HDFS-7707 show the results of the candidate keywords' score calculation.

Bug ID HDFS-7707 is referenced in Table 1.

1.Edit log corruption due to delayed block removal again.
2. Edit log corruption is seen again, even with the fix of HDFS- 6825.
3.Prior to HDFS-6825 fix, if dirx is deleted recursively, an Op_CLOSE can get into edit log for the filey under dirx, thus corrupting the edit log restarting NN with the edit log would fail.

Table 2. Words used in the text and their corresponding score

Content words	Edit	Log	Corrup -tion	delaye d	Block	Re mo v- al	Fi x
Deg(CW)	12	12	6	3	3	3	2
Freq(CW)	5	5	2	1	1	1	2
Ratio (Deg/freq)	2.4	2.4	3	3	3	3	1
Content words	Recur - sively	Op_ clos e	Corrup -ting	Filey	Restart -ing	Dir x	fai l
Deg(CW)	2	1	1	1	1	2	1
Freq(CW)	1	1	1	1	1	2	1
Ratio (Deg/freq)	2	1	1	1	1	1	1

Candidate Words:
1. edit-log-corruption
2.delayed block-removal
3.fix-dirx recursively
4.op_close – edit
5.log – filey – dirx
6.corrupting – edit log-
7.restarting NN – edit log- fail

Score (edit log corruption) = score (edit) + score(log) + score (corruption)
= 2.4 + 2.4 + 3= 7.8 (2)

Score (edit log) = score(edit) + score (log)
= 2.4 + 2.4 = 4.8 (3)

Therefore, the fast automated keyword extraction algorithm selects lengthier candidate keywords and terms that appear mainly. The recall and accuracy achieved by RAKE are greater when compared to other keyword extraction approaches that are already available, such as Tf-idf, Text Rank, Ngram with tag, and others[30, 31].

Option C: Fuzzy Clustering

Clustering is a kind of unsupervised machine learning that uses data division into separate groups. The data points are grouped together according to how close or far apart they are. Hard clustering and soft (fuzzy) clustering are the two primary methods of clustering. As each data point may only take on a binary value of 0 or 1, hard clustering restricts data to a single cluster per dataset.

With soft clustering, each data point has an equal chance of being part of any cluster, up to a specific point. The method most often used for fuzzy clustering is fuzzy C-means clustering [32]. Reduce the total distance (Euclidean distance) between each data point and the cluster center; this is the primary goal of fuzzy c-means (FCM). As a first step, FCM uses a random number generator to choose cluster centers and gives each data point in each cluster a membership value. The degree of membership is determined by solving equations (4) and (5), and the cluster centers are updated after each iteration.

$$M_{ij} = 1 \Big/ \sum_{K=1}^{C} (E_{ij} / E_{ik})^{\left(\frac{2}{f}-1\right)} \qquad (4)$$

$$C_j = \left(\sum_{i=1}^{n} \frac{(M_{ij})^f x_i}{(M_{ij})^f}\right. \qquad (5)$$

$$O_f = \sum_{i=1}^{n}\sum_{j=1}^{c}(M_{ij})^f \left||x_i - c_j\right||^2 \quad (6)$$
$$O_f = \text{Objective function}$$

D) CLUSTERING AND HEIRARCHY

The unsupervised clustering method known as hierarchical clustering [36] combines datasets that are similar into groupings. The two main varieties are agglomerative and divisive. The agglomerative method of clustering works from the bottom up, assigning data to specific clusters, while the divisive method works from the top down, splitting up a single cluster into many related ones. Agglomerative hierarchical clustering is used in the suggested method. Here, we treat each data point as an independent cluster and determine their closeness distance using that model. A single cluster is created by merging comparable clusters based on proximity. In an iterative process, the distance between each freshly created cluster is calculated and then combined into one. Dendograms, which are similar to trees and record several merging sequences, are used to display the results of hierarchical clustering. Also, there are a number of ways to figure out how close or similar two clusters are to one other. The average linking approach is used in this paper.

$$Sim\,(C1,C2) = \sum_{i=1}^{n}\sum_{j=1}^{n} Sim\,(D_i,D_j)/|C1|^*|C2|^*$$

IV. RESEARCH METHODOLOGY

This section explains a number of topics that were important to the investigation. Section IV (E) provides the pseudo code for the suggested method, whereas Section III (F) explains the framework and its several components.

The four cornerstones of summarization are the following in the suggested method:

• Rich information coverage on main topics.

• Data spread out with little overlap in subject matter.

• Revising the summary's order to begin with more important and relevant information.

Relative compression to the source material. An extracted summary of software problem reports may be generated using the methods shown in Fig. 1. The following are descriptions of various parts:

Section A: Creating the Core and Preprocessing Text

The twenty-one bug reports that make up the Apache Project Bug Reporting Collection (APBRC) have been set up. A corpus is built by extracting and segmenting into sentences the one-line description, extended description, and comments of bug reports. Tokenization, stop word removal, and stemming are the next typical pre-processing procedures applied to the sentences. Constructing the document term matrix (DTM) follows pre-processing. The procedure is executed in R utilizing the 'tm' and 'NLP' packages.

B. Extracting Features

Features are extracted after textual data pre-processing. Every textual feature is extracted throughout the feature extraction procedure. Each aspect of text is classified as either a word feature or a sentence feature. In order to find the optimal mix of text feature extraction methods for maximum coverage and relevance in bug reports, several tests have been conducted. The different components of the suggested method are detailed.

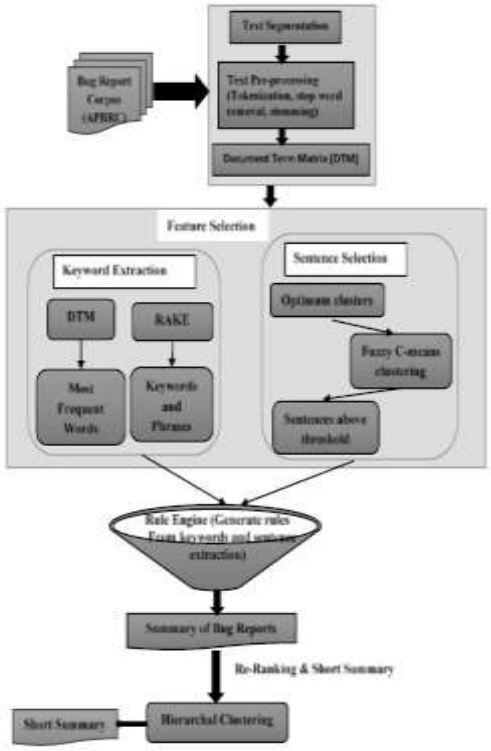
(2) Characteristics at the Sentence Level 2.1 Sentence placement: It says that summary paragraphs should always include the document's leading phrases because of how significant they are. The calculation is:

Sentence characteristics are examined after sentences are chosen based on the most relevant keywords. Here are a few aspects at the sentence level that are detailed.

$$P(S_i) = 1 - ({}^{i-1}/_{N}) \quad (8)$$

2.2.2 Sentence Length: Longer sentences, as opposed to shorter ones, are more informative, critical, and thus need to be included in summaries. Calculation is done as

$$(S_i) = \frac{number\ of\ words\ in\ S_i}{Total\ numnber\ of\ words\ in\ longest\ sentence} \quad (9)$$



Flow Diagram 1 Illustration of the suggested method

V. Implementation

A. DATA COLLECTION

A corpus of bug reports is the primary need for evaluating the efficacy of the suggested method. This is accomplished by compiling 21 bug reports from five separate Apache Software Foundation projects: Hadoop-Hdfs, Hadoop-common, hive, groovy, and base. The corpus is called the Apache Projects Bug Report Corpus (APBRC). A technology called the bug report collecting system (BRCS) was used to retrieve the bug reports from the period of 2015-2018. An enormous amount of data, including titles, descriptions, comments, and other properties, was retrieved from a pool of bug reports. This led to the retrieval of 21 bug reports for APBRC. A minimum of ten comments were retrieved from bug reports of varying lengths. The linkages of the patches were eliminated in order to generate the corpus. In table 3, you can see the statistical data about the complaints of bugs. The current bug report corpus (BRC) is used to assess the suggested method as well. The BRC comprises of bug reports from four open source projects: Firefox, eclipse, gnome, and Kde. You may find all of the bug reports and the source code at link1.

In addition, the recently built APBRC corpus differs from the preexisting BRC corpus in that the bug report includes code samples when creating the summary, which are not in the BRC corpus.

Data from the APBRC Corpus, Organized by Table 3

Name of the Project	Total # of bug reports	# of bug reports selected	Total # of sentences in selected bug reports
Hadoop-Hdfs	10423	3	349
Hadoop-Common	13229	5	620
Hive	13949	5	309
Groovy	7957	2	201
Hbase	16757	6	707

Additionally, a bug report in the APBRC corpus often contains more words than in the BRC dataset.

B. Reports of Bugs: Annotation

In order to generate a manual summary, annotators must annotate bug reports. Three annotators are allocated to each bug report in the APBRC corpus in order to annotate them. Every one of the three annotators chooses a set of sentences to mark, and then their combined scores give each sentence a score between zero and three. A phrase is given a score of 0 if no annotator has chosen it; a value of 1 if at least one annotator has chosen it; and scores of 2 and 3 are similarly allocated. For the golden standard summary, sentences with a score of 2 or 3 are used.

Section C: Survey Questions

We have developed a small set of Research Questions (RQ) to assess how well the suggested method for bug summarization works.

Question 1: When tested on an existing corpus, does the suggested method outperform the state-of-the-art method?

Question 2: How does the suggested method fare on different bug report corpora?

D. Criteria for Assessment

Algorithm performance is evaluated using a number of measures, including Precision, Recall, F-score, and Pyramid Precision, according to the literature. The precision is determined by dividing the total number of sentences in Generated Summary (GS) by the number of sentences that are common in both GS and GSS.

$$\text{Precision} = \frac{|GS \cap GSS|}{|GS|} \quad (9)$$

The focus of recall is on picking out a distinct subset of the total number of sentences in the Golden Standard summary divided by the number of sentences in the Generated summary.

$$\text{Recall} = \frac{|GS \cap GSS|}{|GSS|} \quad (10)$$

$$\text{F-score} = 2.\frac{Precision.Recall}{Precision+Recall} \quad (11)$$

$$\text{Pyramid precision} = \frac{\#\ AL\ (top\ ranked\ sentences)}{Total\ \#\ AL(Summary\ length} \quad (12)$$

E. PROCESS ILLUSTRATION

A bug report #Hadoop-Hdfs-13112 from the Hadoop-hdfs Apache project demonstrates the procedure of the suggested technique.

First thing to do: Keyword extraction

Performing pre-processing on textual data, building a document term matrix (DTM), and then extracting the most common phrases are the steps involved in keyword extraction.

Cause, Corruption, deadlock, expiration, token, acquire, change, nointerruptslock, transitions, fseditlog, concurrent, getdelegationtoken, locking, interrupte, expose, exception

Second Step: Keyword Phrase Extraction

Step two in the process of keyword extraction is this. All of the bug reports are run through a rapid automated keyword extraction algorithm to glean the most relevant and meaningful keyword phrases. We installed the software "rapidraker" and found the keyword phrases together with their scores. In table 4, you can see a few of the sentences that got good marks.

Table 4. Extracted keyword phrases with score

Keyword/Keyword phrases	Score
general {{fseditlog}} thread safety issue	16.366667
normal edit logging activities	11.05113
nointerruptslock technically isn't	9.000
log queue overflows	8.363636
start segment edits	8.18750
expiry isnt essential	8.00000
public void logupdatemasterkey	8.0000
concurrent edit logging	7.55113
abstract secret manager	7.47619
async edit logging	7.05113
token expiration edits	6.06250
locks interruptible method	5.082707
whitespace issues	4.20000
exit condition	4.0000
log corruption	3.8636

VI. RESULTS

Presented below are the discussion and analysis of the study questions developed to assess the efficacy of the suggested method. Four performance metrics—Precision, Recall, F-score, and Pyramid Precision—are used to assess the effectiveness of the suggested method.

I. FIRST QUESTION:

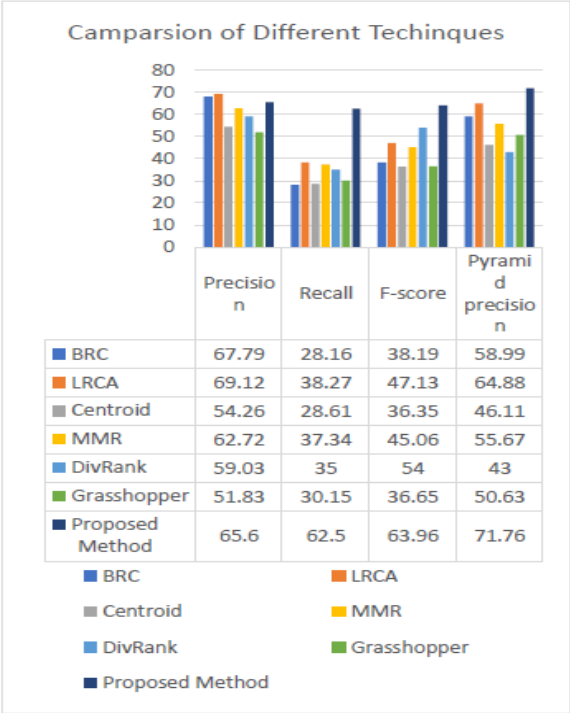
against answer RQ1, we compare the suggested method's performance against that of preexisting supervised and unsupervised methods for summarizing bug reports. Figure 5 shows bar graphs and a table detailing the trial outcomes according to four assessment criteria. When comparing the proposed Automatic keyword and Sentence based approach to BRC[19] and LRCA[21], the results show that the former achieves better results in terms of precision (2.19% vs. 3.52%), recall (34.3% vs. 25.77%), F-score (16.83%), and overall performance (6.88%). But when looking at total performance as measured by F-score, we see an improvement of 25.77% and 16.83%.The suggested method beats Hurried [23] in terms of Precision (7.19%), Recall (26.03%), F-score (20.95%), and Pyramid Precision (15.44%) when compared to unsupervised approaches. The bar chart shows that the suggested method outperforms competing unsupervised algorithms as Grasshopper, DivRank, Centroid, and Maximum Marginal Relevance (MMR). According to the results shown above, the new suggested method always beats the competition. As a result, bug reports might be effectively summarized. Depicted in Figure 5.

Question 2 of the Research (B)

We additionally address RQ2 and analyze the suggested strategy on a freshly built APBRC corpus to further verify its findings. Using RAKE and Fuzzy C-means clustering, we automatically extract keywords and sentences to build bug report

summaries. In comparison to the BRC technique, this unsupervised learning strategy yields better results. The average values for Precision, Recall, F-score, and Pyramid Precision are 78.22%, 82.18%, 80.10%, and 81.66%, respectively. We compute the outcomes in terms of Pyramid Precision, F-score, Recall, and Precision. Take #HDFS-13112, for example, as seen in table 6. Table 7 shows the total amount of sentences in the APBRC corpus for bug reports, broken down by both the Generated and Golden Standard Summary.

See Figure 6 for a comparison of the findings on the APBRC (21 bug reports) and BRC (36 bug reports) corpora using the proposed technique.



	Precision	Recall	F-score	Pyramid precision
BRC	67.79	28.16	38.19	58.99
LRCA	69.12	38.27	47.13	64.88
Centroid	54.26	28.61	36.35	46.11
MMR	62.72	37.34	45.06	55.67
DivRank	59.03	35	54	43
Grasshopper	51.83	30.15	36.65	50.63
Proposed Method	65.6	62.5	63.96	71.76

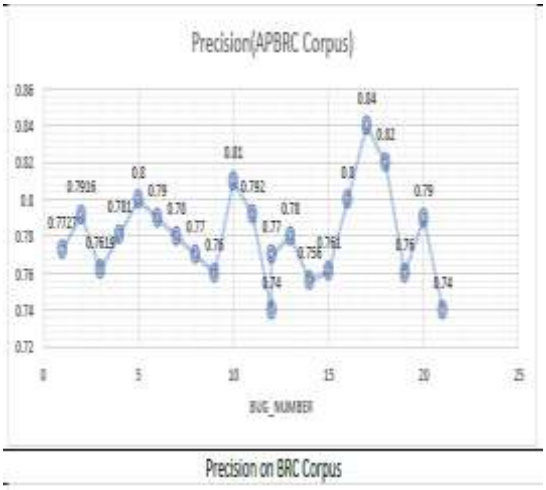
In Figure 5, we can see how various algorithms perform on the BRC corpus.

Results from the APBRC Corpus, Table 6.

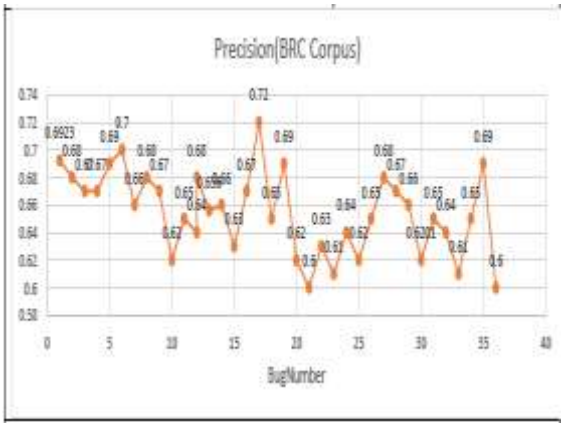
Number of Sentences in Generated Summary (GS)	21
Number of Sentences in Golden Standard Summary (GSS)	18
Number of sentences in common (GS∩GSS)	16
Precision $\frac{\lvert GS \cap GSS \rvert}{\lvert GS \rvert}$	= (16/21) = 0.7619
Recall $\frac{\lvert GS \cap GSS \rvert}{\lvert GSS \rvert}$	= (16/18)= 0.8888
F-score 2. $\frac{Precision.Recall}{Precision+Recall}$	= 0.8203
Pyramid Precision $\frac{\#\ AL\ (top\ ranked\ sentences)}{Total\ \#\ AL(Summary\ length}$	= (45/53) = 0.8490

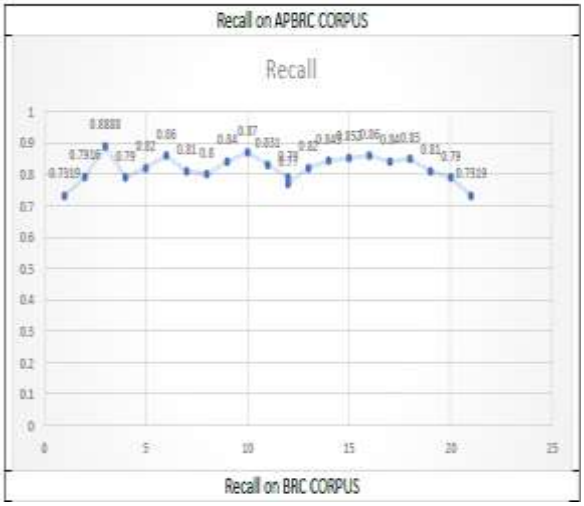
Table 7: APBRC Corpus Detailed Description

Bug #	# total sentences	# sentences manual summary	#sentences generated summary	Bug #	# total sentences	# sentences manual summary	#sentences generated summary
Bug 1	137	23	22	Bug 12	81	20	19
Bug 2	136	24	24	Bug 13	29	8	7
Bug 3	57	19	21	Bug 14	53	13	11
Bug 4	154	28	26	Bug 15	38	10	9
Bug 5	47	12	10	Bug 16	124	28	29
Bug 6	197	40	41	Bug 17	57	18	19
Bug 7	160	55	54	Bug 18	48	12	11
Bug 8	107	23	25	Bug 19	117	27	25
Bug 9	103	24	25	Bug 20	275	50	52
Bug 10	53	12	13	Bug 21	86	22	21
Bug 11	108	26	27				

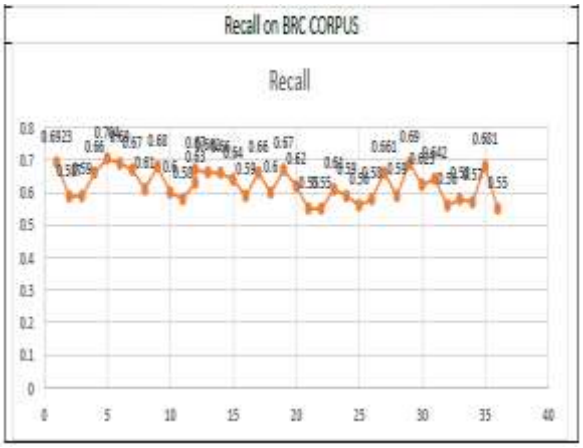


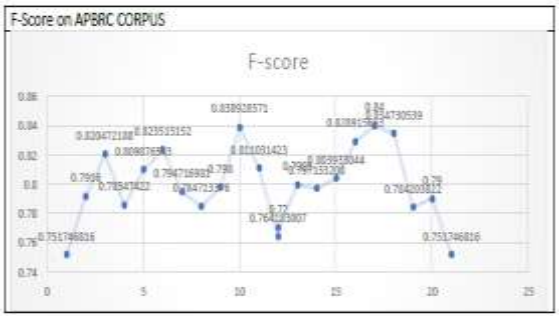
Precision on BRC Corpus

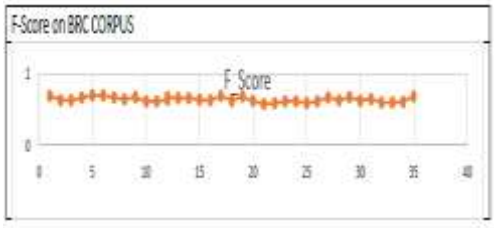




Recall on BRC CORPUS







VIII. RELATED WORK

This section explains the technique used to automatically produce summaries of bug reports.

A. Condensing information

One approach to document summarizing is known as extractive summarization, and it works by snippeting individual phrases from longer texts. Web articles[2], conferences and phone calls[3], summary of several documents[1],[38],[39], automated text highlighting [40], and countless more fields of application have made use of it. A number of methods are used to generate an extracted summary, including genetic algorithms [41], conditional random fields [42], neural networks [43], and semantic similarity techniques such latent semantic analysis [44]-[46]. Fuzzy logic [28], k-means clustering [47], inverse document frequency [49], and other unsupervised learning approaches have also been used alongside these supervised learning techniques. An approach to fuzzy logic-based multi-document summarizing was suggested by D. Patel et.al. Document summaries were generated using fuzzy rules that were based on word and sentence attributes. Cosine similarity measurements are used to eliminate repetition among the resulting summary's phrases. Several criteria, including ROUGE, content coverage score, relative utility, and pyramid accuracy, were used to assess the technique using the DUC 2004 news dataset [28]. R. Abassi-ghaletaki et.al. summarized the DUC 2002 dataset using a fuzzy logic system, evolutionary algorithm, and cellular learning automata. The ROUGE-1 performance metric was used to assess the methodology, and the findings were contrasted with those of alternative methodologies. According to the findings, other methods are surpassed by evolutionary algorithms when paired with the fuzzy logic approach [47]. A approach for extracting the most informative sentences was suggested by F.B. Goularte et.al. using fuzzy metrics. Following the textual data pre-processing, a sentence score was determined by using several attributes. The scores were then normalized to a range of 0 to 1. A rule-based system generated 27 rules for fuzzy analysis, with relevance determined by means of the bell membership function. Texts in Portuguese obtained from VLE were used to test the suggested technique. According to the findings, fuzzy

summarizing enhances the resulting summary's in formativeness and helps identify topics for use in virtual learning environments (VLEs) by both instructors and students [29].

IX. CONCLUSION

An unsupervised method for automatically summarizing software bug reports using sentence-based characteristics and keywords is presented in this work. Using two feature extraction techniques, the shortcomings of corpus-oriented and document-oriented approaches found in literature are eliminated: Quick automated keyword extraction and term frequency inverse document frequency are used. Because of its unsupervised nature, RAKE does not need domain knowledge and is language and domain neutral. Clustering is used for phrase extraction from bug reports. Four approaches are used to determine the optimal number of clusters: K-means, GSS, Silhouette, and WSS. To handle the ambiguity in bug reports, we use fuzzy c-means clustering to find the optimal number of groups and then we choose words inside each cluster based on their membership value. In order to provide a coherent summary, a rule-based method is used to merge sentence and keyword characteristics. Either a single useful and relevant statement at the bottom of the produced summary or many phrases with comparable meaning may make up the summary. Consequently, hierarchical clustering is executed to re-rank the phrases in a created summary and to remove repetition. With regard to the first problem report, a compression ratio of 20% has been reached. This method is suggested for the purpose of summarizing bug reports. Since the BRC corpus is the only readily accessible training set for bug report summarizing, this unsupervised learning method may be used with any set of bug reports to produce relevant and accurate summaries. This is because fuzzy c-means method diminishes problems with incompleteness, ambiguity, and vagueness. By selecting the most relevant and significant phrases from each cluster, fuzzy C-means clustering covers every detail of the problem report.

Since the suggested method is unsupervised and does not need a particular training dataset, it may be tested on any dataset in order to provide a thorough and succinct summary. Both the newly built APBRC corpus and the current BRC corpus are used to assess the method. There is a comparison between the proposed method and many current methods, both supervised and unsupervised, including BRC, LRCA, MMR, Centric, DivRank, Grasshopper, and Hurried. The results show that the automatic keyword and sentence based approach outperforms BRC and LRCA by 34.3 percent in recall, 25.7 percent in F-score, 16.83% in pyramid precision, and 6.88% in overall accuracy, while BRC and LRCA outperform them by 2.19% and 3.52%, respectively, in accuracy. But when looking at total performance as measured by F-score, we see an improvement of 25.77% and 16.83%. The results were far better than those of the previous unsupervised methods. Precision, recall, f-score, and pyramid precision average out to 78.22%, 82.18%, 80.1%, and 81.66% for the APBRC corpus, respectively. When compared to other methods already in use for bug summarizing, the suggested automated method performs better.

REFERENCES

[1] *K. Zechner, "Automatic Summarization of Open-Domain Multiparty Dialogues in Diverse Genres," 2002.*

[2] L. Zhou, E. Hovy, and M. Rey, "A Web-Trained Extraction Summarization System," Proc. HLT-NAACL Conf., no. May, pp. 1–7, 2003.

[3] X. Zhu and G. Penn, "Summarization of Spontaneous Conversations," pp. 1531–1534, 2006.

[4] G. Murray and G. Carenini, "Summarizing Spoken and Written Conversations," EMNLP '08 Proc. Conf. Empir. Methods Nat. Lang. Process., no. October, pp. 773–782, 2008.

[5] O. Rambow and J. Chen, "Summarizing Email Threads."

[6] S. Wan and K. Mckeown, "Generating Overview Summaries of Ongoing Email Thread Discussions." In Proceedings of the 20th international conference on Computational Linguistics, Association for Computational Linguistics, p. 549, 2004.

[7] X. Xia, D. Lo, E. Shihab, and X. Wang, "Automated Bug Report Field Reassignment and Refinement Prediction," IEEE Trans. Reliab., vol. 65, no. 3, pp. 1094–1113, 2016.

[8] A. E. Hassan, "Software Intelligence : The Future of Mining Software Engineering Data," pp. 161–165, 2010.

[9] N. Carolina, D. Lo, T. Xie, S. Thummalapenta, and N. Carolina, "Data Mining for Software Engineering," pp. 55–62, 2009.