Automatic Summarization of Bug Reports Using Keyword Extraction

Md Mainul Hasan Polash
Department of Mathematics and Computer Science
University of Lethbridge
polash@uleth.ca

***Abstract*— In the current software development process, it is vital to access the previous bug reports to understand a particular bug or how previous changes have been made to solve the bug while assigning the tasks of fixing the bugs to a developer. Each developer who works with a bug report usually commits their solution in the bug reports. Besides, some bug reports may contain conversations of more than two developers when a group works to fix a problem. It takes so long time for a new person to review these solutions and conversations in order to understand the current situation of the problem. In order to save the time of the developers, we propose a system which summarizes the bug reports automatically using Natural Language Processing (NLP) technique called keyword extraction. We select the most likely sentences those represent the key message of a particular report by using RAKE algorithm and ILP formulation. I collect the bug report corpus from summarizing software artifacts of University of British Columbia[1]. The proposed system increases the quality of the summaries of the reports which motivate us to add something new with this project.**

KEYWORDS

Summarization, NLP, RAKE algorithm, Candidate keyword.

I. INTRODUCTION

The task of shortening a text document consisting only the important information which can give a overview of the full document is termed as summarization. Summaries can be divided into two parts based on output types - extractive and abstractive. Extractive summaries extract the important sentences from the original documents without modifying them while abstractive summaries contain new phrases, paraphrasing sections or words which are not present in the original document. Considering the fact of increasing data in today's digital world, it becomes mandatory to generate the summaries with the help of a computer tool to reduce the work load of humans as well as save time. It not only saves time but also provides impartial summaries, makes the searching information task easier and improves the effectiveness of indexing.

A bug refers to the inappropriate behaviour of a software system. The manager or responsible person appoints developers to solve the bug with the access to the corresponding bug report. Bug reports pile up valuable information about the nature of the bug, description of previous attempts to solve the same issue, conversations among developers while solving the bug. However, bug report is different from conventional texts as the developers write the report in a complex way [4]. The users can also mention the problems they are facing as well as friendly suggestions about how to solve the issue through adding their comments to the bug report. The length of the bug may vary in length and quality. Some of them may consist of few comments. Others contain informal conversations among developers and those can be very large in terms of length. Developers are trying to use summarization tool to extract the important information of a bug report which will offer them the gist of a report. They adopt both abstractive and extrative summarizer [5], [6] to make summaries of their bug reports. However, most of the summarizer used in bug report corpus are extractive as we only need to know exact information appeared in the document, don't care much about modified information.

Extracting keywords is one of the most important tasks while working with texts. Readers benefit from keywords because they can judge more quickly whether the text is worth reading. For example, website creators benefit from keywords because they can group similar content by its topics and algorithm programmers benefit from keywords because they reduce the dimensionality of text to the essential features. Now we are trying to implement the same thing while analyzing the previous bug reports in order to fix a bug. My system can generate the summary of a bug report which will help the developers to understand the gist of a report without reading the whole report and saves the value time of the developers. An attempt has been made to do the similar thing, and it proves that existing extractive summarizers can provide satisfactory results which motivated me to work on this project [3]. The contribution of our work is as follows:

- Our proposed system can generate the extractive summaries of bug reports which can save time of the developers who are assigned to solve bugs.
- The keywords of a report can also be found with our system.
- It determines the importance of a phrase from their score value.

The remaining of the paper is described as follows. In Section II, we give a brief overview of related works. In Section III, we present how the system of generating summaries using keyword extraction is modeled. Section IV describes the experimental studies conducted and section V

[1] https://www.cs.ubc.ca/cs-research/software-practices-lab/projects/summarizing-software-artifacts

discusses the results. Finally, Section VI concludes the paper and presents some directions for future works.

II. RELATED WORKS

The occurrences of a particular word in a document measure the importance of that word. In, [7], authors proposed an unsupervised approach which introduced some new ideas, such as stemming and stop word filtering, were put forward in this paper that have now been understood as universal preprocessing steps to text analysis. Later, models associated Hidden Markov Models [2] has brought significant improvement in the quality of extractive summary. Now, deep learning has become very popular in the field of text summarization to solve the coherence, ranking and sorting problem. The selection and coherence problems are solved by methods that improve diversity, minimize redundancy and pick up phrases and/or sentences that are somewhat similar so that more relevant information can be covered by the summary in lesser words and the summary is coherent. The ranking problem is solved by learning a certain set of features for each sentence[8]. [9] present SummaRuNNer, a Recurrent Neural Network (RNN) based sequence model for extractive summarization of documents and show that it achieves performance better than or comparable to state-of-the-art.

But, training such a model which is developed using deep neural network requires a large amount of preprocessed data. We are unable to get such data to train a deep neural network based model. However, for the bug report we only need to care about the keywords or key-phrases. To serve our purpose, we use Rapid Automatic Keyword Extraction (RAKE) [3] algorithm to generate scores for trigram, bigram and unigram.

III. PROPOSED APPROACH

We consider the summarizing software artifacts of University of British Columbia which is available in XML format. We use Python's ElementTree [4] parser to parse the XML file, extract the comment section of each bug report and store them in seperated text file. We process the data by removing stop-words. Later, we apply RAKE algorithm to the processed texts which produce the score of every trigram, bigram and unigram. We calculate the score of each sentence using the score generated by RAKE algorithm and sort them in ascending order based on their scores. Finally, we select the highest scored sentences according where the length of the summary is determined by the length of the original report. Fig1, gives an overview of our approach.

[2]https://en.wikipedia.org/wiki/Hidden_Markov_model

[3]http://www.cbs.dtu.dk/courses/introduction_to_systems_biology/chapter1_textmining.pdf

[4]https://docs.python.org/2/library/xml.etree.elementtree.html

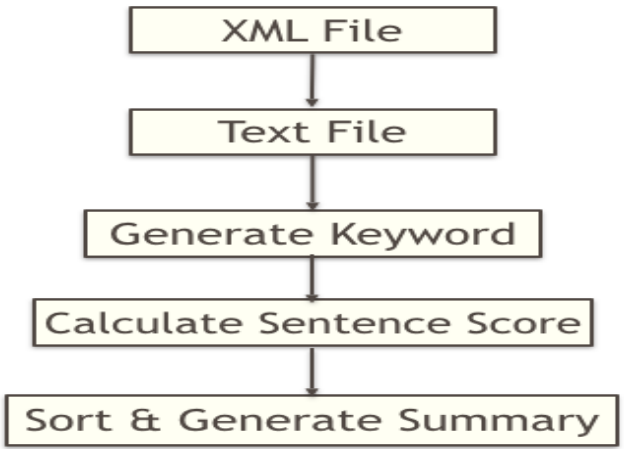

Fig. 1. Overview of our proposed system.

A. DATA SOURCE AND PREPARATION

We were considered Android bug reports to do a summary. All the data have already been processed and stored in a XML file [10]. There is a title and under the title the description of the previous attempt of solving the bug is written. When we started to work with this dataset, we had discovered that most of the reports contain five-six lines of text. It is not necessary to summarize this small reports and there are no such techniques to implement machine learning techniques on such small data. So we have decided to work on a different dataset. In [3], authors build a bug report corpus with good summaries to train their model. The bug report corpus has 36 bug reports containing a total of 2,361 sentences which are collected from four open source software projects. The 36 bug reports have been chosen randomly from all bug reports containing between 5 and 25 comments with two or more people. The bug reports have mostly conversational content and they vary in length. The original bug reports [5] contain ID, title, date and time of committing comments, name of the person who commits comments etc. They also generate an annotated version of the corpus with the help of human annotator. The annotators generate both abstractive and extractive summary of each report which are considered as the standard summary of a bug report. Each report has three set of summaries provided by three different annotators. We consider the original version as the input to generate summary with our system while annotated version is used to evaluate our system.

B. PARSING XML

Extensible Markup Language(XML) is an inherently hierarchical data format which is always represent with a tree. Pyhton have a module called xml.etree.ElementTree which implements a simple Application Program Interface(API) for parsing XML data. ElementTree represents the whole documents as a tree while each Element represents each node of the tree. Uisng this module, we access the bug report using

[5]https://www.cs.ubc.ca/labs/spl/bugreports.xml

the BugReport ID and parse the child nodes. Our main goal is to get the comments. Thus, we parse all the commenting sentences from each bug report and store them in a text file for further processing. After removing stop-words from the text we get the processed on which we apply RAKE algorithm.

C. GENERATE KEYWORD

We have extracted the keywords from each sentence with Rapid Automatic Keyword Extraction (RAKE)[6] algorithm which is an unsupervised, domain and language independent method. This method is used extensively to extract keywords from a sentence. A typical keyword extraction algorithm has three main components:

- Candidate selection: Here, we parse all possible words, phrases, terms or concepts from the texts that can potentially be candidate keywords. This done by splitting the text into an array of words with the word delimiters and later splitting them into sequences of contiguous words by phrase delimiters. Words within a sequence is considered as a keyword.
- Properties calculation of a keyword: For each candidate, we need to calculate properties that indicate that it may be a keyword. For example, a candidate appearing in the title of a bug report is a possible keyword. We evaluate (i) word frequency, (ii) word degree and (iii) ratio of degree to frequency metrices to calculate the score of a keyword.
- Scoring and selecting keywords: All candidates can be scored by either combining the properties into a formula or using a machine learning technique to determine the probability of a candidate being a keyword. Then we sum the score of each member of a keyword to calculate the score of a keyword. We apply a score threshold and select the final set of keywords.

D. GENERATE SUMMARY

After getting the keywords with their scores, we search through the texts to find the occurrences of keywords in a text file and calculate the score of each file in the document based on the keyword score. If a trigram is found in a sentence the weight is multiplied by three where multiply the scores by two in case of bigrams. Traversing through the whole document we get a list which contains sentences with their corresponding score values. The sentence having the highest scoring value can be an important sentence to reflect the information in the text. We sort the list based on the score values. According to the length of the original text we select some of the highest scored sentences to generate the summary of a text. We repeat this for all the 36 bug reports.

IV. EVALUATION PLAN

The annotation version of the dataset contain three extractive summaries for each report. We plan to evaluate our system using only Precision, Recall and F1 score to compare the accuracy rate of my system with the summaries generated by human annotators.

I will use the following equations:

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$$

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ FalsePositive}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

fig2 describes the determination process of true positive, false positive and false negative. We have calculated the precision, recall and F1 score of each report with respect to all the annotators. Then we calculate the average precision, recall and F1 score.

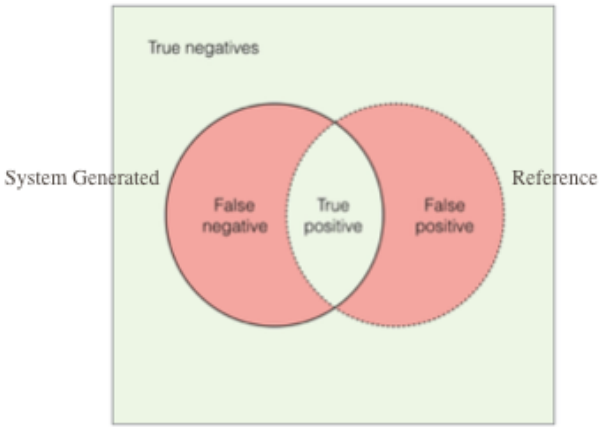

Fig. 2. Evaluation plan.

V. RESULT

Fig3 describe the result of our evaluation. It is shown from th efigure that our system can produce better precision for Annotation 2 (47%) which is lower than baseline (57%). In case of recall, annotation 1 (55%) improves the baseline (35%) by almost (20%). Overall, our system (48%) increase the F1 score of baseline (40%)by almost (8%).

Dataset	Precision	Recall	F1 Score
Annotation 1	0.42	0.55	0.47
Annotation 2	0.47	0.54	0.50
Annotation 3	0.44	0.50	0.47
Average	0.44	**0.53**	**0.48**
Baseline	**0.57**	0.35	0.40

Fig. 3. Evaluation Result.

[6] http://www.cbs.dtu.dk/courses/introduction_to_systems_biology/chapter1_textmining.pdf

VI. CONCLUSION

It is very important for a new developer to understand the current status of a bug report before planning to solve the newly occurred bug. Studies shows this task require a large amount of time. But this time consuming task can be done easier if it is possible to generate summaries of bug reports only mentioning the important informations. Current summarizers have brought a revolutionary change in text processing. But, there are not enough work done to summarize the bug reports with the current summarization tool. In the paper, we try to summarize a small bug report corpus using some unsupervised techniques. Our system generate better summaries than the baseline model which we tried to improve according to the result. The developer explicitly selects a XML file as input where the XML file is implicitly parsed and extracted the comments from the XML file using a parser. The extracted data is the input of our recommendation engine which can generate the summary of the input text. Finally, the result can be shown by using an user interface. Our system has all the three main components(user context, recommendation engine, output mode) of a Recommending Systems for Software Engineering(RSSE) system. Fig4 shows the overview of our RSSSE system.

A. LESSION LEARNED

Previously, I was more focused to supervised machine learing model. After doing this, I realized it's not certain that supervised models cal always generate better result than the unsupervised model. I have gained a better knowledge about parsing XML files.

B. FUTURE PLAN

In future, I will try to generate summary for bug reports with deep neural networks to see how it performs on bug reports. I will evaluate the result of my system by rouge automatic summarization recall package. I also have a plan to generate length limited summaries using ILP formulation.

REFERENCES

[1] Christoph Hannebauer, Michael Patalas, Sebastian St unkel, and Volker Gruhn. 2016. Au-tomatically recommending code reviewers based on their expertise: an empirical com-parison. InProceedings of the 31st IEEE/ACM International Conference on AutomatedSoftware Engineering, ASE 2016, Singapore, September 3-7, 2016, pages 99110.

[2] Chin-Yew Lin. 2004. Rouge: a package for automatic evaluation of summaries.

[3] S. Rastkar, G. C. Murphy and G. Murray, "Automatic Summarization of Bug Reports," in IEEE Transactions on Software Engineering, vol. 40, no. 4, pp. 366-380, 2014.

[4] R. Lotufo, Z. Malik and K. Czarnecki, "Modelling the Hurried bug report reading process to summarize bug reports," 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, 2012, pp. 430-439.

[5] Pimpalshende. A. N, Overview of Text Summarization Extractive Techniques, IJECS Volume 2, Issue 4, pp. 1205-1214, 2013.

[6] Neelima Bhatia, ArunimaJaiswal, Trends in Extractive and Abstractive Techniques inText Summarization. International Journal of Computer Applications (0975-8887) Volume 117- No.6, May 2015.

[7] Luhn, H. P.: The Automatic Creation of Literature Abstracts. IBM Journal of Research and Development, vol. 2, issue 2, 159165 (1958). doi: 10.1147/rd.22.0159

[8] Sukriti Verma, Vagisha Nidhi, "Extractive Summarization using Deep Learning" CoRR 2017, Volume abs/1708.04439.

[9] Ramesh Nallapati, Feifei Zhai, Bowen Zhou, "SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents" , Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17).

[10] Emad Shihab, Yasutaka Kamei, and Pamela Bhattacharya. 2012. Mining challenge 2012:The android platform. In The 9th Working Conference on Mining Software Repositories,page to appear.