The 34th Canadian Conference on Artificial Intelligence
DOI: 0

Canadian AI 2021
(Vancouver, May 25-28, 2021)

Study on Different Text Analysis Techniques for Automatic Summarization of Software Bug Report

First Author[†,*], Second Author[†], Third Author[◇]
[†] Affiliation One
[‡] Affiliation Two
[◇] Affiliation Three

Abstract

Considering the exponential increase of data being stored worldwide, it is quickly becoming impractical to summarize this data manually and extract important information in a short time. In an attempt to save time for the developers, we propose a system which summarizes software engineering bug reports automatically using several text analysis techniques such as keyword extraction using the Rapid Automatic Keyword Extraction (RAKE) algorithm and semantic similarity using the Universal Sentence Encoder (USE). In addition with these two unsupervised methods, we introduce a deep neural network based supervised model to summarize the bug reports using Bidirectional Encoder Representations from Transformers (BERT) model. The increase in summary quality provided by our proposed systems motivated us to add something new to the project which led us to develop our own Question-Answer (Q/A) dataset. We have validated some pretrained Q/A models with our dataset and trying to develop a new state-of-the-art model for Q/A task.

Keywords: Natural Language Processing, Software Recommendation System, Text Summarization, Bug Report.

1. **Introduction**

Summarization involves creating short, concise sentences which contain the most relevant information from a larger body of text. By generating the summaries with the help of a computer tool, such as our model, we can save man-hours and provide impartial summaries, making the search for information more efficient and increasing the accuracy of indexing data. The bug reports contain normal conventional data while several developers are working on a bug report. The length of a bug report increases while new developers are assigned to solve the same type of errors. Moreover, the bug reports are not written in a structured format [1] mostly. The length of the reports vary in length and quality, with some containing only a few sentences. However, reports may contain informal conversations among developers which can cause them to be very long. So, new developers find it difficult to capture the main topics of a bug report without reading the full document which can take months. Thus, developers are adopting both abstractive and extractive summarizers [2, 3] to condense their bug reports. However, most of the summarizers used in the bug report corpus are extractive since only exact information appearing in the document is relevant in most cases. Our system can generate the summary of a previous bug report which will allow the individual to understand the main context of a report without reading the entire document. We have successfully accomplished this, showing that our summarizer system can provide satisfactory results. The contribution of our work is as follows:

- We propose three summarization models which generate the extractive summary of a bug report and save developer's time.

[*] corresponding_author@example.ca

- We develop a new Q/A dataset for software bug reports. This dataset is modified from the bug report corpus which was originally collected by summarizing software artifacts from the University of British Columbia[1].

2. Related Works

With data collection quickly becoming one of the important industries globally, the ability to automate large text documents' summary is a considerable asset. The number of occurrences of a word throughout a document corresponds to the importance of that word. In, [4], the authors proposed an unsupervised approach which introduced some new ideas, such as stemming and stop word filtering. These are now understood as universal preprocessing steps for text analysis. Now a days, deep learning has become very popular in the field of text summarization in order to solve the coherence, ranking and sorting problems. 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. The ranking problem is solved by learning a certain set of features such as sentence length, sentence position relative to paragraph and many more [5]. The inclusion of embedding make the summarization task easy and efficient by computing the similarity between words or sentences and determine the most similar to the target one. WordNet [6] is a novel lexical database of semantic relations that determines the semantic similarity between two words while in other work [7], a system has been described, which learns the semantic similarity between sentences. Word mover's distance [8] and Universal Sentence Encoder [9] are most two familiar algorithms to measure the semantic similarity between sentences based on their meaning. SummaRuNNer [10] presents a Recurrent Neural Network (RNN) based sequence model for extractive summarization of documents that achieves better performance than the state-of-the-art results at that time. But in recent past, BERT [11] based on transformer [12] model has presented state-of-the-art results for many Natural Language Processing (NLP) tasks such as summarization and question answering.

3. Data Source and Preparation

In Rastkar, Murphy, and Murray [13], authors build a bug report corpus with good summaries having three different annotation. We consider the bug report as the inputs to our system while annotated summaries are used to evaluate our system. Python has a module called xml.etree.ElementTree[2] which implements a simple Application Program Interface (API) for parsing XML data. Using this module, we access the bug reports using the BugReport ID and parse the child nodes with main objective to extract the title of the report and all associated comments.

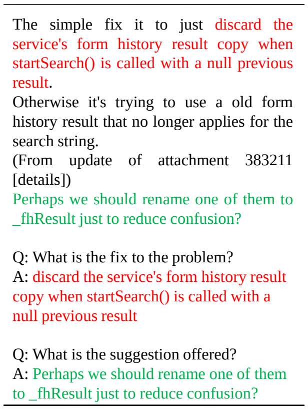
4. Q/A dataset

We have developed a question-answer pair based dataset for the Q/A task. We have 36 bug reports, each report has several blocks of text known as 'turns' each turn contains 5 to 25 comments or messages. The bug reports are chosen from four different project: Eclipse Platform, Gnome, Mozilla and KDE so that the proposed systems for the this dataset can be adopted to varieties of bug report repositories [13]. We hired 2 human annotators to generate question from the comments. They have 3 years of experience to work with different NLP based Q/A datset such as SQuAD [14] and CNN/Daily Mail dataset [15]. They have generated questions for each comment which has at least 15 words. The number

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

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

of questions varies according to the length of the comment. They have focused to generate two question from each comment where one question is related to the bug and other question is related to the solution. However, this approach is not consistent throughout all the reports because all the bug reports were not written in a specific structured format.



The simple fix it to just discard the service's form history result copy when startSearch() is called with a null previous result.
Otherwise it's trying to use a old form history result that no longer applies for the search string.
(From update of attachment 383211 [details])
Perhaps we should rename one of them to _fhResult just to reduce confusion?

Q: What is the fix to the problem?
A: discard the service's form history result copy when startSearch() is called with a null previous result

Q: What is the suggestion offered?
A: Perhaps we should rename one of them to _fhResult just to reduce confusion?

Figure 1. Question-answer pairs for a sample comment.

5. Proposed Models

We propose three models: extraction model, similarity model, and BertSum. In the following section, we will describe our three models in details.

5.1. Extraction Model

At first, we have extracted the keywords from each sentence by using the basic of RAKE [16] algorithm which is an unsupervised, domain and language independent method. This 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 is done by splitting the text into an array of words with the word delimiters. Then 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. We evaluate (i) word frequency, (ii) word degree and (iii) ratio of degree to frequency metrics 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.

We calculate the score of each sentence in the document based on the keyword score. Then we choose the most scored sentences as the summary of the report. The length of the final summary is determined by the length of the original report.

5.2. Similarity Model

Embedding is a representation of words or sentences into equivalent vector space to compute the syntactic and semantic similarity with other words or sentences. GloVE [17], fastText [18], ELMo [19] are some of the popular pre-trained word embedding which are used to represent a sentence to produce the sentence embedding. The Universal Sentence Encoder (USE) model which is implemented using TensorFlow[3] is used to encode sentences into high dimensional vectors. We chose this model since it is trained on many data sources and tasks which allows it to dynamically accommodate many different natural language understanding tasks. The model is trained with a deep averaging network (DAN) encoder [20]. In the DAN encoder, input embedding of words and bi-grams are averaged together in order to send them through a feed-forward deep neural network. Thus, the sentence embedding of a sentence is produced. We use this network as the computation time is linear to the input sequence.

The title and sentence content are then compared by the model for semantic textual similarity. The semantic similarity result is stored in another vector containing the sentence id number and the semantic similarity result (Compared to the title of the bug report). Finally they are sorted in descending order before we select the most similar sentences as the summary of the report.

5.3. BertSum

The purpose of a deep supervised neural network based model is to map a fixed length input into a fixed length output where the lengths may differ. Mostly recognized sequence-to-sequence based model has 3 portions - encoder, intermediate vector and decoder. To serve the sequence dependency in a sequence-to-sequence model recurrent networks such as Long-Short-Term-Memory(LSTM), Gated Recurrent Units(GRU) etc. are used. A new innovation transformer based model which doesn't have any recurrent network (LSTM, GRU) achieves better state-of-the-art results for some NLP tasks[11]. To compare our results, we processed our data-set using the Yang Liu's BertSum RNN Model for extractive summarization[21]. The model is trained for 50000 steps on the CNN / Daily Mail dataset[4] on three GPUs with gradient accumulation every two steps which gives us a batch size of approximately 36. Checkpoints are evaluated every 1000 steps and the top 3 are selected based on their evaluation loses to the validation set. Then the averages are reported for every set. Trigram blocking is used during the prediction process to reduce redundancies. We then evaluated this model on the bug report dataset.

6. Evaluation Plan and Result Analysis

The annotated version of the dataset contains 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 the system with the summaries generated by human annotators. Training a model which is developed using deep neural network requires a large amount of preprocessed data. These large amounts of data are difficult to obtain. For this reason, we trained BertSum model with CNN / Daily Mail dataset and later used the trained model on our data to validate the model.

The results of our evaluation are shown in Table 1 and Table 2. Our extraction model and similarity model can produce better recall and F1 score than the baseline model [13]. Rastkar, Murphy, and Murray [13] propose three logistic regression based classifier, where two of them are trained on conversational data and the other classifier is trained on the bug

[3] https://tfhub.dev/google/universal-sentence-encoder/2

[4] https://github.com/abisee/cnn-dailymail

report data. They use a set of 24 features to calculate the probability of a sentence being the part of the summary. We choose their BRC classifier which is trained on bug report data and performed better than other two classifiers as our baseline model. Our experimental result shows that keyword comes in handy for longer reports during summary generation. However, if developers need to generate summary of a small paragraph with respect to a query, in that case our similarity model produces better output. The result of our BertSum model is shown in Table 2. We haven't get a good ROUGE score to claim our model to be the state-of-the art. As we have experimented using the transfer learning approach with a pre-trained language model for small bug reports, we may get better results if we can use the longer report to fine-tune the pre-trained model. Therefore, we also need to work on to develop a better corpus by removing the unnecessary informal conversation from a bug report.

Model	Precision	Recall	F1 Score
Baseline	**0.57**	0.35	0.40
Extraction Model	0.44	**0.53**	**0.48**
Similarity Model	0.43	0.50	0.46

Table 1. Results of our two unsupervised models on bug report dataset.

Model	ROUGE-1	ROUGE-2	ROUGE-L
BertSum	30.09	11.24	29.09

Table 2. Result of our BertSum model on bug report dataset.

7. Conclusion

In this paper, our proposed Extraction and Similarity models achieved state-of-the-art results on this data. We have applied the BERT model on this data which is a new work, so there is no other work to compare the performances. We are still trying to improve the results by integrating inter-sentence transformer layers on top of the BERT output for summarization task. We are also trying to fine tune BERT for Q/A task on our newly generated dataset to achieve a satisfactory result. We also have a plan to release our system in a software firm to get the real-time performance of our system.

References

[1] R. Lotufo, Z. Malik, and K. Czarnecki. "Modelling the 'Hurried' Bug Report Reading Process to Summarize Bug Reports". In: *Empirical Softw. Engg.* 20.2 (Apr. 2015), pp. 516–548. ISSN: 1382-3256. DOI: 10.1007/s10664-014-9311-2. URL: http://dx.doi.org/10.1007/s10664-014-9311-2.

[2] M. P. A. N. "OVERVIEW OF TEXT SUMMARIZATION EXTRACTIVE TECHNIQUES". In: *International Journal of Engineering and Computer Science* 2.04 (Dec. 2017). URL: http://www.ijecs.in/index.php/ijecs/article/view/701.

[3] N. Bhatia and A. Jaiswal. "Trends in Extractive and Abstractive Techniques in Text Summarization". In: *International Journal of Computer Applications* 117 (May 2015), pp. 21–24. DOI: 10.5120/20559-2947.

[4] H. P. Luhn. "The Automatic Creation of Literature Abstracts". In: *IBM J. Res. Dev.* 2.2 (Apr. 1958), pp. 159–165. ISSN: 0018-8646. DOI: 10.1147/rd.22.0159. URL: http://dx.doi.org/10.1147/rd.22.0159.

[5] S. Verma and V. Nidhi. "Extractive Summarization using Deep Learning". In: *arXiv e-prints*, arXiv:1708.04439 (Aug. 2017), arXiv:1708.04439. arXiv: 1708.04439 [cs.CL].

[6] G. Recski et al. "Measuring semantic similarity of words using concept networks". In: *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016, pp. 193–200.

[7] B. Liu et al. "Matching Natural Language Sentences with Hierarchical Sentence Factorization". In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1237–1246. ISBN: 9781450356398. DOI: 10.1145/3178876.3186022. URL: https://doi.org/10.1145/3178876.3186022.

[8] G. Huang et al. "Supervised Word Mover's Distance". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 4869–4877. ISBN: 9781510838819.

[9] D. Cer et al. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL].

[10] R. Nallapati, F. Zhai, and B. Zhou. "SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 3075–3081.

[11] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[12] A. Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[13] S. Rastkar, G. C. Murphy, and G. Murray. "Automatic Summarization of Bug Reports". In: *IEEE Transactions on Software Engineering* 40.4 (Apr. 2014), pp. 366–380. ISSN: 0098-5589. DOI: 10.1109/TSE.2013.2297712.

[14] P. Rajpurkar et al. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text". In: *CoRR* abs/1606.05250 (2016). arXiv: 1606.05250. URL: http://arxiv.org/abs/1606.05250.

[15] K. M. Hermann et al. "Teaching Machines to Read and Comprehend". In: *CoRR* abs/1506.03340 (2015). arXiv: 1506.03340. URL: http://arxiv.org/abs/1506.03340.

[16] S. Rose et al. "Automatic keyword extraction from individual documents". In: *Text mining: applications and theory* 1 (2010), pp. 1–20.

[17] J. Pennington, R. Socher, and C. Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[18] P. Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://www.aclweb.org/anthology/Q17-1010.

[19] M. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

[20] M. Iyyer et al. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1681–1691. DOI: 10.3115/v1/P15-1162. URL: https://www.aclweb.org/anthology/P15-1162.

[21] Y. Liu. "Fine-tune BERT for Extractive Summarization". In: *CoRR* abs/1903.10318 (2019). arXiv: 1903.10318. URL: http://arxiv.org/abs/1903.10318.