Summarizing bug reports of Android apps hosted on GitHub with NLP

Taihe Chen – 101047827

Supervisor: Dr. Olga Baysal

COMP4905-Honours Project

Carleton University

December 6, 2020

Abstract

Currently, Android applications are becoming more and more popular but almost all Android developers will face to a large number of issues reports every day, and then developers will debug their code based on the issue reports posted by other users. Therefore, analyzing those issue reports of Android apps hosted on GitHub has huge positive effects on Android API development. In the project, based on issue reports text hosted on GitHub, the textRank algorithm and BERT model in the machine learning is used to obtain summaries with a large number issue report. According to comparison of the performance of the two models, although BERT can be said state-of-the-art in NLP techniques, there is still no steady improvement on TextRank in the quality of summaries and combining context.

Acknowledgements

Thanks to all my friends for their encouragement and support.
Thanks to Khadija Osman for helping me to download and understand data.
Thanks to Ryan Taylor for providing me with a virtual machine with extremely excellent performance.

Special thanks my supervisor Dr. Olga Baysal for her very useful paper of textRank algorithm, responding to my email, answering questions and solving many difficulties rapidly when I was in trouble. She also spent much time in meeting with me and discuss about my honours project, which helped me determine the next steps for my research.

Table of Contents

1. Abstract 1

2. Acknowledgments 2

3. Table of Contents 3

4. List of Figures 4

5. List of Tables 5

6. Body 6

    6.1 Introduction 6

        6.1.1 Motivation 6

        6.1.2 Background information 6

        6.1.3 Work Objectives and Structure 7

    6.2 Data and Processing 9

        6.2.1 Data source 9

        6.2.2 Data Pre-process 13

    6.3 TextRank Algorithm 15

        6.3.1 PageRank Algorithm 15

        6.3.2 TextRank Algorithm 17

        6.3.3 Implementation 18

    6.4 BERT Algorithm 19

        6.4.1 Overview and Comparison 19

        6.4.2 Input representation 20

        6.4.3 Mask LM 21

        6.4.4 Fine-Tuning Process 22

        6.4.5 Implementation 23

    6.5 Results 27

    6.6 Conclusion 28

7. References 29

List of Figures

Diagram 6-2-1-1 Amount of Android Applications 9

Diagram 6-2-1-2 Key method name 10

Diagram 6-2-1-3: The second type of issue reports 11

Diagram 6-2-1-4: The third type of issue reports 12

Diagram 6-3-1-1: Five pages contain links to pointing to one another 15

Diagram 6-4-1-2: The input representation of BERT 20

Diagram 6-4-1-3: Process of Fine-Tuning 22

List of Tables

Table 6-3-1-1 WebPages have link directing to each other 15

Table 6-3-1-2: The matrix M will be initialized as above 16

Table 6-5-1-1: Summarization results generated by Textrank and BERT 24

6.1 Introduction

6.1.1 Motivation

Issue reports are bug reports written manually by some developers who wish to understand and run the project other than project code contributors. Sometimes it can be just like compiler error information, sometimes it would be a description of the problem written manually. If we can summarize a large number of issue reports accurately, then project code contributors can learn about their project issues better, instead read hundreds or thousands of issue reports one by one. However, even the state-of-the-art algorithm like BERT leaves much to be desired based on accuracy. It is critical to find out the excellent algorithm for issue reports text summarization.

6.1.2 Background Information

Usually, text summarization can broadly be divided into two categories, extractive summarization and abstractive summarization [PRATEEK, 2018].

The extraction method selects key words and sentences from the original text to form a summary. This method naturally has a low error rate in grammar and syntax, which ensures a certain effect. The traditional abstraction method uses graph method, clustering to complete the unsupervised abstraction. The state-of-the-art algorithm also including index marking method through SummaRuNNer model and other methods [Ramesh et al., 2017]. TextRank algorithm is a typical traditional abstraction method, it takes sentences as nodes and uses similarity between sentences to construct undirected edges. It iteratively updates node values

with weights on the edges, and finally selects N nodes with the highest score as summary.

Abstractive summarizations use advanced NLP techniques to generate an entirely new summary and some parts of this summary may not even appear in the original text [PRATEEK, 2018]. Compared to the extraction method, it performs better in sentences consistency and flexibility with Seq2Seq. However, Seq2Seq still have two primary problems: out-of-vocabulary (OOV) and generating duplication. These problems have been solved by a new Seq2Seq based on attention instead of Long Short-Term Memory (LSTM) [Abigail et al., 2017]. Using the Copy mechanism to calculate the probability of copy or generation at each step of decoding. Since the glossary is fixed, this mechanism can choose to Copy words from the original text into the summarization, effectively solving the problem of OOV. By using the Coverage mechanism, it is necessary to consider the attention weight of each previous step before decoding. In combination with Coverage loss, avoid continuing to consider the part that has gained high weight. This mechanism can effectively alleviate the problem of generating duplication.

6.1.3 Work Objectives and Structure

The goal of this project is to firstly find out top100 most valuable Android applications hosted on GitHub and then implement two different text summarization models with issue reports from these applications based on textRank algorithm and BERT Algorithm, finally evaluate their performance for the completely same text data.

The report is divided into five sections. First, there is an introduction of the whole report including project motivation, project methodology. In the second part will look at overall data generation and how to process data in order to better

performance. The third part is to introduce TextRank algorithm from PageRank algorithm and then implement it with issue reports and get text summarization [PRATEEK, 2018]. The fourth part is to introduce BERT algorithm and use it for text summarization by importing Pytorch library [Derek, 2019]. Finally, through comparing the output of two models to determine BERT model performs better or not.

6.2 Data and Processing

Datasets can be separated by two sections, data source and data pre-processing.

6.2.1 Data Source

Because the datasets text are issue reports of Android applications, I firstly select all Android applications from datasets including all applications in GitHub with GH-Torrent and I got 316,500 applications (Figure 6-2-1-1) [Georgios, 2013]. In order to find out applications whose issue reports including one of specific method names (Figure 6-2-1-2), GitHub API is used to get each issue report of all valid applications and 620 pieces of issue reports are valid.

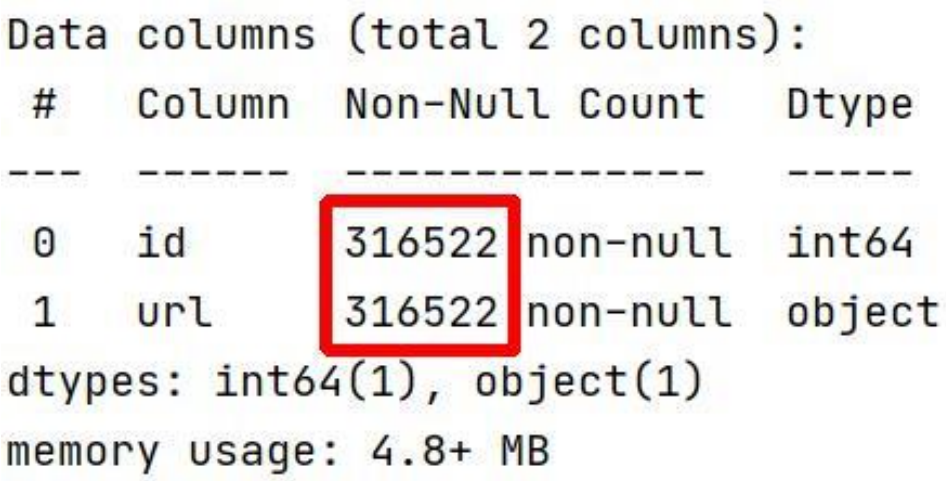

Figure 6-2-1-1 Amount of Android Applications

	A	B
1	findViewById	setContentView
2	onCreate	onViewCreated
3	onSaveInstanceState	getView
4	onPostExecute	onActivityCreated
5	notifyDataSetChanged	startActivity
6	onBackPressed	onDestroy
7	onStart	onDraw
8	onCreateView	onPause
9	doInBackground	onStop
10	onBindViewHolder	onNewIntent
11	onClick	finish
12	onResume	onRestoreInstanceState
13	onActivityResult	invalidate
14	startActivityForResult	OnItemClickListener
15	onMeasure	onCreateOptionMenu

Diagram 6-2-1-2 Key method names

Issue reports content are pushed by developers want to run the project instead of project code contributors, as a result, the issue reports would be extremely chaotic. Mostly, the issue reports can be divided into three categories. The first one involves a normal sentence, like ‘Make sure your adapter calls notifyDataSetChanged when its content changes’, NLP techniques works good for this type of issue reports. The second category is just post what they got from compiler bug reports (Diagram 6-2-1-3). For this type of issue report, NLP completely cannot work. As a result, they are all removed from datasets even if a key method name is involved from compiler bug report. For the third type of issue reports, some of text are typed manually but some of text copied from compiler bug report (Diagram 6-2-1-4). In this setting the text copied from compiler will be removed but the text typed will be retained. Finally, 160 issue reports were used for text summarization.

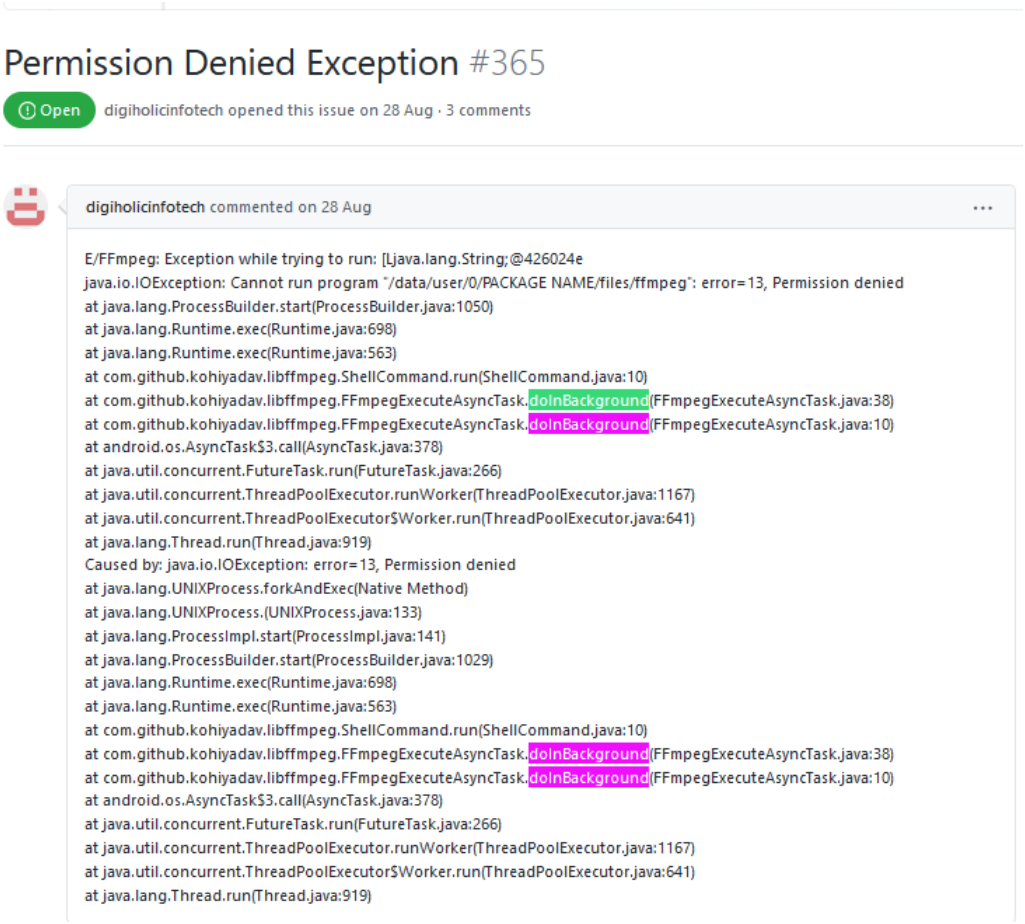

Diagram 6-2-1-3: The second type of issue reports

Diagram 6-2-1-4: The third type of issue reports

In order to make text summarization more accurate and meaningful, the 160 issue reports are divided into several files according to which key method name it including. However, for some key method names, there is only one or two pieces of issue reports and less than five sentences, this type of text summarizations is meaningless because of such little sentences.

6.2.2 Data Pre-process

Overall, the data pre-process contained three distinct steps:

1. Segmentation

A whole issue report usually including several sentences. Using sent_tokenize function of ntlk library as a primary function for dividing a whole issue report into several sentences based on full stop. Some sentences may only have one or two word because of improper usage of full stop and it will be removed from dataset because it's not a full sentence.

2. Cleaning and Normalization

Stopwords are unnecessary words in a sentence. Removing them has no effect on understanding the meaning of the whole sentence. In the text, there will be a large number of function words, pronouns or verbs and nouns without specific meanings. These words are not helpful for text analysis, and I often hope to remove these "stop words". It is implemented by nltk.corpus library and easily get word set for stop words. Otherwise, to reduce the number of words, all uppercase words will be converted to lowercase words with replace function.

3. Feature Extraction

There are two possible approaches to extract features in NLP techniques, the first involves tf-idf, and the second involves word2vec. For the textRank algorithm, it will create vectors for each word in sentences. First fetching vectors in the size of 100 elements for the constituent words in a sentence and then take mean/average of those vectors to arrive at a consolidated vector for the sentence.

$$Word_1 = (n_{101}, n_{102}, n_{103} \dots \dots n_{199}, n_{200})$$

$$Word_2 = (n_{201}, n_{202}, n_{203} \dots \dots n_{299}, n_{300})$$

Then the vector of a sentence with only word1 and word2 is:

$$Sentence = (\frac{n_{101} + n_{201}}{2}, \frac{n_{102} + n_{202}}{2}, \dots \dots \frac{n_{199} + n_{299}}{2} \frac{n_{299} + n_{300}}{2})$$

6.3 TextRank Algorithm

6.3.1 PageRank

TextRank algorithm is improved from pageRank algorithm (Diagram 6-3-1-1).

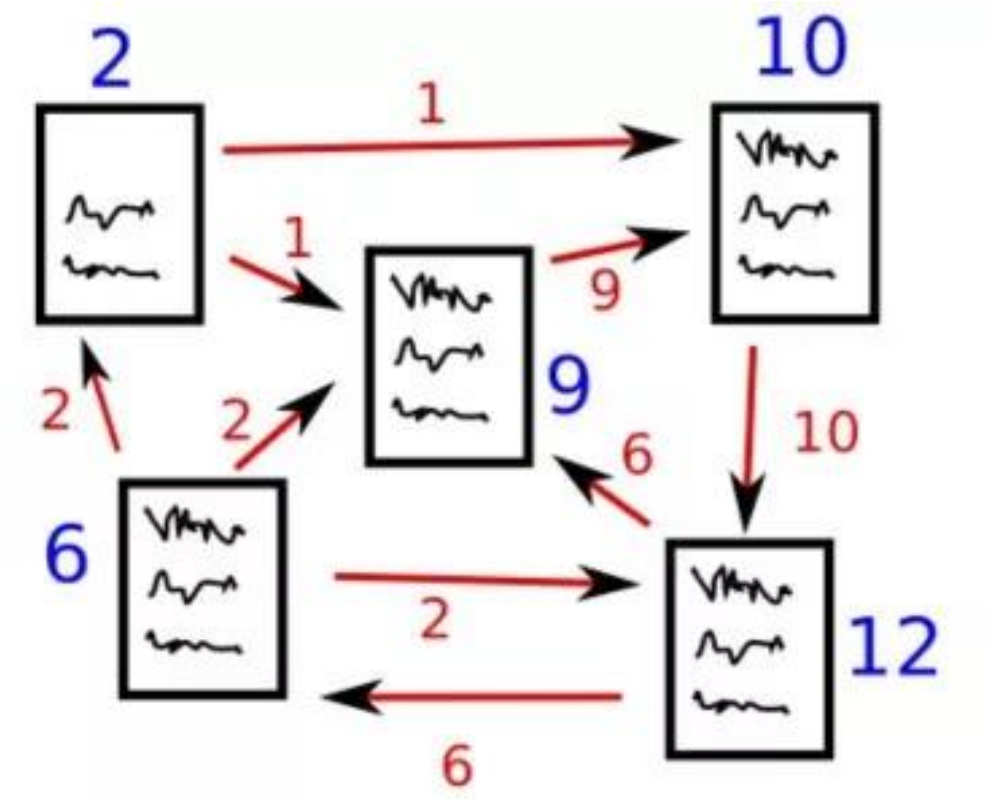

Diagram 6-3-1-1: five pages contain links to pointing to one another [PRATEEK,2018].

Currently suppost I have three pages: w1, w2 and w3. These pages contain links pointing to one another.

WebPage	Links
W1	[w3, w2]
W2	[w1]
W3	[]

Table 6-3-1-1 WebPages have links directing to each other

Now, I have: Web page w1 has links directing to w2 and w4; w2 has a link for page w1, w3 no links and hence it will be called a dangling page. In order to rank these pages, I would have to compute a score called the PageRank score [Prateek, 2018]. This score is the probability of a user visiting that page.

To rank these pages, I would have to compute a score called the PageRank score. This score is the probability of a user visiting that page. To get the probability that the user will jump from one page to another, square matrix M with n rows and n columns will be created, where n is the number of pages.

In order to initialize the probabilities of each page, the steps below will be applied.

1. Probability of going from page i to j, i.e., M [i][j], is initialized with

$$\frac{1}{\text{number of unique links in web page } w_i}$$

2. If there is no link between the page i and j, then the probability will be initialized with 0.

3. If a user has landed on a dangling page, then it is assumed that he is equally likely to transition to any page. Hence, M [i][j] will be initialized with

$$\frac{1}{\text{number of web pages totally}}$$

Therefore, in my case the matrix M will be initialized as follows:

	W1	W2	W3
W1	0	0.66	0.66
W2	1	0	0
W3	0.33	0.33	0.33

Table 6-3-1-2: the matrix M will be initialized as above

Finally, the values in this matrix will be updated in an iterative fashion to arrive at

the web page rankings.

6.3.2 TextRank

TextRank is highly similar to PageRank but using sentences in place of web pages. Otherwise, TextRank will use similarity scores to replace the probability in the PageRank and store similarity scores in a square matrix. The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation.

The score of the iterative process is according to the formula:

$$S(V_i) =$$

$$(1-d) + d \times \sum_{j \in In(V_i)}^{b} \frac{w_{ji}}{\sum V_k \in Out(V_j) w_{jk}} WS(V_j)$$

$w_{ij} is\ the\ weight\ from\ V_i\ to\ V_j, d\ is\ the\ probability\ from\ a\ node\ to\ any\ other\ nodes,$

$which\ is\ 0.85, In(V_i)\ is\ the\ set\ of\ points\ to\ V_i,$ th J in the figure,

Out(Vj)is the sum of weights of all points connected w_i.

In the graph constructed by TextRank, nodes are sentences and

$s_{ij}\ is\ similarity\ score\ between\ sentence\ i\ and\ sentence\ j.$

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \subseteq S_i \& w_k \subseteq S_j\}|}{\log(|S_i|) + \log(|S_j|)}$$

When calculating the score of each node in the graph, it is also necessary to specify any initial value for the node in the graph, usually set to 1 and then

recursively compute it until it converges.

6.3.3 Code Implementation

```
# similarity matrix (all zeros)
sim_mat = np.zeros([len(sentences), len(sentences)])

from sklearn.metrics.pairwise import cosine_similarity

for i in range(len(sentences)):
    print(i)
    for j in range(len(sentences)):
        if i != j:
            sim_mat[i][j] = cosine_similarity(sentence_vectors[i].reshape(1, 100), sentence_vectors[j].reshape(1

nx_graph = nx.from_numpy_array(sim_mat)
scores = nx.pagerank(nx_graph)
ranked_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)), reverse=True)

# Specify number of sentences to form the summary
sn = 4

# Generate summary
for i in range(sn):
    print(ranked_sentences[i][1])
```

Firstly, defining a zero matrix of dimensions (n * n), then initialize this matrix with cosine similarity scores of the sentences and using Cosine Similarity to compute the similarity between a pair of sentences. From_numpy_array function of networkx library is used to convert the similarity matrix sim_mat into a directional graph. The nodes of this graph will represent the sentences and the edges will represent the similarity scores between the sentences. Through pagerank function in network library to calculate weight of each node, the loop will end after iterations for all nodes in the graph. Finally, extracting the top 3 sentences based on their scores after n iterations.

6.4 BERT ALGORITHM

6.4.1 Overview and Comparison

In addition to abstractive summarization and extractive summarization according to output types, text summarization can also be divided into supervised abstracts and unsupervised abstracts according to supervised and unsupervised data.

TextRank is a typical unsupervised learning model. Word embedding technique is used for transfer words to a vector with 100 numbers through Glove dataset, every word is corresponding to a fixed vector in Glove file [Jeffrey et al., 2014]. Word Embedding solves the problem of scarcity of features of traditional machine learning methods. By mapping a Word into a low-dimensional dense semantic space, similar words can share context information, thus enhancing their generalization ability. High-quality word vectors can be obtained through unsupervised training and then the semantic data can be transferred to specific tasks with less data. However, what word embedding learned is all meaning of a word. For the bank instance, bank can be a financial institution, also can be “by the river”. If we have to encode the meaning with a fixed vector, we can only encode the meanings of both words, but the fact is only one of the meanings in a sentence makes sense, which is obviously problem.

However, due to the fact that all abstractive summarization algorithms need labeled dataset for training, which is supervised learning, issue reports data is not allowed to use supervised learning because of no labeled text data, this article can only describe their algorithm mechanisms and call pre-trained model through Pytorch.

BERT is the acronym for "Bidirectional Encoder from Transformers". BERT can use information from both directions simultaneously, while LSTM and RNN can only use information from a single direction [Jacob, 2019].

6.4.2 Input representation:

The input representation as diagram 6-4-1-2. For example, there are only two sentences as input, “my dog is cute” and “he likes playing”. The first thing is adding a special Token [CLS] at the beginning of the first sentence and add a token [SEP] after cute to end the first sentence, also, adding a [SEP] after “ing”. The reason why separate playing as “play” and “ing” is it is a common approach to solve OOV problem. Next, each word is Embedding for three times: token embeddings, positional embeddings and segment Embeddings. For token embeddings: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence. Position embeddings can reflect the position to a vector with low dimension. Segment embeddings can indicate which sentence it is.

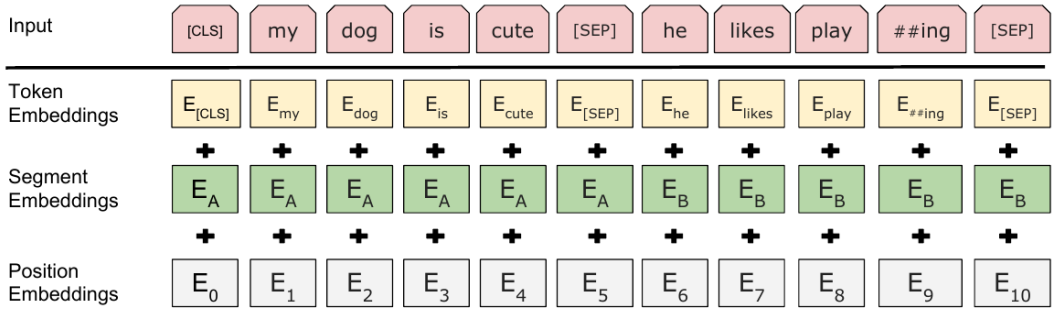

Diagram 6-4-1-2: The input representation of BERT [Jacob et al., 2019]

Otherwise, The BERT model requires a fixed Sequence length, such as 128. If less than it then padding or grab the extra Token to make sure the input is a fixed length Token sequence.

6.4.3 Mask LM

In order to solve the problem of using only one-way information, BERT uses a Mask language model rather than a normal language model. BERT will randomly mask out 15% of the words in the input and replacing them with a [MASK] token, run the entire sequence through the BERT attention based encoder and then predict only the masked words, based on the context provided by the other non-masked words in the sequence. By adjusting the parameters of the model, the accuracy is as large as possible, which is equivalent to the loss function of cross entropy.

However, A special Token will occur when pre-training LM, but it does not occur in the fine-tuning afterwards. Therefore, if a Token is among the 15% of tokens selected, it will be executed randomly in the following way:
1. 80% of the tokens are actually replaced with the token [MASK]. For example, my dog is hairy → my dog is [MASK].
2. 10% of the time tokens are replaced with a random token. For example, my dog is hairy → my dog is apple.
3. 10% of the time tokens are left unchanged. For example, my dog is hairy → my dog is hairy.

BERT doesn't know which word [MASK] was replaced, because any word might be replaced. This forces the model to encode a sentence not to rely too much on the current word, but to "correct" it for its context. For the above instance, the model is encoded using the context "my dog is", where the word “Apple” should be encoded with the word “hairy” rather than “Apple”

6.4.4 Fine-Tuning Process

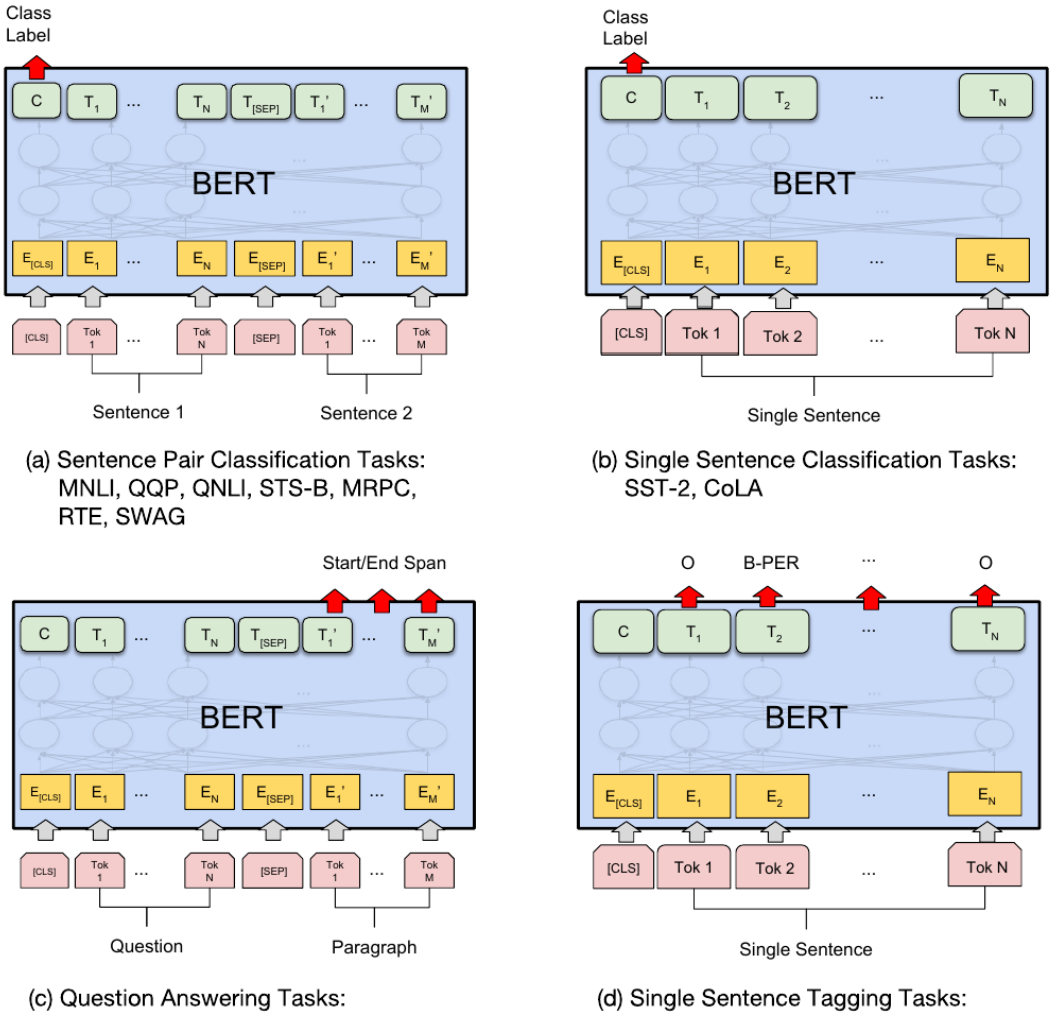

Diagram 6-4-1-3: Process of Fine-Tuning [Jacob et al., 2019]

For the task like calculating similarity between two sentences, the process as diagram 6-4-1-3-a, two sentences correspond to different segments, 0 and 1. Softmax layer was connected with the output of the last layer of the first special Token (CLS) for classification and fine-tuning with the classified data.

For the normal classification task, the process as diagram 6-4-1-3-b, all tokens are in segments, 0 and 1. Softmax layer was connected with the output of the last layer of the first special Token (CLS) for classification and fine-tuning with the classified data.

For the Question Answering Task, input is a question and a paragraph with long answer like diagram 6-4-1-3-c, the output can find out the answer to the question in this paragraph.

For the sequence tagging task, the input is a single sentence like diagram 6-4-1-3-d, every word except [CLS] and [SEP] will have a output tag, then process Fine-Tuning with output tag.

6.4.5 Implementation

Since BERT model cannot be used to unsupervised learning, that means, it is necessary to have a large number of training data to implementation BERT algorithm. However, unfortunately for this project, there is absolutely no training data that can be used to do supervised learning. In this case, I used a pre-trained model by using Pytorch and it has been trained through lecture text data and encapsulated [Derek, 2019].

6.5 Results

In this section, the focus is on the results of the BERT model comparing the output to TextRank model. Since this is absolutely unsupervised learning and there were no training data and test data, there were no other metrics used besides human comparison.

MethodName	TextRank	BERT
doInBackground	1. I tried various methods cancel true in the AsyncTask when an exception is triggered in socketexceptionset tcpClient.	1. time connection attempted fails memory increment kb never released.
	I have put the debug point on return log, statement in doInBackground .. to view th logs	put debug point return log statement doinbackground view th logs.
finish	So next time the user does Monday, the program will ask for 155 lbs instead of the original 135 lbs.I would have that added as a new setting which the user can turn on or off as they wish.Some things that I though of and can be open for discussion Should the weight only increase if the user completed the requested number of reps. From the example above, if the user only did 5 reps of 155, should it still increase the weight of the program or keep it as-is.	android background service limitations.
	What do you see instead.Users expect the content to play from start to finish, despite buffer issues.Instead, the content playback often stops a few seconds before the end of thestory and advances to the next story.	reason need using materialintroview explain possible swipe view holder view pager.
invalidate	find a way to stop it, or invalidate it, or not update it with the old data if the station changed, or something.	referencedir parameter always ignored matter write inside screenshots saved projectdir screenshots folder.
	In the childItem s onclicklistener event, I can t change its background, even I can show a Toast message or set text to another view.	specifically animationupdatelistener uses following

		invalidate refresh view newly calculated animation coords.
notifyDataSetChanged	does move layout manager of dialog to proper position, but does not get adapter select current item.	indicatoradapter getindicatoradapter notifydatasetchanged scrollbar.
	The application s PagerAdapter changed the adapter s contents without calling PagerAdapter#notifyDataSetChan.	make sure adapter calls notifydatasetchanged content changes.
onCreate	This has an extra constraint due to where onCreateDialog is called 3.An example of how to align the foregound view for an Activity that used both a UIViewTypeHandler and a UIFragmentTypeHandler Ideally Fragments onCreateView would have access to the Presenter so any views can be initialised with frame based data.	way change swipe swipe messages along way.
	GetReaders always call AddonTerminal.java getPackageNames always get a whole list of all installedPackages 2 then it does an expensive loop of comparison against 2 strings, that can be 100ms+ If AddonTerminal.getPackagesNames is not thread/process safe, then Getinstalledpackages in ObjectManager can also take extended period of time that s the case, the long operation can easily be pre-emptted and result in large amount of memory being allocated at the same time.	due srp refactors uitypehandlers registered differnt ui types e. views dialogs fragments etc leaves challenges around working fragments lifecycle safe start using presenter.
onDestroy	I ve found this topic https //stackoverflow.com/questions/47531742/startforeground-fail-after-upgrade-to-android-8-1 The error is not systematic and depend on the robospice service if the service is completed when onDestroy is called = no crash, if the service is still running = crash.	android invalid channel service notification.
	I m not 100% that the crash is due to Robospice, but I do not use notification manager in my code, and it always crash when leaving the app with service un background.	got new error users using android android app remoteserviceexception bad notification startforeground java lang runtimeexception invalid channel

		service notification notification channel null.
onDraw	Better alternative would be to use a NavigationDrawer, so I ll look into this.	move invalidate ontouchevent ondraw slider java.
	when i add PolygonDrawable to the VectorLayer Polygon holes cannot be drawn， Is there a way to solve this problem.， Is my style wrong?	remove unnecessary condition ondraw.
onResume	I used MicrophoneInput class to my app, which must dynamically turn on or off microphone due to power consumption.	caststatelistner getting called fragment.
	when fragment starts nothing happen during onresume .But when app goes in background so onstop is being called then coming back to foreground onresume is being called , at that time listner works .	chapter error glsurfaceviewtest java source code.
onStart	Un-comment the part in the onStartCommand function, and you will see it will write to the log each time something gets copied, even if you leave the app.	possible make app native clipboard manager work android q well found possible get clipboard data background app foreground top view.
	Will it be possible to make the app of Native Clipboard Manager work on Android Q well.. For now, I ve found out that it s possible to get the clipboard data in the background, but only if the app is in the foreground with an on-top view .	un comment part onstartcommand function see write log time something gets copied even leave app.
startActivity	Also, do you know how I would change the code to use the front-facing camera.	look post android crop intent many developers calling startactivity intent action com android camera action crop.
	It seems that using the so called System Crop Action com.android.camera.action.CROPis not a wise choice.	however safer bets relying upon undocumented intent action app may exist given user device.
startActivityForRes	I am using the single pick with these lines Intent i = new Intent Action.ACTION_PICK ,	flag activity new task startactivityforresult

ult	startActivityForResult i, 100 , But in the activity result method, the request code has changed and I can t find where is changing.	onactivity.
	Let users sign into multiple accounts at the same time.	let users sign multiple accounts time.
onClick	Is that mean I should use the custom OnClickListener Interface approach if i want to return data from the previous Activity?	multiple run onclicklistener body.
	Hello ï¼š Thank you very much for sharing your results, And saved me a lot of time, Thank you very much againl, but, I found some problems, When I say item listens for an event, When the sliding view has no fingers, It triggers an onClick event directly to the view, So l'd like to make a suggestion, Define a Boolean type variable, in setCanAutoRotation, False in ACTION_DOWN from setCanAutoRotation, true in ACTION_MOVE, The onClick event of the subview in the checkChildView then determines that the Boolean type variable is false, To enter the onItemClick event.	button onclick method construct new ambilwarna dialog show.

Table 6-5-1-1: complete summarization results generated by Textrank and BERT

From two summaries above, it is obvious that TextRank output is better than Bert output in majority of method names according to quality of summaries and the degree of the integrity, although BERT performs not bad for a few method names such as “onStart”. That’s because of three reasons. Firstly, textRank algorithm used extraction summarization and what it did is to select two sentences which have highly connection to other sentences. It is inevitable to have fluent and complete sentences. Second of all, BERT algorithm used abstract to create new text summarization, even though it is state-of-the art, it is also very strict with the text to be summarized. However, issue reports are littered with informal text, such as questions and answers, code related entities and improper symbols. The reasons above mentioned lead to the result that BERT can’t recognize text to run its characteristic mask mechanism. Finally, because lack of labeled data, the pre-trained BERT model used lecture text instead of issue reports, causing the performance for issue reports summarization become worse.

6.5 Conclusion

In this paper, we showcased how selected issue reports can be usefully applied in text summarization. I introduced TextRank algorithm and proposed to do text summarization with BERT algorithm. Although leveraging the most current deep learning NLP model called BERT, there is still no improvement on other dated approaches such as TextRank in the quality of summaries and the degree of integrity because of informal text and improper pre-training text data. Although the performance for BERT model was not perfect, it will provide better service in future when proper training data and formal text are applied. For the future work, it is necessary to conduct human evaluations to better understand their assessment of summarization results quality.

7. Reference

Yang Liu, Mirella Lapata. 2019. Text Summarization with Pretrained Encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *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)*

Aishwarya Jadhav and Vaibhav Rajan. Extractive summarization with swap-net: Sentences and words from alternating pointer networks. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), volume 1, pages 142–151, 2018.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In Thirty-First AAAI Conference on Artificial Intelligence.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. 2017. Attention Is All You Need.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Derek Miller. 2019. Leveraging BERT for Extractive Text Summarization on

Lectures.

Georgios Gousios. The GHTorrent dataset and tool suite. In Proceedings of the 10th Conference on Mining Software Repositories, MSR ’13, pages 233–236, 2013.