CS559 Deep Learning Course Term Project Progress Report

Emre Dogan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
emre.dogan@bilkent.edu.tr

Hamdi Alperen Cetin
Department of Computer Engineering
Bilkent University
Ankara, Turkey
alperen.cetin@bilkent.edu.tr

***Abstract*—This is the progress report of CS559 Deep Learning Course Term Project.**

Our project is about bug traiging. Bug triaging is the process of prioritizing bugs and assigning an appropriate developer to given bugs. The main task in our project is to predict the most appropriate developer to fix a software bug from the a given bug report. This problem can be defined as a classification problem in which textual bug attributes (bug title, description etc.) are inputs and one of the available developers (class labels) is the output. We employ Deep Bidirectional Recurrent Neural Network with Attention(DBRNN-A) approach to accomplish this task.

***Index Terms*—recurrent neural networks, long short-term memory, bug triaging**

I. Introduction

It is an important task to assign an appropriate developer for a given bug report from both developer's and organization's perspectives. A significant amount of time is spent by software developers to understand, identify and fix the bug. A poor developer assignment for a bug report might reduce his/her efficiency. From the organization's perspective, bug fixing time is important as the corresponding bug might block the working of a product/service and cost a large amount of money. For all these reasons, assigning the most appropriate developer for a bug report, one of the primary tasks of bug triaging process, is an important and active research area.

This paper is organized as follows. Section II gives details about the dataset. Section III describes the implementation steps briefly. Section IV mentions the challenges that we faced up to now. Finally, section V states the future work that we are planning to do.

II. Dataset

Our dataset consists of bug reports from three open source systems: Google Chromium, Mozilla Core and Mozilla Firefox. The details of data collecting process are mentioned in the paper [1]. The authors provide the dataset available online, and we plan to use it directly. In total, there are 383,104 bug reports from Google Chromium, 314,388 bug reports from Mozilla Core, and 162,307 bug reports from Mozilla Firefox. All datasets have attributes id, issue id, issue title, reported time, owner, description and status. Chromium dataset has another attribute named type and the other Mozilla datasets have another attribute named resolution. In DBRNN-A model, only owner, title, description and status attributes are used. We shared dataset links and brief explanation about dataset contents in our GitHub Repository.[1]

III. Implemetation

A. Preprocessing

The bug report datasets having title, description, reported time, status and owner are shared online by the authors in json format. Before using the text in the titles and the descriptions of the bug reports, some parts of them need to be removed.

First, URLs, stack traces and the hex codes are removed with the help of regex library, and all characters are changed to lower case. Then, using Stanford NLTK[2] library, words are tokenized. After that, all punctuation and "None" words are deleted. The same process is followed for both open bug reports data and closed bug reports data. At the end, the final vocabulary is created from a set of unique words that occurred for at least k-times in open bug reports data by using Gensim[3] Word2Vec, and the owner information is acquired from the "owner" label in the closed bug reports. First 3 steps in Fig 1 are preprocessings steps.

B. Model Description

Deep Bidirectional Recurrent Neural Network with Attention(DBRNN-A) [1] works in the following way:

First the model learns feature representation of bug reports from untriaged bugs in an unsupervised manner by using long-short term memory(LSTM) cells. Then, attention mechanism is used, because all words in the content may not be useful in the classification task. Also, bidirectional RNN considers the words in both forward and backward direction, and concatenates both representations.

Simple approaches like bag-of-words and n-grams are not sufficient to represent the features of bug reports. The problem with bag-of-words approach is that it loses the ordering of words in the contextual manner and the semantic similarity between synonymous words. On the other hand, n-grams model offer a better bug report representation but it fails to

[1]https://github.com/bilsengroup/bug-triaging-rnn
[2]http://www.nltk.org/index.html
[3]https://radimrehurek.com/project/gensim/

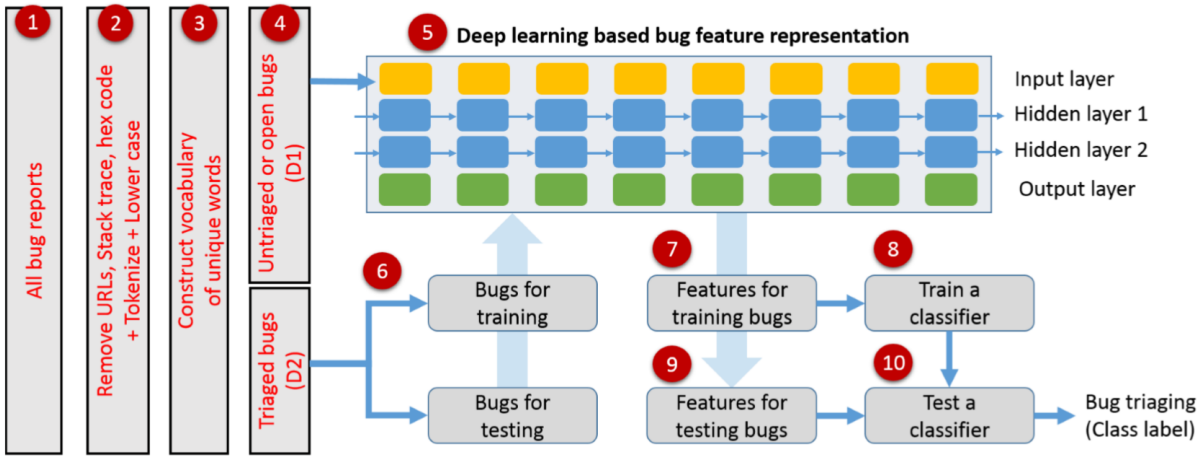

Fig. 1. The flow diagram of the overall proposed algorithm highlighting the important steps. [1]

deal with high dimensionality and sparse data. As these two representations are not good enough to represent a bug report, the authors come up with a new approach to represent bug reports, word2vec embedding, such that disadvantages of bag-of-words and n-gram representations will be prevented.

After learning features from untriaged bugs, traiged bugs are used while training and testing different classifiers such as Naive Bayes(NB) and Support Vector Machines(SVM).

Overall structure of the whole model can be seen in Fig 1.

IV. Current Status and Faced Challenges

Until this point we accomplish preprocessing and word2vec embedding of dataset. While implementing the Keras model, we encounter some problems about implementing the soft attention layer. Detailed information about the job we done are as following.

- **Preprocessing Part:**
  We have datasets consisting a of bug report corpus having title, description, reported time, status and owner(assigned developer) attributes. But when we investigate these bug reports, it is observed that there are lots of unstructured text parts within these bug reports such as URLs, stack trace, hex-code, and the code snippets. We removed these contents from all reports. Then, by gathering all words occured for at least k-times (k is optional) in the corpus, we created our dictionary to be fed to Attention Based Bidirectional Recurrent Neural Network Model. We completed the preprocessing part succesfully and extracted the word2vec embedding and the corresponding dictionary.

- **Soft Attention Layer Implementation:**
  As described in the model description, we use an attention based bidirectional RNN model. We have implemented a bidirectional RNN model in Keras. But, there is not an attention layer implementation available in Keras. For this reason, we check some attention layer implementations available in Github. When we adapt and integrate attention label to the RNN model, we will be able to begin our experiments and advance with the proposed future work. We are not able to get results for now, because we have some dimensionality problems when we try to use attetion implementations.

V. Future Work

Our brief plan for this term project will be in the following order:

1) We will complete the whole implementation by using Keras(Tensorflow). We completed bidirectional RNN model. As stated in Section IV, we are currently trying to adapt an external attention model to our RNN model.

2) As stated in our proposal report, we are planning to apply transfer learning between different datasets. Beyond this, we want to apply ensemble learning by using models trained with different datasets. This part is important for us. Because all bug reports have title and description parts, and we want see if transfer learning is possible between different software projects or not.

3) In the inspired paper, authors suggest some future works. One of them is to predict whether a bug will be fixed or not by using DBRNN-A approach and status attribute of bug reports. We intend to put it into practice after building DBRNN-A architecture.

References

[1] Mani, Senthil, Anush Sankaran, and Rahul Aralikatte. "Deeptriage: Exploring the effectiveness of deep learning for bug triaging." Proceedings of the ACM India Joint International Conference on Data Science and Management of Data. ACM, 2019.