Classifying Bug Reports into Bugs and Non-bugs Using LSTM

Hanmin Qin
Peking University
qinhanmin2005@sina.com

Xin Sun
Peking University
sunx5@pku.edu.cn

ABSTRACT

Studies have found that significant amount of bug reports are misclassified between bugs and non-bugs, which inevitably affects relevant studies, e.g., bug prediction. Manually classifying bug reports helps reduce the noise but is often time-consuming. To ease the problem, we propose a bug classification method based on Long Short-Term Memory (LSTM), a typical recurrent neural network which is widely used in text classification tasks. Our method outperforms existing topic-based method and n-gram IDF-based method on four datasets from three popular JAVA open source projects. We believe our work can assist developers and researches to classify bug reports and identify misclassified bug reports. Datasets and scripts used in this work are provided on GitHub[1] for others to reproduce and further improve our study.

CCS CONCEPTS

• **Software and its engineering → Software development process management**;

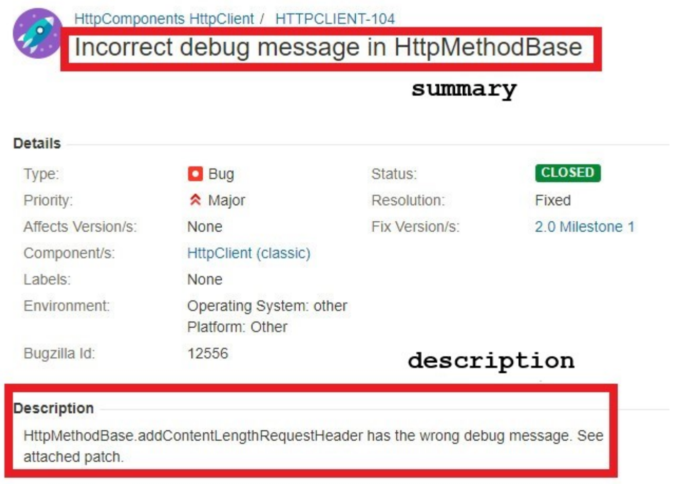
KEYWORDS

bug classification, LSTM

ACM Reference Format:
Hanmin Qin and Xin Sun. 2018. Classifying Bug Reports into Bugs and Non-bugs Using LSTM. In *The Tenth Asia-Pacific Symposium on Internetware (Internetware '18), September 16, 2018, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3275219.3275239

1 INTRODUCTION

Bug reports are essential software artifacts used to record bugs, feature requests and other software activities. Figure 1 shows a typical bug report from HttpClient. It characterizes current issue through unstructured data (e.g., summary and description) and structured data (e.g., priority and components). Bug reports are used for various software development tasks [10], such as bug assignment and bug localization. However, previous studies show that problems of bug reports can severely impact the quality of studies and tools that leverage such data [3]. Misclassification of bug reports is one of the series quality problems of bug reports [1, 3], i.e., significant amount of bug reports are misclassified between bugs and non-bugs. These errors happen mostly due to reporter's misunderstanding and the complicating nature of issue tracking system, which manages bugs and non-bugs together.


HttpComponents HttpClient / HTTPCLIENT-104
Incorrect debug message in HttpMethodBase

Details

Type:	Bug	Status:	CLOSED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	2.0 Milestone 1
Component/s:	HttpClient (classic)		
Labels:	None		
Environment:	Operating System: other Platform: Other		
Bugzilla Id:	12556		

Description
HttpMethodBase.addContentLengthRequestHeader has the wrong debug message. See attached patch.

Figure 1: Example of a Bug Report

Distinguishing bugs from non-bugs is a challenging task since each report requires an individual inspection to identify the incorrectness. Some studies [1, 3] had to spend part of their work schedule on manually classifying bug reports. Herzig et al. [3] took over 700 hours to manually classify 7401 bug reports from 5 JAVA open source projects and found that every third bug report is no bug report. According to their study, manually classifying bug reports helps reduce the noise but is often time-consuming.

To ease the problem, several studies tried to classify bug reports automatically. Existing methods include method based on bag-of-words [1], method based on n-gram idf [9] and method based on topic model [6, 8]. However, these methods are not good at capturing the meaning of the words and the structure of the sentences.

Motivated by the recent success of recurrent neural networks in text classification, we propose a bug classification method based on Long Short-Term Memory (LSTM) [5]. First, we preprocess the report and map words to vectors of real numbers (i.e, word embedding). Then, we use LSTM to compute the vector representation of the report from the vector representation of words it contains. Finally, we classify the report into bug or non-bug with a softmax layer. Our method is capable of better capturing the meaning of words through word embedding and better capturing the structure of the sentences through LSTM. Experiments on four datasets from three popular JAVA open source projects show that our method outperforms existing topic-based method and n-gram IDF-based method. We believe our work can assist developers and researches to classify bug reports and identify misclassified bug reports.

[1]https://github.com/qinhanmin2014/bug-classification-LSTM/

Internetware '18, September 16, 2018, Beijing, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6590-1/18/09…$15.00
https://doi.org/10.1145/3275219.3275239

The contribution of this paper can be summarized as follows:

- We propose a bug classification method based on Long Short-Term Memory (LSTM), a typical recurrent neural network which is widely used in text classification tasks.
- We compare our method with existing topic-based method and n-gram IDF-based method.

The rest of this paper is organized as follows. We present related work in Section 2 and introduce our method in Section 3. Section 4 introduces our experiments and Section 5 includes the results of these experiments. We discuss the limitation of our work in Section 6 and conclude in Section 7.

2 RELATED WORK

Bug reports are essential software artifacts and a considerable amount of researches have been carried out on bug-report analysis [10]. Problems of bug reports, such as misclassification of bug reports, can severely impact the quality of tools and studies that leverage such data [3]. Antoniol et al. [1] first addressed misclassification problem of bug reports and found that less than half of bug reports are actually related to bugs. Herzig et al. [3] manually classified 7401 bug reports from 5 JAVA open source projects and found that every third bug report is no bug report. They also found significant impact of misclassification problem on previous studies, such as bug prediction.

To ease the problem, several studies tried to classify bug reports automatically. Antoniol et al. [1] proposed automated classification method based on bag-of-words model. Terdchanakul et al. [9] further proposed a method based on N-gram IDF (Inverse Document Frequency). Pingclasai et al. [8] proposed a method based on topic model: Latent Dirichlet Allocation (LDA) and Limsettho et al. [6] proposed a method based on another topic model: Hierarchical Dirichlet Process (HDP). Based on similar idea, Chawla et al. [2] proposed a method based on fuzzy set theory. Zhou et al. [11] utilized both structured data and unstructured data in the bug report. They proposed a hybrid approach by combining text mining and data mining techniques using data grafting techniques. In this paper, we propose a method based on Long Short-Term Memory (LSTM), a typical recurrent neural network which is widely used in text classification tasks.

3 METHODOLOGY

Figure 2 shows the overview of our method based on Long Short-Term Memory (LSTM). First, we preprocess the report and map words to vectors of real numbers (i.e, word embedding). Then, we use LSTM to compute the vector representation of the report from the vector representation of words it contains. Finally, we classify the report into bug or non-bug with a softmax layer. We implement our method with TensorFlow[2].

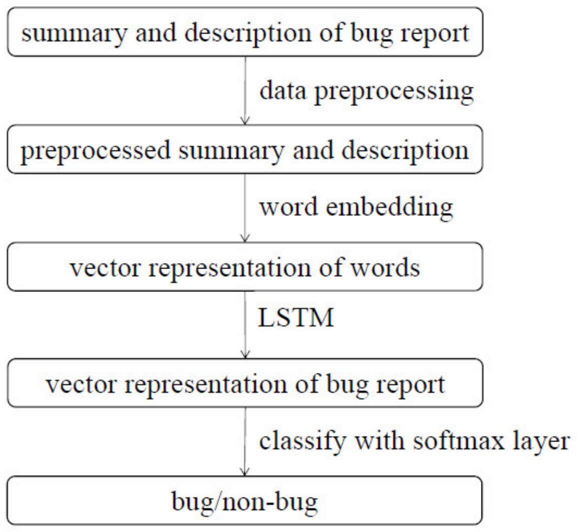

summary and description of bug report
data preprocessing
preprocessed summary and description
word embedding
vector representation of words
LSTM
vector representation of bug report
classify with softmax layer
bug/non-bug

Figure 2: Overview of Our Methodology

3.1 Data Preprocessing

We use the summary and description of bug reports (See Figure 1 for detail) as the input of our model. Summary usually provides an one-sentence overview of the report and description usually contains more detailed information. These two fields are generally available when a bug report is submitted. Note that a few reports do not have description. For these reports, we only use the summary as the input of our model.

We tokenize summary and description using the recommended word tokenizer provided by NLTK [3]. In order to reduce noise, we replace some special words with certain tokens. For example, we replace all the numbers with a special token <*NUM*>. Since we feed all the reports to the same LSTM, we need to choose a specific length $MAXLEN$. For longer reports, we only take the first $MAXLEN$ words. For shorter reports, we use a special token to pad the report to $MAXLEN$. In our experiment, we set the length to 100 according to the median length of the reports.[4]

3.2 Word Embedding

The basic idea of word embedding is to represent each word as a low dimensional, continuous and real-valued vector. It allows words with similar meaning to have a similar representation, thus is capable of capturing the meaning of words. All the word vectors are then stacked in a word embedding matrix $M_e \in \mathbb{R}^{d*|V|}$, where d is the dimension of word vector and $|V|$ is vocabulary size, to serve as the input of LSTM.

We use two methods to construct the word embedding matrix. One is to randomly initialize the word embedding matrix, i.e., rely on LSTM to differentiate the meaning of words by looking at the data. In this work, we randomly initialize 100-dimensional uniformly distributed vectors to serve as the initial value of word embedding matrix. The other is to use pre-trained word embedding. In this work, we use pre-trained word embedding provided by Google [7], which contains 300-dimensional vectors for 3 million words and phrases trained using news articles.

[2]https://www.tensorflow.org/

[3]http://www.nltk.org/

[4]The median length of the four datasets (i.e., Jackrabbit, Lucene, HttpClient and Cross Project) are 68, 83, 82 and 76

3.3 LSTM

We use LSTM to compute the vector representation of a report from the vector representation of words it contains (i.e., the word embedding matrix). LSTM [5] is a typical recurrent neural network (RNN). In recurrent neural networks, predictions are made sequentially (e.g., for the vector representation of each word), and the hidden layer from one prediction is fed to the hidden layer of the next prediction, making it capable of capturing the structure of the sentences. LSTM add additional factors to a traditional RNN that give it more of a fine-grained control over memory. These factors control 1) how much the current input matters in creating the new memory, 2) how much the previous memories matters in creating the new memory, and 3) what parts of the memory are important in generating the output. Compared with traditional RNN, LSTM is good at handling long term dependency (i.e., a late output depends on a very early input).

We use dropout [4] to reduce over-fitting. The basic idea is to randomly drop units (along with their connections) from the neural network during training. In our experiment, the dropout rate is set to 0.5, which corresponds to the maximum amount of regularization. Finally, we collect the output vector of each word and average these vectors to serve as the vector representation of the report. Note that there are other ways to get the vector representation of the report from LSTM (e.g., use the last output vector). We choose this way because it preforms better.

3.4 Classify with Softmax Layer

After obtaining the vector representation of the reports, we feed it to a linear layer whose output length is class number. Finally, we obtain the probability of different classes (i.e., bug and non-bug) with a softmax layer, which is capable of transforming a vector of real numbers to a probability distribution. The class with higher probability is regarded as the predicted class. We use cross entropy as the loss function and trained the model with Adam optimizer.

4 EXPERIMENTAL DESIGN

4.1 DataSets

We conduct our experiment based on the datasets from Herzig et al. [3][5]. Their datasets provide bug report ID, original type and corrected type of 7401 bug reports from 5 JAVA open source projects. We focus on the 3 projects which use JIRA as their issue tracking systems, i.e., Jackrabbit, Lucene and HttpClient. We use JIRA REST APIs [6] to obtain the content of the bug report according to bug report ID and regard corrected type as type of the bug report. Note that the original datasets classify bug into six types. In this work, we regard *BUG* as bugs and other types as non-bugs. Inspired by Pannavat et al. [9], we combine all the reports from these three projects and create a new dataset, i.e., the cross project dataset. Table 1 shows detailed information about the four datasets we use.

4.2 Evaluation

We split the datasets into training set and testing set based on when these bug reports are created, i.e., the oldest 90% of bug reports are utilized as training set and the newest 10% bug reports are utilized as testing set. Compared with cross validation, we include time factor when splitting the datasets, thus make the scenario more practical. Note that when training LSTM, we randomly take part of the training set (i.e., 5% for cross project dataset and 10% for other datasets) as the validation set, so that we can stop training when the error on the validation dataset increases to avoid over-fitting.

Table 1: Detailed Information about the Datasets

	# of bug reports	# of bugs	# of non-bugs
Jackrabbit	2402	938	1464
Lucene	2443	697	1746
HttpClient	746	305	441
Cross Project	5591	1940	3651

We use F-score to evaluate the performance of different methods. F-score is widely used to measure document classification performance and is defined as:

$$precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{1}$$

$$recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2}$$

$$F - score = \frac{2 * precision * recall}{precision + recall} \tag{3}$$

We implement LSTM-based method with and without pre-trained word embedding. We compare our LSTM-based method with existing topic-based method and N-gram IDF-based method, because these two recent methods outperform previous studies.

- **Topic-based method**: Topic-based method is from Pingclasai et al. [8]. First, the paper used LDA (Latent Dirichlet Allocation) to extract topics from processed bug reports. Then, the paper treated whether these topics appear as features of bug reports and fed these features to a classifier. We carefully follow the paper to reproduce their work on our datasets. As is recommended in the paper, we set number of topics to 50 and use Naive Bayes as the classifier. In our experiment, LDA is implemented with gensim[7] and Naive Bayse is implemented with scikit-learn[8].
- **N-gram IDF-based method**: N-gram IDF-based method is from Terdchanakul et al. [9]. First, the paper applied N-gram IDF to processed bug reports. Then, the paper filtered out less important N-grams with feature selection algorithms and fed the remaining features to a classifier. Since we use the same datasets and the same method to split the datasets, we take the result directly from the paper. Note that the paper use Logistic Regression and Random Forest as their classifiers. Since two classifiers obtain similar results, we choose Random Forest to serve as our baseline.

5 RESULTS

Table 2 shows the F-score of the three different methods, i.e., topic-based method, n-gram IDF-based method, LSTM-based method

[5]http://www.st.cs.uni-saarland.de/softevo/bugclassify/
[6]https://developer.atlassian.com/server/jira/platform/rest-apis/
[7]https://radimrehurek.com/gensim/
[8]http://scikit-learn.org/stable/

Table 2: F-score of Three Different Methods on Four Datasets

	Jackrabbit	Lucene	HttpClient	Cross Project
topic-based method [8]	0.591	0.420	0.675	0.531
n-gram IDF-based method [9]	0.628	0.685	0.673	0.674
LSTM-based method (without pre-trained word embedding)	**0.771**	0.712	**0.757**	0.734
LSTM-based method (with pre-trained word embedding)	0.752	**0.717**	0.750	**0.746**

without pre-trained word embedding, and LSTM-based method with pre-trained word embedding. Since we get slightly different result on every run while applying randomized algorithms to the same problem instance, we repeat our experiment for 5 times and regard the average F-score as the final F-score.

We can find that our LSTM-based method outperforms topic-based method and N-gram IDF-based method. Compared with n-gram IDF-based method (which performs better than topic-based method on all the four dataasets), our LSTM-based method without word embedding increase the performance from 0.628 to 0.771 (i.e., 22.8% improvement) for Jackrabbit dataset, from 0.685 to 0.712 (i.e., 3.9% improvement) for Lucene dataset, from 0.673 to 0.757 (i.e., 12.5% improvement) for HttpClient dataset, from 0.674 to 0.734 (i.e., 8.6% improvement) for Cross Project dataset, indicating the effectiveness of our method. Considering that we take part of the training set as validation set when training LSTM, our model actually achieve better result with less training data.

We also implement LSTM-based method with pre-trained word embedding provided by Google. We expect to get better result since pre-trained word embeddings are trained on large corpus, thus are generally better at capturing the meaning of words. However, we get very similar results compared with LSTM-based method without pre-trained word embedding (See Table 2). Possible reason might be that this word embedding is trained using news articles, thus do not suit bug reports very well.

6 THREATS TO VALIDITY

Datasets depend on previous study. This introduces construct validity. Since significant amout of bug reports are misclassified, We have to conduct our experiments based on manually labeled datasets from previous study. Although the original paper proposed a set of rules to support their classification, errors might still occur since these rules along with the labeling process depend on an individual perspective.

Fail to obtain the scripts of related work. This introduces construct validity. We need to compare our method with existing methods to prove that our method is effective, but many related studies do not provide their scripts. To ease the problem, we use the same datasets and take the result directly from the original paper when comparing with n-gram IDF-based method. When reproducing the topic-based method, we tried our best to follow the original paper, but still cannot promise that our implementation is exactly the same as the original paper.

Study subjects limited to JAVA open-source projects using JIRA. This introduces external validity. We obtain out datasets from 3 JAVA open source projects which use JIRA as their issue issue tracking systems. The results might not be generalized to projects written in other languages or using other issue tracking systems.

7 CONCLUSION

According to previous studies, significant amount of bug reports are misclassified between bugs and non-bugs, which impacts the quality of relevant studies and tools. However, manually classifying bug reports is often time-consuming. Motivated by the recent success of recurrent neural networks in text classification, we propose a bug classification method based on LSTM (Long Short-Term Memory), which is capable of better capturing the meaning of the words and the structure of the sentences. Experiments on four datasets from three popular JAVA open source projects show that our method outperforms existing topic-based method and N-gram IDF-based method. We provide our datasets and scripts for others to reproduce and further improve our study.

In the future, we plan to develop tools based on our method to help developers and researchers classify bug reports and identify misclassified bug reports.

REFERENCES

[1] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, and Foutse Khomh. 2008. Is it a bug or an enhancement?:a text-based approach to classify change requests. In *Conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada.* 304–318.

[2] Indu Chawla and Sandeep K. Singh. 2015. An Automated approach for Bug Categorization using Fuzzy Logic. In *India Software Engineering Conference.* 90–99.

[3] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *International Conference on Software Engineering.* 392–401.

[4] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *Computer Science* 3, 4 (2012), pÃągs. 212–223.

[5] Sepp Hochreiter and JÃijrgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[6] Nachai Limsettho, Hideaki Hata, and Ken Ichi Matsumoto. 2014. *Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification.* 1–6 pages.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems* 26 (2013), 3111–3119.

[8] Natthakul Pingclasai, Hideaki Hata, and Ken Ichi Matsumoto. 2014. Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling. In *Software Engineering Conference.* 13–18.

[9] Pannavat Terdchanakul, Hideaki Hata, Passakorn Phannachitta, and Kenichi Matsumoto. 2017. Bug or Not? Bug Report Classification Using N-Gram IDF. (2017), 534–538.

[10] Jie Zhang, Xiao Yin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. 2015. A survey on bug-report analysis. *Science China(Information Sciences)* 58, 2 (2015), 1–24.

[11] Yu Zhou, Yanxiang Tong, Ruihang Gu, and H Gall. 2016. Combining Text Mining and Data Mining for Bug Report Classification. In *IEEE International Conference on Software Maintenance and Evolution.* 311–320.