Bug or Not Bug? Labeling Issue Reports via User Reviews for Mobile Apps

Haoming Li[1], Tao Zhang[1,2*], Ziyuan Wang[3]
[1]College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China
[2]Key Laboratory of Network Assessment Technology, Institute of Information Engineering, CAS, Bejing 100093, China
[3]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
{heulaoyoutiao, cstzhang}@hrbeu.edu.cn, wangziyuan@njupt.edu.cn
*Corresponding author: Tao Zhang

Abstract—**A great number of mobile applications (apps) have been released to the market. Therefore, software maintenance for these apps become an important and challenging task. For each app, developers usually submit issue reports to report the bugs, the features, the questions, and other changes appearing in it. In the process of software maintenance, developers refer to the corresponding labels to decide which one should be fixed first. If the label of an issue report is "bug" which means the report describes a serious error, developers should fix the bug first. Otherwise, if the label is not "bug" (*e.g.*, feature or question), developers can resolve it later. However, according to our investigation, 36.7% of issue reports in top-10 popular mobile apps are not labeled. In other words, there are not any labels in them. It is difficult for developers to decide which issue should be resolved preferentially. To resolve this problem, we propose a method to verify whether an issue report describes a bug or not by using user reviews. Developers usually extract useful information from user reviews to maintain mobile apps. In this work, we utilize tf • idf, Word2Vec, and Microsoft Concept Graph (MCG) to compute the textual similarity between issue reports and user reviews related to real bug in order to find the issue reports which describe the real bugs. As a result, our approach with Word2Vec performs the best among three similarity metrics.**

Keywords-issue report; user reviews; simialrity metrics; mobile apps; software maintenance

I. INTRODUCTION

Recently, the number of mobile devices such as smartphones and tablet computers have been produced for a wider group of people. These mobile devices result in the increase of the number of mobile applications (apps). Therefore, maintaining mobile apps is becoming important and challenging [1]. Fixing bugs is a core task in software maintenance activities [2]. Once a bug is found, a developer can upload an issue report to describe this bug. The detailed information can help developers fix it. However, we find that 36.7% of issue reports in top-10 popular mobile apps were not labeled as "bug" or others. In this situation, developers should verify whether these issue reports describe the real faults or not. For a large number of issue reports, it is a time-consuming work. Thus, it is necessary to develop an approach to automatically label the issue reports instead of manual versification.

In online app stores (*e.g.*, Google Play Store, Apple Store), users can evaluate each app by using scores and post their reviews. These reviews are free-form text that may include important information such as bugs that need to be fixed by developers. The buggy user reviews can guide developers resolve bugs appearing in apps. Therefore, the bugs appearing in user reviews may be very close to the real faults reported in the issue report so that they can help to verify whether an issue report describes a bug or not. To the best of our knowledge, there was no any work to study the relationship between user reviews and issue reports in mobile apps.

In this paper, we propose an automated labeling approach to verify whether an issue report describes a bug or not. First, according to the suggestion mentioned in the literature [3], we choose user reviews that have less than 3 stars or lower because the reviews with few number of stars have high probability of describing bugs. Second, we use SURF [4], a popular review analysis tool, to automatically classify these user reviews into five categories: information giving, information seeking, feature request, *problem discovery*, and others. Next, we utilize natural language processing techniques to pre-process the user reviews in the category-*problem discovery* to build an index set. Third, we compute the similarity scores between each candidate issue report and the index set by using three popular similarity metrics that include tf•idf [5], Word2Vec [6], and Microsoft Concept Graph (MCG) [7]. If the similarity is more than a threshold, the issue report is labeled as "bug". Otherwise, the issue report does not describe a software fault.

We perform experiments on user reviews and issue report selected from 10 open source mobile apps on GitHub. As a result, the average F1 scores of our approach using tf•idf, Word2Vec, and MCG achieve 56.1%, 61.4%, and 58.9%, respectively when the best threshold is set for each metric. The result denotes that our approach with Word2Vec performs the best among three similarity metrics.

We summarize the contributions of our work as follows:

- We *first* propose an approach to automatically label the issue reports with "bug" by computing the similarity scores between the issue reports and buggy user reviews.

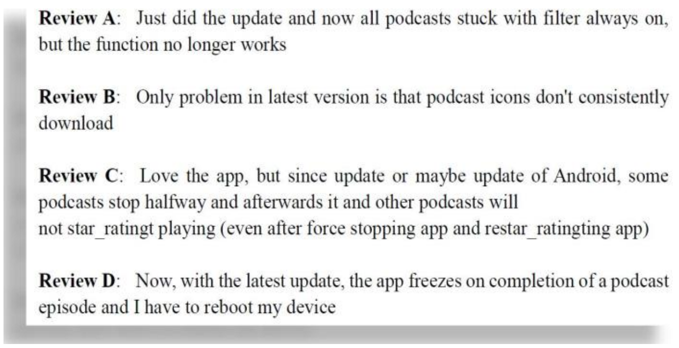
DOI reference number: 10.18293/SEKE2018-059

- We perform our approach on top-10 popular mobile apps. The result shown that our approach with Word2Vec performs the best among three similarity metrics.

Roadmap. Section 2 introduce the background and motivation of our work. In Section 3, we detail the proposed automated labeling approach. Section 4 presents the experimental results. In Section 5, we show the limitations of this work and corresponding solutions. Section 6 introduces the related work. Section 7 concludes the paper and introduce the future work.

II. BACKGROUND AND MOTIVATION

For each app, users can input free-text to comment it. These comments are called user reviews. Since some user reviews describe the real bugs, they can help us label the issue reports as "bug" or not. Fig. 1 shows the examples of four reviews in AntennaPod, which are collected from Google Play Store.



Review A: Just did the update and now all podcasts stuck with filter always on, but the function no longer works

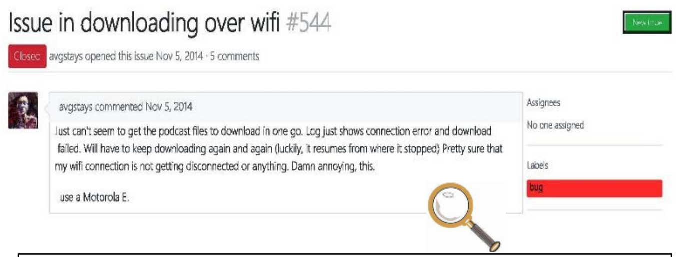
Review B: Only problem in latest version is that podcast icons don't consistently download

Review C: Love the app, but since update or maybe update of Android, some podcasts stop halfway and afterwards it and other podcasts will not star_ratingt playing (even after force stopping app and restar_ratingting app)

Review D: Now, with the latest update, the app freezes on completion of a podcast episode and I have to reboot my device

Fig. 1 User reviews in AntennaPod

From these user reviews, we note that users find the running problems of podcasts due to the update of the versions. They post their comments in Google Play Store and give the relatively low scores (*i.e.*, less than 4 stars). In fact, this is a bug appearing in AntennaPod. An issue report also describes it, we show it in Fig. 2.



Issue in downloading over wifi #544

Closed avgstays opened this issue Nov 5, 2014 · 5 comments

avgstays commented Nov 5, 2014

Just can't seem to get the podcast files to download in one go. Log just shows connection error and download failed. Will have to keep downloading again and again (luckily, it resumes from where it stopped) Pretty sure that my wifi connection is not getting disconnected or anything. Damn annoying, this.

use a Motorola E.

Assignees: No one assigned

Labels: bug

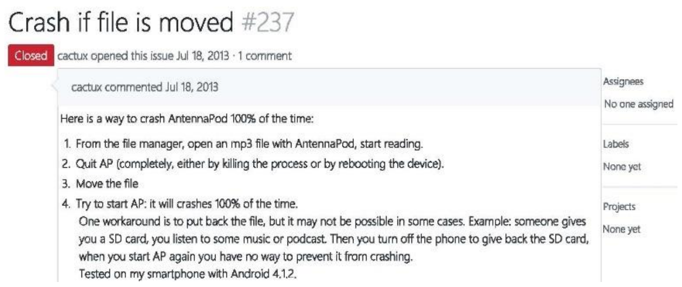
Just can't seem to get the podcast files to download in one go. Log just shows connection error and download failed. Will have to keep downloading again and again (luckily, it resumes from where it stopped) Pretty sure that my wifi connection is not getting disconnected or anything. Damn annoying, this. I use a Motorola E.

Fig. 2 An issue report with the label- "bug"

We note that the developer reported a problem when he downloaded podcast files. The bug is also caused by the version update. It is similar to the problems described in four reviews shown in Fig. 1, especially for Review B.

The label of the issue report-#544 is "bug", therefore, the user reviews which describe the real faults have the close-knit relations with the issue reports which are labeled by "bug".

According to our investigation for issue reports in top-10 popular mobile apps, we find that 36.7% of issue reports are not labeled (See Table I). In Fig. 3, we show an example of unlabeled issue report.



Crash if file is moved #237

Closed cactux opened this issue Jul 18, 2013 · 1 comment

cactux commented Jul 18, 2013

Here is a way to crash AntennaPod 100% of the time:

1. From the file manager, open an mp3 file with AntennaPod, start reading.
2. Quit AP (completely, either by killing the process or by rebooting the device).
3. Move the file
4. Try to start AP: it will crashes 100% of the time.
   One workaround is to put back the file, but it may not be possible in some cases. Example: someone gives you a SD card, you listen to some music or podcast. Then you turn off the phone to give back the SD card, when you start AP again you have no way to prevent it from crashing.
   Tested on my smartphone with Android 4.1.2.

Assignees: No one assigned

Labels: None yet

Projects: None yet

Fig. 3 An example of unlabeled issue report

In Fig. 3, we can see that the reporter did not label this issue report. In such a situation, a bug fixer should verify whether the issue report describes a bug or not so that he or she can decide the priority of resolving the issue. Obviously, this is a time-consuming task when he or she needs to resolve the increasing number of unlabeled issue reports.

When we carefully read the issue report-#237, note that the report describes a crash problem appearing in AntennaPod. In the other words, this is a bug. Thus, it should be fixed first. By analyzing the relationship between the user reviews shown in Fig. 1 and the issue report-#544 shown in Fig. 2, we think that buggy reviews can help to verify whether an issue report describes a bug or not. In the following sections, we introduce the details of the method and the corresponding experiment.

III. RELATED WORK

A. *Software maintenance for mobile apps*

Software maintenance for mobile apps become an important task due to an increasing number of mobile apps. However, only a few research teams study this problem. Syer et al. [12] analyzed 15 most popular open source Android apps, and they found that the "best practices" of existing desktop software development cannot be utilized for mobile apps due to the different features. Bhattacharya et al. [13] executed an empirical analysis of issue reports and bug fixing in open source Android apps. They analyzed the bug-fixing process and the quality of issue reports. Zhou et al. [14] conducted a cross-platform analysis of bugs and bug-fixing process in open source projects of different platforms such as desktop, Android, and IOS. They analyzed the different features such as fixing time and severity of bug-fixing process in these different platforms.

These studies on empirical analysis of issue reports and bug fixing process of mobile apps give us the inspiration for beginning this work. In our work, we not only analyze issue reports, but also analyze user reviews. In addition, we utilize user reviews to label the issue reports which describe the real software bugs.

B. User review analysis

In online app stores such as Google Play Store, Apple App Store, and Windows Phone App Store, users can rate the apps by selecting the stars from 1 (the lowest rating level) to 5 (the highest rating level) and inputting the reviews. These reviews describe users' impressions, experience, and preference degree. Therefore, they can be used by developers as a feedback to facilitate the process of software maintenance. Some studies focus on user review analysis to extract important information. Palomba et al. [15] traced informative crowd reviews onto source code change, and use this relation to analyze the impact of reviews on the development process. Ciurumelea et al. [16] analyzed the reviews and classify them. They also recommended for a particular review what are the source code files that need to be modified to hander the issue. Genc-Nayebi and Abran [17] presented the proposed solutions for mining online opinions in app store user reviews. Chen et al. [18] proposed an approach to identify attackers of collusive promotion groups in an app store by exploiting the unusual ranking change patterns from user reviews.

In our work, we not only analyze and classify user reviews, but also utilize the relation between user reviews and issue reports to automatically label the unlabeled reports as "bug" or "not bug".

IV. METHOD

A. Framework

In this paper, we propose an automated labeling approach to verify whether an issue report describes a real bug or not. Fig. 4 shows the framework of this method.

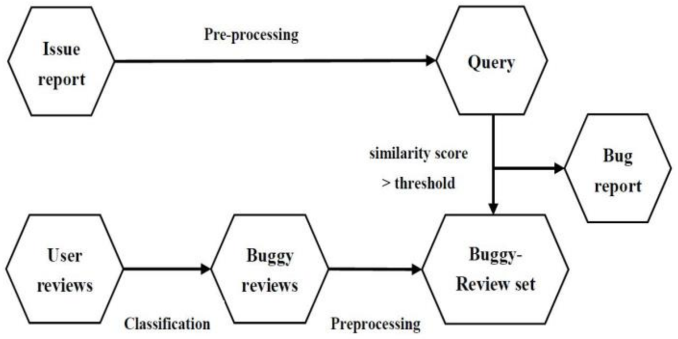


Fig. 4 The framework of automated labeling approach

From this figure, we first use Natural Language Processing (NLP) techniques to pre-process the issue reports in our data sets. These issue reports are treated as queries to be labeled. Second, we classify user reviews into five categories: information giving, information seeking, feature request, *problem discovery*, and others by using an automated review analysis tool. We extract the user reviews in *problem discovery* as buggy reviews according to the suggestion proposed in the literature [3]. Third, we also preprocess these buggy reviews to produce the buggy-review set. Finally, we compute the textual similarity between each query and the buggy-review set by using three similarity metrics that include tf•idf, Word2Vec, and MCG. If the similarity score is more than the threshold, we label this issue report to "bug".

In the following subsections, we show the details of each step.

B. Document Preprocessing

As a first step, we preprocess the issue reports in our data set by using NLP technologies [8]. We use the python libraries NLTK[1] and TEXTBLOB[2] to implement the following steps:

- **Tokenization**: an issue report is divided into a bag of words (*i.e.*, tokens), which can be used to compute the textual similarity.
- **Stop word removal**: Some stop words such as "the", "a", "are" are common words that are frequent in written English. These words cannot provide more semantic information. Thus, they should be removed according to the list of WordNet English stop words.
- **Stemming**: the words should be transformed to their basic forms (*i.e.*, stems) in order to keep the high accuracy when we compute the textual similarity. For example, "providing" is changed to "provide", and "faults" is changed to "fault".
- **Nouns and verbs filtering**: we adopt a part of speech (POS) tagging classification to identify the nouns and verbs from issue reports. Since these words are the most representative, they are considered to compute the textual similarity scores.

C. Review classification

In order to remove the uninformative reviews and find buggy reviews, we adopt SURF [4], a state-of-the-art review analysis tool, to classify them into five categories: Information Giving, Information Seeking, Feature Request, Problem Discovery, and Others. Because we focus on buggy reviews, we only collect user reviews in the category **Problem Discovery** to label the issue reports as real bugs.

To verify whether the classification is acceptable, we randomly select 20% of user reviews in the category **Problem Discovery**. The first author and one research assistant (RA)-Mr. Jiachi Chen from the Hong Kong Polytechnic University are responsible for checking whether each user review describes a real fault. The selected user reviews are equally divided into two groups. Each person is invited to check one group. In order to reduce the possible bias, two persons exchange their data to execute the verification again. The corresponding author can make a final decision when the verification results are inconsistent. As a result, we get the accuracy of 91.4%. Therefore, the classification results are acceptable.

D. Structuring buggy-review set

When extracting the user reviews in the category P**roblem Discovery**, we preprocess these reviews using the same

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

[2] http://textblob.readthedocs.io/en/dev/

approach described in Section Ⅲ.B. Then they are grouped into buggy-review set to be used to compute the similarity scores with issue reports.

E. Bug report verification

In our work, we propose an automated labeling approach to verify whether an issue report is a bug or not. To implement this purpose, we adopt buggy user reviews to verify bug reports by computing the textual similarity between issue reports and the buggy-review set via three similarity metrics that include tf•idf, Word2Vect, and MCG. These metrics are introduced to transfer the documents to different kinds of vectors so that they can be input into cosine similarity function [9] to compute the similarity scores. These metrics are presented as follows:

tf•idf: it is a popular metric to represent documents as vectors of words. The value for each word is its tf-idf weight which is defined by:

$$\text{tf} - \text{idf weight} = tf_{t,d} \times log\frac{N}{n_t}, \quad (1)$$

where $tf_{t,d}$ is an appearing frequency of term *t* in the document *d*. $log\frac{N}{n_t}$ is the inverse document frequency which is a measure of how much information the word provides. N is the total number of documents while n_t is the number of documents which contain term *t*.

When we get all words' tf-idf weights, a document can be transferred to a vector of the tf-idf weights. Thus, we can use the cosine similarity function to compute the textual similarity between an issue report IR_i and buggy-review set *BRS*. It is defined by:

$$\text{sim}(IR_i, \text{BRS}) = \frac{\sum_{k=1}^{n}\omega_{ki}\omega_k}{\sqrt{\sum_{k=1}^{n}\omega_{ki}^2}\times\sqrt{\sum_{k=1}^{n}\omega_k^2}}, \quad (2)$$

where ω_{ki} and ω_k denote the weight of k^{th} word in IR_i and *BRS*, respectively. They are computed by *tf-idf weight* (see formula (1)).

Word2Vec: it maps a word into semantic word embedding. A large corpus of text can be transferred to a vector space, and each unique word in the corpus being assigned a corresponding vector in the space. We utilize Word2Vec with the skip-gram model [10]. In k dimensions (k=100 in our work), each word can be represented as the vector defined as follows:

$$\overrightarrow{vec(word)} =< v_1, v_2, \ldots, v_k > \quad (3)$$

Thus, a document can then be mapped into the space by:

$$C_s = \theta^T \cdot H^W, \quad (4)$$

where θ^T is the vector of the *tf-idf weights* of the words in the document computed by formula (1) and H^W is the word vector matrix. In this matrix, the *i-th* line represents the word vector of the word *i*. The matrix is constructed by concatenating the word vectors of all words in the document. Via matric multiplication, a document is transferred to a vector of semantic categories, denoted by C_s.

When we get the word vectors of the issue report and buggy-review set, we can use cosine similarity defined by formula (2) to compute their semantic similarity.

MCG: it maps text format entities into semantic concept categories with some probabilities. This similarity metric can also overcome the limitation in traditional token-based models such as tf•idf that only compares lexical words in the document. It captures the semantics of words by mapping words to their concept categories. By using MCG, a word can be represented as its semantic concept categories with probabilities. For example, the word "Baidu", which can be categorized into a large number of concepts such as "company", "software", and "search". Therefore, a word can be transferred to a concept vector so that a document can then be mapped into the space by:

$$C_d = \theta^T \cdot H^M, \quad (5)$$

where θ^T is the vector of the *tf-idf weights* of the words in the document computed by formula (1) and H^M is the concept matrix, which is constructed by concatenating the concept vectors of all words in the document. Via matrix multiplication, a document is transferred to a vector of concept categories, denoted as C_d. Actually, the document is mapped to the concept space by assigning a probability to each concept category to which the document belongs. This probability is estimated by summing up the corresponding probabilities of all the words contained in the document.

When we get the concept vectors of the issue report and buggy-review set, we can use cosine similarity defined by formula (2) to compute their semantic similarity.

When the similarity score is more than the threshold, we treat the issue report as the bug report. In other words, the issue report is labeled as "bug". Otherwise, the issue report's label is not bug.

V. Experiment

A. Setup of experiment

We collect the issue reports and the user reviews which have less than 3 stars or lower from 10 open source mobile apps in GitHub. Note that we treat the closed issue reports as the experimental object because they have the whole records of life-cycle so that it is easy to verify the experimental results. We first download top-100 popular open source mobile apps according to Ranking Repositories[3] in GitHub as our candidate projects, and then we filter out the projects which have less than 1400 reviews because a small number of user reviews can affect the result of automated labelling. The scale of our data set is shown in Table I.

In this data set, we note that there are 36.7% (2921/7966 ≈36.7) of issue reports are not be labeled. Therefore, the goal of our experiment is to automatically label them using our approach described in Section Ⅲ.

In order to easily verify our approach, we also utilize the proposed approach to predict the label "bug" for the issue report which had labeled as "bug". If the final list includes this issue report, the prediction is correct; otherwise, the prediction is wrong. For unlabeled issue reports, we first label them manually to verify the final result using our automated approach. The first author and Mr. Jiachi Chen who is a RA at the Hong

[3] https://github-ranking.com/repositories

Kong Polytechnic University are responsible for marking the unlabeled issue reports. They all have more than 3 years debugging experiences and are familiar with the projects in GitHub. One person is responsible for labeling half of unlabeled data while another person is responsible for labeling another half of data. Then they exchange their data each other. If the result is not consistent, a senior developer who has more than 10 years debugging experience and also has the experience to develop the projects managed in GitHub is invited to make the final decision for ensuring the reliability of the final result.

Table Ⅰ The scale of our data set

Project	#reports	#unlabeled reports	#reviews	Period
AntennaPod	1,107	382	2,082	03.08.2012-21.12.2016
Automattic	222	44	1,394	18.01.2013-30.11.2016
cgeo	1,434	331	4,466	18.01.2011
chrislacy	193	22	1,471	
k-9 mail	783	337	4,456	15.03.2015-23.01.2017
OneBusAway	314	30	2,103	16.02.2013-28.06.2014
Twidere	653	245	2,031	06.07.2014-16.12.2016
UweTrottmann	399	192	4,459	26.07.2011-23.11.2016
WhisperSystems	1,524	1,170	4,443	26.12.2011-17.01.2017
WordPress	1,337	168	4,433	08.03.2013-14.01.2017
All	7,966	2,921	31,368	

We utilize F1 score [11] to evaluate our result. F1 is a frequently-used evaluation function, which is defined by:

$$\mathrm{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \qquad (6)$$

where Precision and Recall are computed by:

$$\mathrm{Precision} = \frac{TP}{TP+FP}, \qquad (7)$$

$$\mathrm{Recall} = \frac{TP}{TP+FN} \qquad (8)$$

Here, TP (*i.e.*, Ture Positive instances) indicates the number of instances (*i.e.*, issue reports in our work) labeled correctly, FP (*i.e.*, False Positive instances) represents the number of instances labeled incorrectly, and FN (*i.e.*, False Negative instances) shows the number of correct instances that are not labeled by the approach.

B. *Parameter adjusting*

After we compute the similarity score between each issue report and the buggy-review set by using three metrics-tf•idf, Word2Vec, and MCG, the issue reports are labeled as "bug" when the scores are more than the defined threshold. Therefore, the threshold is the important parameter, which decides the performance of our approach. Table II shows the F1 scores of 10 projects when we select the different thresholds (0.1 to 0.9) using Word2Vec. We highlight the best values when selecting the corresponding threshold. Due to the limited space, we do not show the adjusting process of the thresholds for 10 projects by using other similarity metrics, *i.e.*, tf•idf and MCG. But we show them at our GitHub repository: https://github.com/heulaoyoutiao/bugtag in order to let each scholar easily reproduce our work.

C. *Experimental result*

We adopt the best threshold to implement our approach. Table III shows the Precision, Recall, and F1 scores of all projects using three metrics.

From this table, we note that our approach using Word2Vec shows the best performance due to the highest Precision (47.26%), Recall (93.77%), and F1 scores (61.47%). Our approach using MCG shows the second-best performance while our approach using tf•idf shows the lowest performance.

We analyze the possible reasons for this evaluation result. tf•idf adopts term frequency and inverse document frequency, but it does not consider the term's semantic concept. On the contrary, Word2Vec and MCG also preserve terms' semantic and syntactic relationships. Therefore, our approach using Word2Vec or MCG performs better than that using tf•idf. Word2Vec performs the best may be caused by the characteristics of our data sets. We will deeply analyze the reasons in the future.

VI. LIMITATION

We only collected the issue reports and bug reports from 10 mobile apps managed by GitHub to perform our experiments. These apps are selected according to Ranking Repositories in GitHub. Thus, our approach may not be generalizable to other projects. Even though we think that these popular projects are representative, we would like to further explore more projects in our future work.

Moreover, we only consider to automatically label the "bug" for issue reports by computing the textual similarity between buggy user reviews and issue reports. Machine learning techniques can also be utilized to do this task. In this future, we can consider to implement deep learning-based automated labelling approach for recommending more labels such as feature and question to unlabeled issue reports.

VII. CONCLUSTION AND FUTURE WORK

In this paper, we propose an approach to automatically label an issue report as "bug" or not. This approach considers the relationship between buggy user reviews and issue report. It computes the similarity scores between them using three metrics such as tf•idf, Word2Vec, and MCG. The experimental result shows that our approach with Word2Vec performs the best.

In the future, we will propose an approach and corresponding system to automatically recommend more labels (*e.g.*, feature, question) for issue reports. Moreover, we consider to utilize deep learning to tag these labels.

ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under grant 61602258, the China Postdoctoral Science Foundation under grant 2017M621247, the Heilongjiang Postdoctoral Science Foundation under grant LBH-Z17047, and the Fundamental Research Funds for the Central Universities under grant HEUCFJ170604.

Table II F1 scores (%) of 10 projects when selecting different threshold using Word2Vec

Threshold	**F1 scores (%) of 10 projects**									
	AntennaPod	Automattic	cgeo	chrislacy	k-9 mail	OneBusAway	Twidere	UweTrottmann	WhisperSystems	WordPress
0.1	**63.05**	**76.51**	**65.40**	**57.50**	62.18	70.12	46.66	**62.20**	74.85	**35.41**
0.2	63.01	76.13	65.24	57.05	62.29	70.12	46.99	62.06	**74.89**	35.32
0.3	62.70	74.23	65.20	57.41	62.17	**70.27**	**47.07**	60.41	73.73	35.29
0.4	62.90	74.53	64.17	55.77	**62.39**	70.00	45.80	60.69	72.41	35.31
0.5	60.80	72.26	62.12	53.33	61.00	70.06	44.81	58.15	69.30	35.34
0.6	54.45	72.92	57.22	51.23	56.39	70.95	37.43	56.84	60.89	36.01
0.7	41.93	60.41	46.57	33.33	45.68	68.60	32.56	48.45	45.11	30.29
0.8	26.79	40.21	30.91	19.21	31.59	55.93	17.56	26.86	18.54	22.88
0.9	5.73	4.11	7.30	1.40	9.65	11.34	2.60	2.07	3.44	2.74

Table III Performance of our approach using three similarity metrics when the best threshold is selected

Project	**Our approach using tf•idf**			**Our approach using MCG**			**Our approach using Word2Vec**		
	Precision(%)	Recall(%)	F1(%)	Precision(%)	Recall(%)	F1(%)	Precision(%)	Recall(%)	F1(%)
AntennaPod	45.37	90.75	60.50	44.60	93.50	60.39	46.13	99.61	63.05
Automattic	76.03	78.72	77.35	67.03	86.52	75.54	66.49	90.07	76.51
cgeo	48.47	77.19	59.55	47.45	92.11	62.63	48.83	99.00	65.40
chrislacy	49.01	52.86	50.86	50.30	60.00	54.72	51.11	65.71	57.50
k-9 mail	38.30	74.22	50.53	41.48	90.37	56.86	45.51	99.15	62.39
OneBusAway	55.81	84.71	67.29	54.43	97.65	69.89	54.34	99.41	70.27
Twidere	30.61	79.71	44.24	29.90	89.86	44.87	31.27	95.17	47.07
UweTrottmann	46.67	64.32	54.09	46.75	81.62	59.45	47.44	90.27	62.20
WhisperSystems	55.29	80.76	65.64	55.90	91.21	69.32	59.97	99.67	74.89
WordPress	21.35	56.09	30.93	21.55	94.46	35.09	21.53	99.63	35.41
Average	46.69	73.93	56.10	45.94	87.73	58.88	**47.26**	**93.77**	**61.47**

REFERENCES

[1] T. Zhang, J. Chen, X. Luo, and T. Li, "Bug reports for desktop software and mobile apps in github: what is the difference?" IEEE Software, published online, 2017.

[2] N. Shahmehri, M. Kamkar, and P. Fritzson, "Semi-automatic bug localization in software maintenance," Proceedings of the International Conference on Software Maintenance 1990, 1990, pp.30-36.

[3] B. Fu, J. Lin, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 1276-1284.

[4] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? Summarizing app reviews for recommending software changes," Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 499-510.

[5] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "Interpreting tf-idf term weights as making relevance decisions," ACM Transactions on Information Systems, vol. 26, no. 3, pp. 1-37, 2008.

[6] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Proceedings of the 26th International Conference on Neural Information Processing Systems, 2013, pp. 3111-3119.

[7] L. Wei, Y. Liu, and S.-C.Cheung, "Oasis: Prioritizing static analysis warnings for android apps based on app user reviews," Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 672-682.

[8] E. Cambria, an B. White, "Jumping NLP curves: a review of natural language processing research," IEEE Computational Intelligence Magazine, vol. 9, no. 2, pp. 48-57, 2014.

[9] S. Tata, and J. M. Patel, "Estimating the selectivity of tf-idf based cosine similarity predicates," SIGMOD Record, vol. 36, no. 2, pp. 7-12, 2007.

[10] L. De Vinc, G. Zuccon, B. Koopman, L. Sitbon, and P. Bruza, "Medical semantic similarity with a neural language model," Proceedings of the 23rd ACM International Conference on Information and Knowledge management, 2014, pp. 1819-1822.

[11] C. Goutte, and E. Gaussier, "A probabilistic interpretation of precision, recall, and f-score, with implication for evaluation," Advances in information retrieval, 2005, pp. 345-359.

[12] M. D. Syer, M. Nagappam, A. E. Hassan, and B. Adams, "Revisiting prior empirical findings for moble apps: an empirical case study on the 15 most popular open-source android apps," Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, 2013, pp. 283-297.

[13] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," Proceedings of the 17th European Conference on Software Maintenance and Reengineering, 2013, pp. 133-143.

[14] B. Zhou, I. Neamtiu, and R. Gupta, "A cross-platform analysis of bugs and bug-fixing in open source projects: desktop vs. android vs. ios," Proceedings of the 19th International Conferrence on Evaluation and Assessment in Software Engineering, 2015, no. 7, pp. 1-10.

[15] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," Proceedings of the 39th International Conference on Software Engineering, 2017, pp. 106-117.

[16] A. Ciurumelea, A. Schaufelbhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, 2017, pp. 91-102.

[17] N. Genc-Nayebi, and A. Abran, "A systematic literature review: opinion mining studies from mobile app store user reviews," Journal of Systems and Software, vol. 125, pp. 207–219, 2017.

[18] H. Chen, D. He, S. Zhu, and J. Yang, "Toward detecting collusive ranking manipulation attackers in mobile app markets," Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, 2017, pp. 58-70.