Automatically Extracting Bug Reproducing Steps from Android Bug Reports

Yu Zhao[1](✉), Kye Miller[1], Tingting Yu[1], Wei Zheng[2], and Minchao Pu[2]

[1] University of Kentucky, Lexington, USA
{yzh355,kemi232}@g.uky.edu, tyu@cs.uky.edu
[2] Northwestern Polytechnical University, Xi'an, China
pmcmc@mail.nwpu.edu.cn, wzheng@nwpu.deu.cn

Abstract. Many modern software projects use bug-tracking systems (e.g., Bugzilla, Google Code Issue Tracker) to track software issues and help developers reproduce these issues. There has been recent work on automatically translating the natural language text (i.e., steps to reproduce) of bug reports to reproducing scripts, targeted at Android apps, to facilitate app debugging process. The scripts describe the event sequences leading to the app issues and thus can be reused for testing newer versions of the apps. However, existing techniques require manually providing the text description of steps to reproduce for generating reproducing scripts, which is a non-trivial task because natural language text in bug reports can be complex and contain much information irrelevant for bug reproduction. In this paper, we propose an approach that can automatically extract the text description of steps to reproduce (S2R) from bug reports to advance automated software issue diagnosis and test script reuse. The approach is implemented as a tool, called S2RMiner, which combines HTML parsing, natural language processing, and machine learning techniques. We have evaluated S2RMiner on 1000 original Android bug reports. The results show that S2RMiner can extract S2R with high accuracy.

Keywords: Bug reports · Steps to reproduce · Android apps

1 Introduction

Mobile devices with advanced computing ability, such as smartphones and tablets, have become increasingly prevalent. Mobile applications (a.k.a apps) in different domains are developed and used on these devices. In 2017, there were over 3.5 million apps published in Google app store [4]. As mobile apps are becoming complex due to developers adding more features to make them competitive in the market, there is an urgent need to ensure the quality of the apps. A recent survey indicates that 88% users are likely to abandon the apps

X. Peng et al. (Eds.): ICSR 2019, LNCS 11602, pp. 100–111, 2019.
https://doi.org/10.1007/978-3-030-22888-0_8

if they repeated encounter the same issue [21]. Therefore, developers need to rapidly resolve app issues to avoid losing customers.

Many modern software projects use bug tracking systems (e.g., GitHub [2], Google Code [3], Bitbucket [1]) to help developers track and reproduce app issues and thus expedite the process of resolving the issues. These systems allow users or developers to submit *bug reports* describing the issues they encountered in the apps, typically involving bug symptoms, steps to reproduce (S2R), and expected behaviors. Once a developer receives a bug report, he or she will try to reproduce the bug according to the description of S2R. However, reproducing bugs from bug reports is a challenging task [20]. This is because bug reports are often written by natural language, which can be imprecise and often incomplete. In addition, even if a bug report is perfectly understood by developers, the event-driven nature of mobile apps can make actual process of bug reproduction complex as it may require developers to manually navigate through a number of actions before exposing the bug [26].

To help developers reproduce issues reported for mobile apps, there has been some work that can automatically translate the natural language text of a bug report into a test script that can directly execute on mobile apps [15,26]. The translated test scripts can reproduce bugs or be re-used to perform regression testing for future versions of apps. For example, YAKUSU [15] analyzes the natural language text of steps to reproduce (S2R) in an Android bug report and automatically translates it into actual test cases. ReCDroid [26] uses grammar patters to extract key information from S2R and then leverages dynamic GUI exploration to reproduce the app crash guided by the extracted information. The above work uses S2R as input and assumes it is readily available and manually provided by developers. This requires additional manual effort and prevents the bug reproduction tools from being used in fully automated environment (e.g., continuous integration). Therefore, *a tool that can automatically extract S2R from bug reports is needed to enhance the efficiency of bug reproduction and the generation of reusable test scripts.*

There has been little research on targeting at detecting S2R in bug report descriptions. Most existing techniques aim to detect other types of information, such as source code snippets [10,23] and stack traces [9,22]. While one approach has been proposed to detect S2R [13], it focuses on determining whether S2R exists in a bug report rather than extract the the actual text of S2R. In fact, extracting S2R is non-trivial because natural language bug descriptions are often unstructured. Although some bug tracking systems (e.g., Bugzilla) provide semi-structured templates for reporters to write bug symptoms, S2R, and expected behaviors, they cannot guarantee that reporters will provide such information.

In this paper, we propose a new technique, S2RMiner, targeted at Android apps, that can automatically analyze bug reports to extract S2R. The extracted S2R can achieve several straightforward benefits, such as providing succinct information to help developers understand the reported bug and providing insights for improving the quality of bug reports (e.g., an absent S2R indicates a low-quality bug report). Moreover, S2RMiner facilitates *software reuse* in the context of regression testing [25]. For example, bug-triggering test cases are often reused to test newer versions of software. Therefore, the extracted S2R by S2RMiner, once translated into automated test scripts, can be reused in regression testing.

S2RMiner leverages HTML parsing, natural language processing (NLP), and machine learning techniques to analyze the bug reports (in HTML format) directly downloaded from the issue tracking systems for extracting S2R. Specifically, HTML parsing extracts text relevant to S2R from the HTML files of bug reports. NLP is used to obtain different types of text features of each sentence by employing part-of-speech (POS) tags, dependency parsing, and stemming. With text features, a Support Vector Machine(SVM)-based machine learning method [17] is used to predict and extract S2R.

To determine the effectiveness of our approach, we apply S2RMiner on 1000 bug reports randomly selected from GitHub and Google code. The results showed that S2RMiner is effective at extracting S2R. The average F-measures are 0.65 and 0.65 on GitHub and Google code, respectively. The accuracy scores on the two datasets are 0.87 and 0.93.

In summary, our paper makes the following contributions:

– The design and development of the first approach that can extract S2R sentences directly from the textual description of bug reports.
– An empirical study showing that S2RMiner is effective at extracting S2R with high precision.
– The implementation of our approach as a publicly available tool, S2RMiner, along with all experiment data [5].

In the next section, we introduce a motivating example. We then present the details of S2RMiner in Sect. 3. Our empirical study and results are presented Sects. 4 and 5. We present the related work in Sect. 6, and then give our conclusions in Sect. 7.

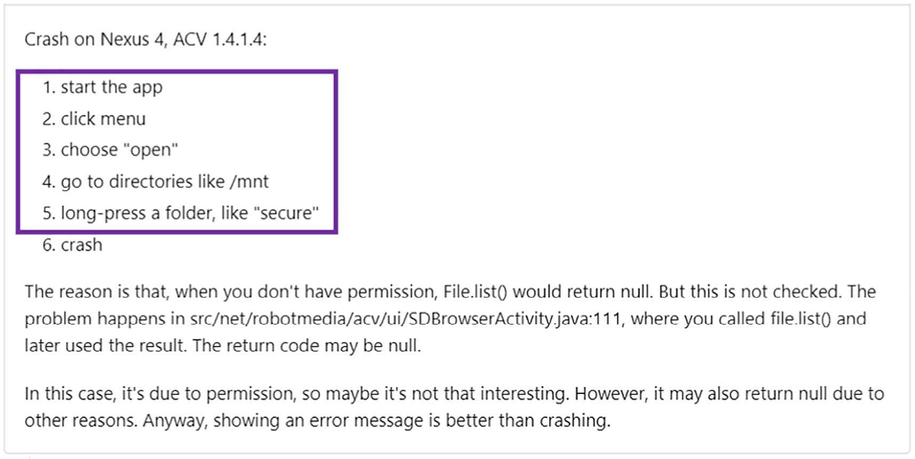
2 A Motivating Example

Figure 1 shows an example bug report. Given the whole text description, it is unclear which sentence belongs to S2R. Therefore, existing bug reproduction tools (e.g., YAKUSU and ReCDroid) cannot directly work on the raw description of the bug report.

S2RMiner is designed and implemented to accurately extract all S2R sentences from the bug report. Specifically, S2RMiner takes the HTML format of the issue page (Fig. 2) as input and outputs a sequence of S2R sentences (i.e., the text inside the rectangle indicates S2R). If a bug report does not have S2R (e.g., Fig. 3), S2RMiner will report S2R is missing.

3 S2RMiner Approach

S2RMiner consists two major phases. In the first phase, S2RMiner uses a HTML parser to extract the key text containing S2R from the HTML format of an issue



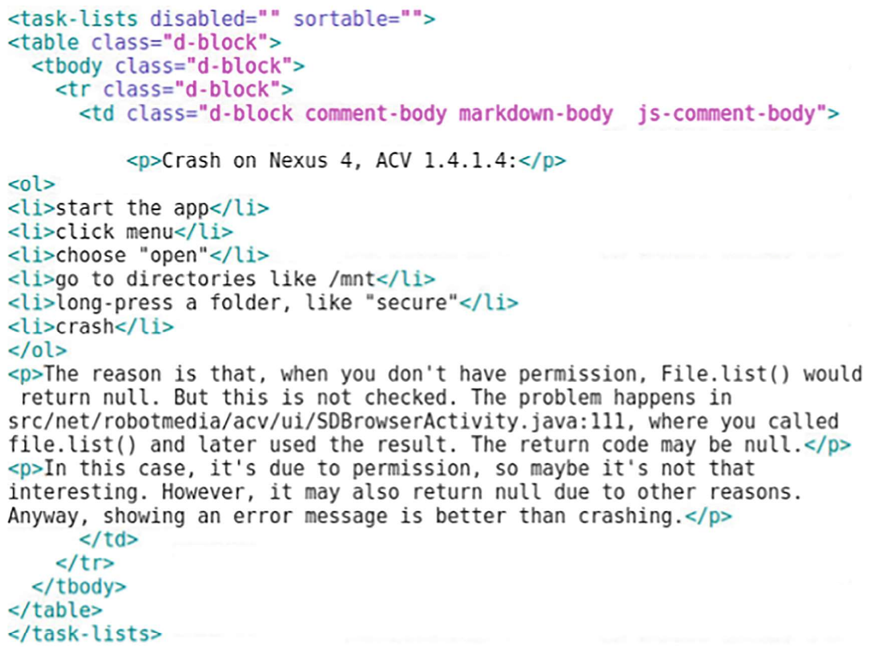
Crash on Nexus 4, ACV 1.4.1.4:

1. start the app
2. click menu
3. choose "open"
4. go to directories like /mnt
5. long-press a folder, like "secure"
6. crash

The reason is that, when you don't have permission, File.list() would return null. But this is not checked. The problem happens in src/net/robotmedia/acv/ui/SDBrowserActivity.java:111, where you called file.list() and later used the result. The return code may be null.

In this case, it's due to permission, so maybe it's not that interesting. However, it may also return null due to other reasons. Anyway, showing an error message is better than crashing.

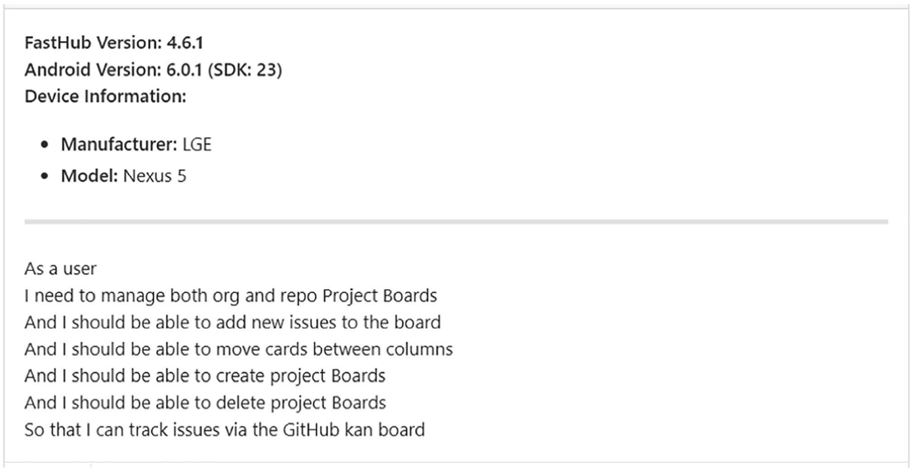
Fig. 1. A bug report



```
<task-lists disabled="" sortable="">
<table class="d-block">
  <tbody class="d-block">
    <tr class="d-block">
      <td class="d-block comment-body markdown-body  js-comment-body">

          <p>Crash on Nexus 4, ACV 1.4.1.4:</p>
<ol>
<li>start the app</li>
<li>click menu</li>
<li>choose "open"</li>
<li>go to directories like /mnt</li>
<li>long-press a folder, like "secure"</li>
<li>crash</li>
</ol>
<p>The reason is that, when you don't have permission, File.list() would
 return null. But this is not checked. The problem happens in
src/net/robotmedia/acv/ui/SDBrowserActivity.java:111, where you called
file.list() and later used the result. The return code may be null.</p>
<p>In this case, it's due to permission, so maybe it's not that
interesting. However, it may also return null due to other reasons.
Anyway, showing an error message is better than crashing.</p>
      </td>
    </tr>
  </tbody>
</table>
</task-lists>
```

Fig. 2. HTML format of Fig. 1


FastHub Version: 4.6.1
Android Version: 6.0.1 (SDK: 23)
Device Information:

- **Manufacturer:** LGE
- **Model:** Nexus 5

As a user
I need to manage both org and repo Project Boards
And I should be able to add new issues to the board
And I should be able to move cards between columns
And I should be able to create project Boards
And I should be able to delete project Boards
So that I can track issues via the GitHub kan board

Fig. 3. Missing S2R

page. As shown in Fig. 2, the HTML issue page contains HTML tags, such as </li>, </ol>, and <tbody class = "d-block">. S2RMiner filters out the HTML tags and obtains only text "start the app".

In the second phase, S2RMiner uses NLP techniques to extract text features from the sentences of the filtered text. It then uses machine learning to label whether a sentence belongs to S2R. Finally, the sentences labeled with S2R are saved into a output file. The output can either be manually analyzed by developers or provided as an input to the test script generation tool (e.g., YAKUSU) or the bug reproduction tool (ReCDroid).

3.1 Phase 1: HTML Parsing

Many bug tracking systems allow reporters to submit bug reports through web pages and developers can reply to the bug report by adding comments to the page. Therefore, bug report descriptions are often downloaded as HTML files. The original HTML file has a number of HTML tags. In addition, the raw HTML file contains many other types of information, such as bug symptoms, expected behaviors, developers' replies, CSS code, page information, and so on. These types of information are irrelevant to S2R. Even on a simple bug report shown in Fig. 1, the associated HTML file contains 1371 lines and is as large as 104 KB. S2RMiner needs to eliminate all such information to obtain the minimum amount of text containing S2R.

Specifically, S2RMiner removes all HTML tags and parses the first block of text in the HTML page. The intuition is that only the first comment involves S2R described by the reporter.

3.2 Phase 2: S2R Extraction

The problem of detecting S2R sentences can be formulated into the problem of text classification [12]. Given a sentence, a text classification tool can predict whether it is a S2R sentence or not. S2RMiner performs the classification in three steps. First, it splits the text into individual sentences by employing several heuristics. Second, for each sentence, S2RMiner extracts text features used for building a classifier. Third, leveraging the text features, S2RMiner builds a classifier that can predict whether a sentence is S2R. All S2R sentences are saved into an output file.

Splitting Text into Sentences. S2RMiner first needs to detect individual sentences for being labeled as S2R sentence or non-S2R sentence. We cannot simply view each text line as a sentence because a line may contain more than one sentence. In the example of Fig. 1, the first line in the second to last paragraph ("The reason is that ...") contains two sentences. While tools such as spaCy [7] have the capability of detecting sentences, they are not accurate because they are not intended to deal with bug report text.

To address this problem, S2RMiner designs several heuristics to identify sentences from each line of the text: (1) one text line contains at least one sentence; (2) a text segment ending with a full stop "." is a sentence; (3) if a full stop is preceded by a number (e.g., "1.") or a part of ellipsis, it is not considered to be the end of a sentence.

Extracting Text Features. S2RMiner employs a well-known NLP tool spaCy [7] to extract text features from each sentence. We consider three types of features. The first type of feature is *stemming*, which transforms each word in the sentence to its stem. Stemming is the process of removing the ending of a derived word to get its root form. For example, "clicking", "clicks", and "clicked" become "click". Without stemming, multiple words with the same meaning would be used as different features, resulting in too many features and thus a low quality machine learning model.

The second type of feature is *part-of-speech (POS)* tags, which labels each word with a POS tag. The features used by S2RMiner are words labeled as "noun", "verb", and "adjective".

The third type of features is *dependency parsing*, which analyzes the grammar structure of the sentence. Specifically, words labeled as root, predicate, and object are considered as features.

Building a Text Classifier. We use n-grams and CountVectorizer [19] to transform text features into numerical features, which is easy to process by a machine learning tool. A n-gram a contiguous sequence of n items from a given sequence of text. For example, 1-gram (or unigram) indicates single word tokens and 2-gram (or bigrams) indicates two consecutive word tokens.

The current implementation of S2RMiner uses Support Vector Machines [18] (SVM) to do binary classification given the extracted text features. SVM outputs a “1” if a sentence is a S2R and a “0” otherwise. S2RMiner saves the sentence labeled with “1” into the result file for each bug report.

4 Evaluation

To evaluate S2RMiner, we consider two research questions:

RQ1: What is the performance of S2RMiner in extracting S2R from bug reports?
RQ2: Which types of text features have the best performance in extracting S2R from bug reports?

RQ1 lets us evaluate the effectiveness of S2RMiner in extracting S2R. RQ2 lets us investigate how different types of text features influence of the performance of S2RMiner.

4.1 Datasets

We evaluated S2RMiner on bug reports from GitHub [2] and Google Code [3]. To prepare the training set, we randomly crawled 500 bug reports from GitHub and 500 reports from Google Code. We hired two undergraduate students to label the sentences of each bug report as S2R and non-S2R. During the labeling process, the inspector read the reports with sufficient details in the bug descriptions to identify S2R sentences. To ensure the correctness of our results, the manual inspections were performed independently by the two undergraduate students. Any time there was dissension, the authors and the inspectors discussed to reach a consensus.

We randomly divided the 500 bug reports from both datasets into two sets—400 for training 100 for testing. Each bug report contains one or more sentences, which are the instances for building machine learning models.

4.2 Experiment Design

The experiment was conducted on a physical x86 machine running with Ubuntu 14.04 installed. The NLP techniques of S2RMiner was implemented by the spaCy dependency parser [7]. The classifier was implemented by Scikit-Learn [6].

Performance Metrics. We chose performance metrics allowing us to answer each of our two research questions. Specifically, we employ accuracy, precision, recall, and F1-measure. A sentence can be classified as: S2R when it is truly a S2R sentence (true positive, TP); it can be classified as a S2R sentence when it is actually not (false positive, FP); it can be classified as a non-S2R sentence when it is actually a S2R sentence (false negative, FN); or it can be correctly classified as a non-S2R sentence (true negative, TN).

– **Accuracy:** the number of instances correctly classified over the total number of instances.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

– **Precision:** the number of instances correctly classified as S2R over the number of all instances classified as S2R.

$$P = \frac{TP}{TP + FP}$$

– **Recall:** the number of instances correctly classified as S2R over the total number of S2R instances.

$$R = \frac{TP}{TP + FN}$$

– **F-measure:** a composite measure of precision and recall for buggy instances.

$$F(b) = \frac{2 * P * R}{P + R}$$

Combinations of Different Text Features. RQ2 aims to evaluate how S2RMiner performs when using the combinations of different types of text features. Table 1 shows the features used for evaluation.

Table 1. Types of text features

No NLP techniques	Only use original words as features
Stem(1 gram)	Only stem of the word as features
Stem(3 gram)	Only stem of the word but consider 3 g relationship
Stem(3 gram)+pos	Combine stem and part of speech as features
Stem(3 gram)+dep	Combine stem and dependency as features
Stem(3 gram)+pos+dep	Combine three of them as features
(Stem+pos+dep)(3 gram)	Add 3 gram relationship to all of features

4.3 Threats to Validity

The primary threat to external validity for this study involves the representativeness of our subjects and bug reports. Other subjects may exhibit different behaviors. Data recorded in bug tracking systems can have a systematic bias relative to the full population of bug reports [11] and can be incomplete or incorrect [8]. However, we do reduce this threat to some extent by using two well studied open source projects and bug sources for our study. We cannot claim that our results can be generalized to all systems of all domains though.

The primary threat to internal validity involves the use of manual inspection to identify the S2R sentences To minimize the risk of incorrect results given by manual inspection, the sentences are labeled independently by two people.

The primary threat to construct validity involves the dataset and metrics used in the study. To mitigate this threat, we used bug reports from two bug tracking systems, which are publicly available and generally well understood. We also used the well known, accepted, and validated measures of accuracy, recall, precision, and F-measure.

5 Results and Analysis

Tables 2 and 3 summarizes the results of the two datasets.

RQ1: Performance of S2RMiner. Tables 2 and 3 show that S2RMiner is able to extract S2R from bug reports in GitHub and Google Code. The accuracy is above 0.85 for GitHub and above 0.92 for Google code. Regarding the precision and recall, the best F-score is above 0.6 for both Github and Google code. We consider F-measures over 0.6 to be good [14].

Table 2. GitHub result

GitHub result	TP	TN	FP	FN	Accuracy	Precision	Recall	F-score
No NLP techniques	79	612	33	84	0.85	0.69	0.47	0.56
Stem(1 gram)	99	596	49	69	0.86	0.66	0.62	0.64
Stem(3 gram)	88	612	33	71	0.87	0.72	0.55	0.63
Stem(3 gram)+pos	94	605	40	65	0.86	0.70	0.59	0.64
Stem(3 gram)+dep	94	609	36	65	0.87	0.72	0.59	0.65
Stem(3 gram)+pos+dep	90	607	38	69	0.86	0.70	0.57	0.63
(Stem+pos+dep)(3 gram)	87	605	40	72	0.86	0.69	0.55	0.61

In both bug tracking systems, the precision scores are better than the scores of recall. We analyzed the results and found that the low recall could be due to (1) the small training set; (2) incorrect labels. As part of the future, we intend to expand the training set and perform more robust labeling work.

In summary, the above results imply that *S2RMiner is effective at extracting S2R*.

RQ2: Comparison of Different Types of Text Features. As we can see from the two tables, comparing the text feature without using NLP technique (F_n) with the other feature combinations in the GitHub dataset, F_n performed the worst. However, this is not true in the Google Code dataset. When comparing different feature combinations with NLP applied, the dependency parsing feature type slightly improved the performance in terms of F-measures.

Overall, these results imply that the *the stemming and dependency parsing feature types can potentially improve the performance of S2RMiner.*

6 Related Work

Yukusu [15] translate S2R descriptions into executable test scripts. ReCDroid [26] use S2R descriptions to guide Android crash reproduction. Both tools assume that S2R is readily available and can be provided by users. In contrast, S2RMiner aims to automatically extract S2R sentences from raw bug reports.

Table 3. Google code result

Google Result	TP	TN	FP	FN	Accuracy	Precision	Recall	F-score
No NLP techniques	40	516	13	32	0.92	0.75	0.56	0.64
Stem(1 gram)	44	507	22	28	0.92	0.67	0.61	0.64
Stem(3 gram)	38	519	10	34	0.93	0.79	0.53	0.63
Stem(3 gram)+pos	36	518	11	36	0.92	0.77	0.5	0.61
Stem(3 gram)+dep	39	520	9	33	0.93	0.81	0.54	0.65
Stem(3 gram)+pos+dep	40	517	12	32	0.93	0.77	0.56	0.65
(Stem+pos+dep)(3 gram)	36	518	11	36	0.92	0.76	0.5	0.61

Chaparro et al. [13] proposed an approach, called DeMIBuD, to detect whether S2R is missing in a bug report. Their approach is probably most related to S2RMiner. However, DeMIBuD focuses on detecting whether a bug report contains S2R or not, whereas S2RMiner aims to extract all S2R in a bug report. In addition, S2RMiner analyzes original issue page of a bug report (e.g.,in the HTML format), whereas DeMIBuD can only handle the regular text.

There has been some research on mining bug repositories to classify and predict specific fault types. For example, Gegick et al. [16] classify bug reports as either security- or non-security-related. However, these techniques neither classify configuration bug reports nor identify concrete bug sources. Xia et al. [24] use text mining to categorize configuration bug reports related to system settings and compatibilities. In contrast, S2RMiner analyzes bug reports at the sentence to extract S2R.

7 Conclusion

We have presented S2RMiner, an automated approach to extract step to reproduce (S2R) sentences from bug reports. S2RMiner leverages HTML parsing, natural language processing, and machine learning techniques to analyze bug reports in the HTML formats and extract the needed contents from it. We have evaluated S2RMiner on two datasets from two popular bug tracking systems—GitHub and Google Code. The results showed that S2RMiner can extract S2R with a high accuracy and that the stem and grammar dependency text features play important roles in improving the performance of S2RMiner.

Acknowledgements. This research is supported in part by the NSF grant CCF-1652149.

References

1. Bitbucket. https://bitbucket.org
2. Github. https://github.com
3. Google code archive. https://code.google.com/archive/
4. Google play data. https://en.wikipedia.org/wiki/Google_Play
5. S2rminer publick link. https://github.com/AndroidTestBugReport/S2RMiner
6. Scikit-learn. https://scikit-learn.org/stable/
7. spaCy. https://spacy.io/
8. Aranda, J., Venolia, G.: The secret life of bugs: going past the errors and omissions in software repositories, pp. 298–308 (2009)
9. Bacchelli, A., Cleve, A., Lanza, M., Mocci, A.: Extracting structured data from natural language documents with island parsing. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 476–479. IEEE (2011)
10. Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.: Extracting structural information from bug reports. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories, pp. 27–30. ACM (2008)
11. Bird, C., et al.: Fair and balanced?: bias in bug-fix datasets, pp. 121–130 (2009)
12. Burstein, J., Marcu, D., Andreyev, S., Chodorow, M.: Towards automatic classification of discourse elements in essays. In: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, pp. 98–105. Association for Computational Linguistics (2001)
13. Chaparro, O., et al.: Detecting missing information in bug descriptions. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 396–407 (2017)
14. Dougherty, G.: Pattern Recognition and Classification: An Introduction. Springer, New York (2012). https://doi.org/10.1007/978-1-4614-5323-9
15. Fazzini, M., Prammer, M., d'Amorim, M., Orso, A.: Automatically translating bug reports into test cases for mobile apps. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 141–152. ACM (2018)
16. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: International Working Conference on Mining Software Repositories, pp. 11–20 (2010)
17. Joachims, T.: Making large-scale SVM learning practical. Technical report, SFB 475: Komplexitätsreduktion in Multivariaten ... (1998)
18. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0026683

19. Kulkarni, A., Shivananda, A.: Converting text to features. In: Kulkarni, A., Shivananda, A. (eds.) Natural Language Processing Recipes, pp. 67–96. Springer, Heidelberg (2019). https://doi.org/10.1007/978-1-4842-4267-4_3
20. Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., Poshyvanyk, D.: Autocompleting bug reports for android applications. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 673–686. ACM (2015)
21. Packard, H.: Failing to meet mobile app user expectations: a mobile user survey. Technical report (2015)
22. Ponzanelli, L., Mocci, A., Lanza, M.: StORMeD: stack overflow ready made data. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 474–477. IEEE (2015)
23. Rigby, P.C., Robillard, M.P.: Discovering essential code elements in informal documentation. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 832–841. IEEE (2013)
24. Xia, X., Lo, D., Qiu, W., Wang, X., Zhou, B.: Automated configuration bug report prediction using text mining. In: Computer Software and Applications Conference, pp. 107–116 (2014)
25. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. STVR **22**(2), 67–120 (2012)
26. Zhao, Y., Yu, T., Su, T., Liu, Y., Zheng, W., Zhang, J., Halfond, W.G.: Recdroid: automatically reproducing android application crashes from bug reports. In: Proceedings of the 2019 41st ACM/IEEE International Conference on Software Engineering (2019)