Information Retrieval-based Bug Localization Experiment and Research Report

Jiaxuan Tong

Abstract

Bugs are inevitable in software development, and bug fixing is an important topic that people take seriously. Which source file should developers fix when they receive a bug report? Which source file could be relevant? In this report, the information retrieval-based bug localization based on bug reports will be introduced and studied. The idea and operation are generally followed by the article: *Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports.* The implementation and experiment of bug localization include applying different IRFL (Information Retrieval-based Fault Localization) approaches, in other words, simplified versions of BugLocation, and evaluating the effectiveness of the proposed bug localization method by using metrics such as MAP(Mean Average Precision). Furthermore, many information retrieval models used in bug localization methods such as TF.IDF (Term Frequency-Inverse Document Frequency) and LSA(Latent Semantic Analysis) will be researched, implemented and compared.

1 Introduction

For a large software system development and maintenance cycle, developers always receive a large number of bug reports. These bug reports will record bug description and target list of fixed file paths later. In order to find out the corresponding source files and fix the bugs, developers have to go through all source files and check the relevant documents. However, there is always a large number of source files in a large software project. Manually screening is difficult and impractical, especially when the project has been taken over to another team. The idea of information retrieval-based bug localization based on bug reports is to compare the textual similarity between the initial bug report and the source code using standard information retrieval(IR) such as revised Vector Spca Model(rVSM), and return the rank list of potential buggy source files. To evaluate the result, returned list is compared with actual buggy files to compute accuracy.[1]

2 Background

In order to clearly explain what bug localization can do, a typical example of a bug report is referred to Figure 2.1 Example of the Bug Report. The summary and description in the bug report are usually the plain text that explains the problem and condition of the bug, and sometimes, they also include some coding segments to support. The actual buggy files will also be indicated in the bug report.

Category	Description
Summary	**DecompressingEntity** not calling close on InputStream retrieved by getContent
Description	The method **DecompressingEntity**.writeTo(OutputStream outstream) does not close the InputStream retrieved by getContent().
Buggy Files	**DecompressingEntity.java**

Figure 2.1 Example of the Bug Report [2]

2.1 General Bug Localization Process

In general, there are four bug localization processes:

Copus Creation: This step is to preprocess the source code file and create a vector of lexical tokens. In this step, the keywords, separators and operators that are common in programming language will be removed. English stop words will also be removed. The concatenation of words such as purchaseHistory will be separated as purchase and History. All words with the same root form will be recorded as one token.
Indexing: By using the indexes in the corpus, the buggy files can be located after comparing the relevance.
Query Construction: After preprocessing the source code file content, the bug report will also be processed properly. In general a bug report will be preprocessed as a query in bug localization. In this step, since the bug report is usually plain text, the stop words will be removed and each word will be stemmed and saved into the query corpus.
Retrieval and Ranking: This step compares the textual similarity between the query and each source code file content. Different information retrieval methods will be used to compute the relevance score between query and file content.

3 Method

3.1 Term frequency-inverse document frequency (tf-idf)

Term frequency is the number of times that the word appears in the document. Document frequency is the number of documents containing the particular term. The tf-idf rescalling formula is shown as below:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \cdot \text{idf(t)}$$

$$\text{idf}(t) = log\frac{1+n_d}{1+\text{df}(d,t)} + 1$$

Total number of documents

Number of documents containing term t

The resulting tf-idf vectors are then normalized by the Euclidean norm

Figure 3.1Tf-idf Rescalling Formula[3]

Instead of dropping features that are deemed unimportant, tf-idf keeps all features but rescal them by the informative level that people expect them to be. To say is in a more straightforward way, tf-idf give to any term that appears often in a particular document, but not in many document.[3] A simple example is shown as below:

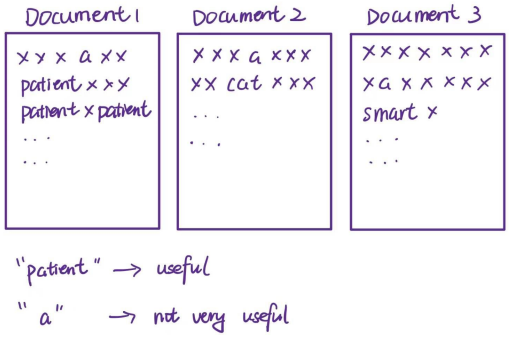

Figure 3.2 Term Weights Example

3.2 Latent Semantic Analysis

The concept of LSA in the field of information retrieval is mainly used to solve the polysemy and synonymy. For example, the word “bank”, it means water bank or financial institute;[4] the word “kids” and “children” generally have the same meaning. The cosine similarity method can not solve the problem well. Therefore, the LSA method is proposed, it applies dimensionality reduction by using truncated SVD(singular value decomposition) to map the term and text to the same space.
Using this IR method should theoretically improve the accuracy of the similarity comparison because the LSA does not only consider the term frequency but also consider the word meaning.

3.3 Mean Average Precision

The evaluation metric used for the bug localization is MAP. Map provides a single-figure measure of quality of information retrieval. For bug locators, each project may contain many queries, and each query may have multiple ranked relevant source code files, the precision score will be calculated after comparing the relevance between the actual fixed file path(ground-truth) and the ranked relevant source code file path (target scores) for each query. Then the average of the precision values for all queries in a project can be calculated. After that, depending on different sets of queries, the MAP is the mean of the average precision values for different sets of queries.[5] The relevant formula is shown as below:

$$AvgP_i = \sum_{i=1}^{M} \frac{P(j) \times pos(j)}{\text{number of positive instances}}$$

Figure 3.3 Average of the Precision Values Obtained for the Query[5]

4 Experiment

The dataset called Bench4BL is a collection of bug reports and the corresponding source code files. The collection contains 10017 bug reports collected from 51 open source projects, which is a great resource to support bug localization research.[6]
For this project, a subset of Bench4ML which includes the bug report and its corresponding source code files from 12 projects will be used. After implementing the different IRFL approaches, the map value will be calculated for each of the Project.

4.1 Objectives

The objective of the experiment is to implement and apply several IRFL approaches on the Bench4ML dataset. Rank the most top 1, 5, and 10 relevant buggy files after comparing the textual similarity with the actual buggy files. Then evaluate different approaches by using the MAP value and analyze the difference of the different methods in bug localization.

4.2 Procedure

The procedure will follow the general bug localization process that was being mentioned in the previous section 2.1. Before starting to implement the bug locator, the preparation includes reading the data including the bug reports and source code files of all 12 projects, and saving them as two pickle files have been done beforehand.

4.2.1 Read pickle files into DataFrames

The first step is to transform source code files and bug reports into the DataFrame in order to preprocess the text content inside of them. The sample DataFrames are shown below:

	filename	unprocessed_code	project
0	\gitrepo\src\java\org\apache\commons\collectio...	/*\n * Licensed to the Apache Software Founda...	COLLECTIONS
1	\gitrepo\src\java\org\apache\commons\collectio...	/*\n * Licensed to the Apache Software Founda...	COLLECTIONS

Figure 4.1 Sample Source Code Files DataFrame[7]

id	fix	text	fixdate	summary	description	project	average_precision
217	[org.apache.commons.collections.map.flat3map.j...	NaN	2006-07-18 22:02:11	Flat3Map.Entry.setValue() overwrites other Ent...	Flat3Map's Entry objects will overwri...	COLLECTIONS	0.0
214	[org.apache.commons.collections.testextendedpr...	NaN	2006-07-18 22:44:33	ExtendedProperties - field include should be n...	The field "include" in ExtendedProperties is c...	COLLECTIONS	0.0

Figure 4.2 Sample Bug Reports DataFrame[7]

4.2.2 Preprocessing the content

In order to properly preprocess the content of source code files, only keep the meaningful tokens, and reduce the size of corpus and computation time (comparing the textual similarity in later steps). Many self-defined text preprocess methods apply specifically for code. The tasks are listed below:

- Replace all dot to space due to the naming rules of java (For instance, “str.translate” needs to be splitted as “str” and “translate”)
- Remove all punctuations
- Split the text
- Eliminate all words belonging to stop words and java_keywords (For instance, “the”, ”a”, ”boolean”, “int”, etc.)
- Split camel case words
- Stem and append each word in to the list

Some external libraries such as *sklearn.feature_extraction.text*[] and a self-defined *java keyword* text file[8] are used in this step.

By looking through the content of the bug reports. There are some code segments inside the summary and description. Therefore, the bug reports will not only be preprocessed by removing the stop words and stemming, but also need to deal with code. So the same text preprocessing method applies to the bug report content.

The processed code saves in the processed_code column after preprocessing the unprocessed_code column. The processed content of bug report saves in the query column after preprocessing the summary and description columns. The sample of processed DataFrames are shown below:

	filename	unprocessed_code	project	processed_code
0	.gitrepo.src.java.org.apache.commons.collectio...	/*\n * Licensed to the Apache Software Founda...	COLLECTIONS	licens apach softwar foundat asf contributor l...
1	.gitrepo.src.java.org.apache.commons.collectio...	/*\n * Licensed to the Apache Software Founda...	COLLECTIONS	licens apach softwar foundat asf contributor l...

Figure 4.3 Sample Processed Source Code Files DataFrame[7]

id	fix	text	fixdate	summary	description	project	average_precision	query
217	[org.apache.commons.collections.map.flat3map.j...	NaN	2006-07-18 22:02:11	flat map entri set valu overwrit entri valu	flat mapampaposs entri object overwrit entryam...	COLLECTIONS	0.0	flat map entri set valu overwrit entri valufla...
214	[org.apache.commons.collections.testextendedpr...	NaN	2006-07-18 22:44:33	extend properti field includ nonstat	the field includ extend properti current insta...	COLLECTIONS	0.0	extend properti field includ nonstatthe field ...

Figure 4.4 Sample Processed Bug Reports DataFrame[7]

4.3 Measurements

4.3.1 Preparing for comparing the file path

The format of file path saved in the source code files is different with bug reports. Since the file path will be used to evaluate the effectiveness of the bug locator, the file paths have to be consistent. For instance: “gitrepo\src\java\org\apache\commons\collection… “ is transformed into “.gitrepo.src.java.org.apache.commons.collection…”.

4.3.2 Method 1

Comparing the similarity between each query and each source code file by using TF.IDF method and getting a similarity matrix by using the method cosine_similarity(). The code segment is shown below:

```
vectorizer = TfidfVectorizer().fit(select_source_codes['processed_code'])
bug_mat = vectorizer.transform(select_bug_reports['query'])
src_mat = vectorizer.transform(select_source_codes['processed_code'])
sim = cosine_similarity(bug_mat, src_mat)
```

Figure 4.5 Code Segment in Method 1

4.3.3 Method 2

Comparing the similarity between each query and each source code file by using LSA method and getting a similarity matrix by using a “naive” algorithm that uses ARPACK as an eigensolver on X * X.T or X.T * X[9]. The code segment is shown below:

```
bug_mat = CountVectorizer().fit_transform(select_bug_reports['query'])
src_mat = CountVectorizer().fit_transform(select_source_codes['processed_code'])
# fit lsa
lsa = TruncatedSVD(2, algorithm = 'randomized')
bug_lsa = lsa.fit_transform(bug_mat)
src_lsa = lsa.fit_transform(src_mat)
sim = np.asarray(numpy.asmatrix(bug_lsa) * numpy.asmatrix(src_lsa).T)
```

Figure 4.6 Code Segment in Method 2

4.4 Research questions

The research questions in this project includes study of the processes of bug localization and the topic of information retrieval methods, including the required method TF-IDF and other methods used in existing bug localization methods such as Smoothed Unigram Model(SUM), Vector Space Model (VSM) and Latent Semantic Analysis(LSA). Furthermore, other research topics include the evaluation metrics Mean Reciprocal Rank(MRR) and Mean Average Precision(MAP). After researching all alternative methods and evaluation metrics, the LSA method and MAP metrics are picked to be implemented in this project

4.5 Results

By following the theoretical process of MAP calculations mentioned in the previous section 3.3. The result tables of method 1 and 2 are shown below:

	Project	MAP
0	SPR	0.082821
1	SEC	0.000000
2	LDAP	0.513208
3	DATACMNS	0.289241
4	COLLECTIONS	0.821920
5	DATAMONGO	0.000000
6	DATAREST	0.623990
7	SOCIALFB	0.188889
8	LANG	0.000000
9	CONFIGURATION	0.755054
10	IO	0.990293
11	ELY	0.000000

Table 4.7 MAP Result Table of Method 1

	Project	MAP
0	SPR	0.000000
1	SEC	0.000000
2	LDAP	0.653774
3	DATACMNS	0.261603
4	COLLECTIONS	0.329167
5	DATAMONGO	0.000000
6	DATAREST	0.000000
7	SOCIALFB	0.000000
8	LANG	0.000000
9	CONFIGURATION	0.318797
10	IO	0.271612
11	ELY	0.000000

Figure 4.7 MAP Result Table of Method 2

5 Discussion on the Results

From Table 4.7, the Map values are non-zero for 8 out of 12 projects, which means for 66.67% (8/12) projects, the bug locator found at least 1 buggy file correctly in the top 10 relevant source code files. 5 of the MAP values are over 0.5 from the 12 projects which means for 41.67%(5/12) projects, they keep high bug locating accuracy by using method 1. For method 2, there are only 5 MAP values that are non-zero, and only 1 of them (8.33% projects) is over 0.5. This means the implementation of bug localization by using method2 is not very accurate.

5.1 Analysis and improvement suggestions

The LSA method was picked because it not only considers the term frequency but also the polysemy and synonymy. For the LSA method, the term count/tf-idf matrices can be used before using TruncatedSVD. Since method 1 already covers the tf-idf method, in order to distinguish between two methods, the implementation of method 2 actually uses CountVectorizer before using TruncatedSVD. It can be the reason that the LSA method is not outperforming method 1.
Another improvement can be using another way to stem words when preprocessing the query and source code. Because sometimes stemming a word by using stemmer will result in having a not meaningful word but sharing the most possible word. By considering the term meaning in comparing the similarities, lemmatization could be another method to improve the accuracy of bug localization.

6 Related Work

- Latent Semantic Analysis Example[10]: an example of implementation of textual similarity comparison by using LSA.
- Stemmer vs Lemmatization[11]: stemmer can get the word shared with most possible words but may not be very meaningful; lemmatization gets the word based on the English language, it may end up with a bigger word corpus than stemmer.
- Implementation of Lemmatization with NLTK[12]: The code segment imports WordNetLemmatizer from nltk.stem.
- Dimensionality reduction using truncated SVD[9]: The key concept of LSA implementation
- Polysemy and synonymy[4]
- MRR (Mean Reciprocal Rank)[5]: An alternative evaluation metric. The mean reciprocal rank is the average of the reciprocal ranks of results of a set of queries.
- Top N Rank[13]: An alternative evaluation metric. The number of bugs whose associated files are ranked in the top N (N= 1, 5, 10) of the returned results.
- N-gram Models[14]: An alternative IR method, it does not only tokenize the text as unigrams, but also consider the word combination, it can be more useful for some particular sentences.
- Vector Space Model[15]: An alternative IR method, the performance of VSM model in bug localization is worse than Smoothed Unigram Model(SUM) but better than Latent Dirichlet Allocation(LDA) and Latent Semantic Indexing(LSI).
- Latent Dirichlet Allocation[16]: An alternative IR method, it extracts the latent topics from a collection of documents.

7 Conclusion

In this project, the key steps have been completed, which include reading papers and articles related to bug localization and textual similarity, studying for the terminologies and examples, researching on the similar solutions that are already excited, implementing two IRFL methods on the Bench4ML dataset, and analyzing the result by calculating the MAP scores. Finally suggest better approaches and compare the different methods.

References

[1] Information referred from SlideShare: Potential Biases in Bug Localization: Do They Matter? [Online] Available: https://www.slideshare.net/pavneetkochhar/potential-biases-in-bug-localization-do-they-matter. [Accessed on Dec.13, 2021]

[2] Information referred from SlideShare: Potential Biases in Bug Localization: Do They Matter? Slide 13 [Online] Available: https://www.slideshare.net/pavneetkochhar/potential-biases-in-bug-localization-do-they-matter. [Accessed on Dec.13, 2021]

[3] Information referred from University of Calgary: ENSF 544 Lecture 14 Slides [Online] [Accessed on Dec.13, 2021]

[4] Information referred from Quora: Five Examples of Polysemy. [Online] Available: https://www.quora.com/What-are-five-examples-of-polysemy. [Accessed on Dec.13, 2021]

[5] Information referred from Singapore Management University: Where should the bugs be fixed? More accurate information retrieval-based bug localization based reports [Online] Available: https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=2530&context=sis_research. [Accessed on Dec.13, 2021]

[6] Information referred from Github [Online] Available: https://github.com/exatoa/Bench4BL. [Accessed on Dec.13, 2021]

[7] Information referred from ENSF544 Final Project: method 1 python notebook by JIaxuan Tong [Accessed on Dec.13, 2021]

[8] Information referred from ENSF544 Final Project: java_keywords.txt by JIaxuan Tong [Accessed on Dec.13, 2021]

[9] Information referred from SkLearn [Online] Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. [Accessed on Dec.13, 2021]

[10] Information referred from Datascienceassn [Online] Available: https://www.datascienceassn.org/sites/default/files/users/user1/lsa_presentation_final.pdf. [Accessed on Dec.15, 2021]

[11] Information referred from stackoverflow [Online] Available: https://stackoverflow.com/questions/1787110/what-is-the-difference-between-lemmatization-vs-stemming. [Accessed on Dec.15, 2021]

[12] Information referred from Geeksforgeeks [Online] Available: https://www.geeksforgeeks.org/python-lemmatization-with-nltk/. [Accessed on Dec.16, 2021]

[13] Information referred from [Online] Available: https://lkpy.readthedocs.io/en/stable/evaluation/topn-metrics.html. [Accessed on Dec.16, 2021]

[14] Information referred from Cornell [Online] Available: http://www.cs.cornell.edu/courses/cs4740/2014sp/lectures/smoothing+backoff.pdf. [Accessed on Dec.15, 2021]

[15] Information referred from Machinelearningmastery [Online] Available: https://machinelearningmastery.com/a-gentle-introduction-to-vector-space-models/. [Accessed on Dec.15, 2021]

[16] Information referred from Towardsdatascience [Online] Available: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2. [Accessed on Dec.15, 2021]