

Rank and Classification of Bug Reports and Feature Evaluation Using SVM

Gomathi S¹, Swathini K², Suvetha M³, Deepa M⁴

¹Assistant Professor, Department of Computer Science and Engineering, Sri Krishna College of Technology, Coimbatore

^{2, 3, 4}UG Scholar, Department of Computer Science and Engineering, Sri Krishna College of Technology, Coimbatore

Abstract: When a new bug report is received, developers usually need to procreate the bug and perform code reviews to find the origination, this process can be tedious and time consuming. A instrument for ranking all the source files with regards to how likely they are to contain the cause of the bug would enable developers to narrow down their search and better result. This paper introduce an adaptive ranking approach that leverages project acquaintance through functional decomposition of source code, API metaphors of library components, the bug-fixing record, the code change record, and the file dependency graph. Given a bug report, the ranking evaluation of each source file is computed as a weighted collection of an array of feature film, where the weights are trained mechanically on previously resolved bug reports using a learning-to-rank technique. We estimate the ranking system on six large scale open source Java projects, using the earlier-fix version of the project for every bug report. The explore results show the Support Vector Machine approach that outperforms in defect prediction.

Keywords: SVM, Bug Localization, Ranking, Information Retrieval

1. Introduction

Software is more than just a program code. A program is an executable code, which suffice some computational reason. Software is measured to be collection of executable programming code, related libraries and documentations. Software, when made for a particular necessity is called software product. A software bug is a coding mistake that may cause an unplanned or unexpected action of the software component. Upon discovering an uneven manners of the software project, document will be reported by developer or user, called a bug report. A bug report supply information that could help in repair a bug, with the overall aim of improving the software quality. A ample number of bug reports could be opened during the development life-cycle of a software product. In a software team, bug reports are extensively used by both managers and developers in their daily development process.

A developer who is appointed a bug report usually needs to reproduce the irregular behavior and perform code reviews in order to find the cause. However, the variety and uneven quality of bug reports can make this process nontrivial. Essential information is often missing from a bug report. If the bug report is understand as a query and the source code files in the software repository are viewed as a group of documents, then the problem of discovering source files that are relevant for a given bug report can be formed as a standard task in information retrieval (IR).

2. Literature Review

Locating bugs is essential, hard, and high-priced, particularly for large-scale environments. To address this, natural language information retrieval techniques are increasingly being used to propose potential faulty source files given bug reports. Our key insight is that structured information retrieval based on code constructs like class and method names, enables more surgical bug localization. We

present BLUIR (Bug Localization Using information Retrieval), which embodies this insight, requires only the source code and bug reports, and takes benefit of bug similarity data if available.

Developers often learn to use APIs (Application Programming Interfaces) by focusing at existent examples of API usage. Code repositories contains instances of such usage of APIs. Nevertheless, usual information retrieval techniques fail to do well in retrieving API usage examples from code repositories. This paper presents SSI (Structural Semantic Indexing), which is a technique to associate words to source code entities based on sameness of API usage. The formula following this technique is that entities (classes, methods, etc.) that show parallel uses of APIs are semantically related because they do similar things.

3. Mapping Bug Reports to Relevant Files

We posture to approach it as a ranking problem, in which the source files are ranked with respect to their relevance to a given bug report. Here, relevance is equated with the likeliness that a particular source file contains the basis of the bug represented in the bug report. The ranking task is defined as a weighted combination of features, where the features depict to a great extent on knowledge specific to the software engineering domain in order to assess relevant relation among the bug reports and the source code files. While a bug report may share textual tokens with its relevant source files, in generic there is a significant inherent counterpart between the natural language hired in the bug report and the programming language used in the code. We initiate a learning-to-rank approach that emulates the bug finding process employed by developers. The ranking model qualify useful relationships between a bug report and source code files by leveraging domain acquaintance, such as API specifications, the syntactic structure of code, or issue tracking data. The adaptive ranking approach is commonly applicable to software projects for which there exists a

adequate amount of project specific knowledge, such as a comprehensive API documentation and an initial number of previously fixed bug reports. To determine a bug, developers use not only the content of the bug report but also domain knowledge relevant to the software project. Furthermore, the proposed ranking model exceed three recent state-of-the-art approaches. Feature selection experiments employing greedy backward feature eliminations, it shows that all features are useful. When united with runtime analysis, the feature selection results can be utilized to pick a subset of features in order to conquer a target trade-off between system accuracy and runtime complexity. Furthermore, the ranking performance can benefit from instructive bug reports and well documented code major to a better lexical similarity, and from source code files that already have a bug-fixing history. Correspondence the bugs to the relevant files is a paper based documents, it did not have the clear idea about the bug. This system uses only less ranking models. It consumes large amount of time for reproduce the problem.

4. Support Vector Machine (SVM)

In machine learning, support vector machines (SVMs, also support vector networks[4]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression investigation. Given a set of training examples, each noticeable as belonging to one or the other of two categories, an Support Vector Machine training algorithm constructs a model that allocates new examples to one collection or the other, building it a non-probabilistic binary linear classifier. An SVM model is a sign of the examples as points in space, plotted so that the examples of the separate collection are divided by a clear gap that is as wide as possible. In our work, we will leveraging further types of domain knowledge, such as the stack traces submitted with bug reports and the file change history, as well as features previously used in fault prediction systems. We also plan to use the ranking SVM with nonlinear kernels and further measure the approach on projects in other programming languages. Using Support Vector Machine algorithm we can easily find out the bug and it can be quickly employed by the developers. This system uses more ranking methods and it takes less time for processing.

When data are not labeled, supervised learning is not probable, and an unsupervised learning approach is required, which attempts to find earthy clustering of the data to groups, and then plot new data to these groups. The clustering algorithm provides an betterment to the SVM is recognized as support vector clustering[2] and is repeatedly used in industrial applications either when data are not labeled or when only some data are labeled as a preprocessing for a classification pass.

4.1 Preprocessing

The first step towards managing and inspect textual data formats in general is to consider the text based information available in free formatted text documents or text. Initially the pre-processing is done by the following process. *Removing Stop words and Stem words*[3]. The first step is to remove the un-necessary information available in the sentence of stop words. These include few verbs,

conjunctions, disjunctions and pronouns, etc. (e.g. is, am, the, of, an, we, our) and Stemming words e.g. 'deliver', 'delivering' and 'delivered' are stemmed to 'deliver'.

4.2 Collaborative Filtering

In collaborative filtering process, if previously fixed bug reports are textually similar with the current bug report, then the files that have been associated with the similar reports may also be relevant for the current report[6]. It has been observed in that a file that has been fixed before may be responsible for similar bugs. The feature computes the textual similarity between the text of the current bug report and the summaries of all the bug reports there is not much historical information that can be used for computing features that are based on collaborative filtering or the file revisal history. In particular, there is fewer possibility for exploiting duplicated bug reports.

4.3 Bug Report Analysis

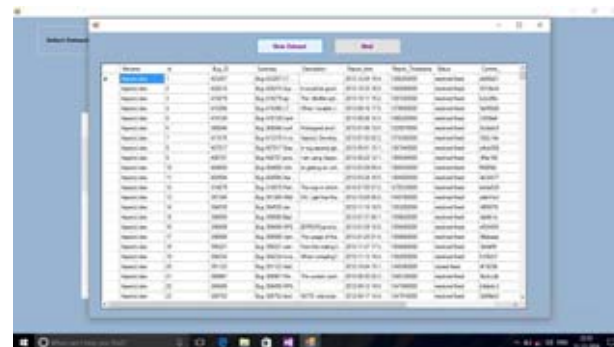
In general there are seven types of bugs are accessible, Functionality Errors, Communication Errors, Missing command errors, Syntactic Error, Error handling errors, Calculation Errors and Control flow errors. The bug reports can be analyzed for which kind of bugs[5].

4.4 Ranking

The resultant ranking function is a linear collection of features, whose weights are automatically trained on previously solved bug reports using a learning-to-rank technique[8]. The ranking approach to the problem of mapping source files to bug reports that enables the unseamed integration of a wide diversity of features; exploiting before fixed bug reports as training examples for the planned ranking model in conjunction with a learning-to-rank technique.

4.5 Classification Result

In this classification, the bug reports has been analyzed and classified into which categories[4]. The model of the training task as a classification in which bug reports and files are assigned to multiple topics, we directly train our framework for ranking, which we believe is a better match for the way the model is used.



ID	Title	Description	Status	Date
1	NullPointerException	NullPointerException at line 10	Open	2017-01-01
2	NullPointerException	NullPointerException at line 10	Open	2017-01-01
3	NullPointerException	NullPointerException at line 10	Open	2017-01-01
4	NullPointerException	NullPointerException at line 10	Open	2017-01-01
5	NullPointerException	NullPointerException at line 10	Open	2017-01-01
6	NullPointerException	NullPointerException at line 10	Open	2017-01-01
7	NullPointerException	NullPointerException at line 10	Open	2017-01-01
8	NullPointerException	NullPointerException at line 10	Open	2017-01-01
9	NullPointerException	NullPointerException at line 10	Open	2017-01-01
10	NullPointerException	NullPointerException at line 10	Open	2017-01-01

Figure 1: Bug Reports(Dataset)

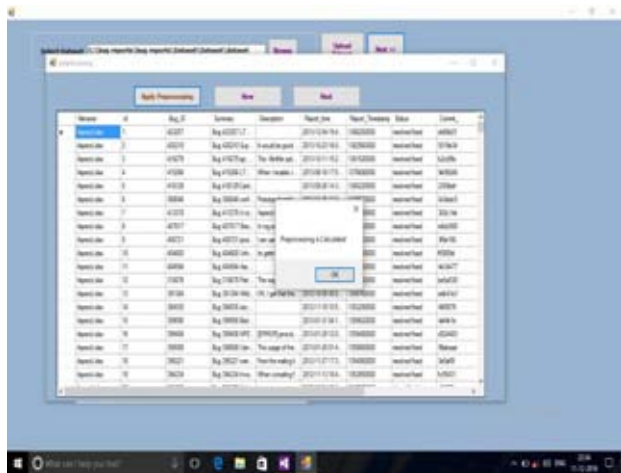


Figure 2: Preprocessing

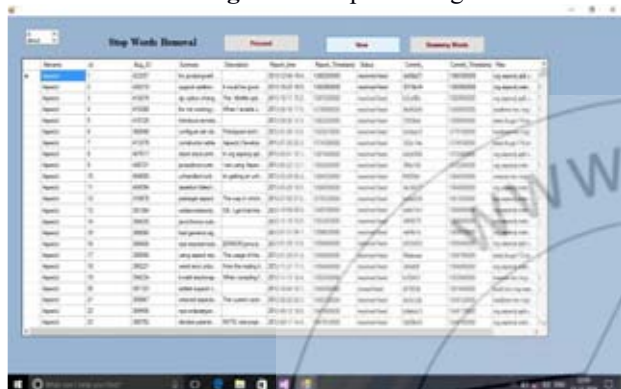


Figure 3: Stop words removal

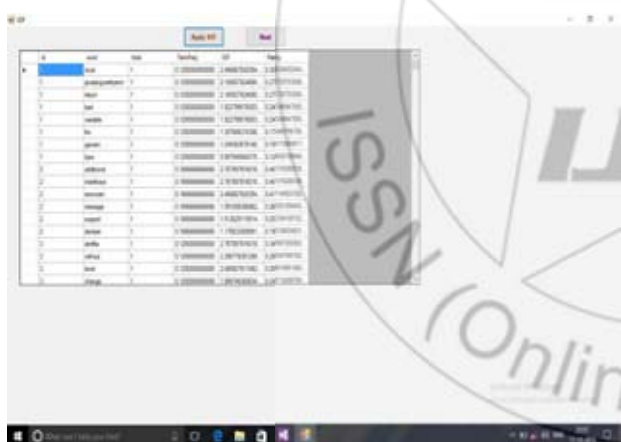


Figure 4: Calculating Term Frequency

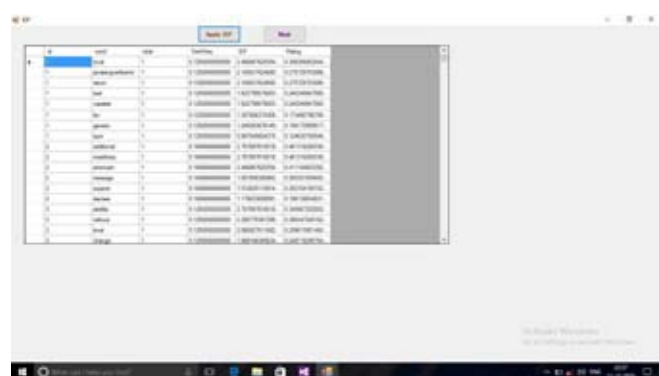


Figure 5: Calculating IDF

5. Conclusion

To find out a bug, developers use not only the content of the bug report but also field knowledge applicable to the software project. We introduce a learning-to-rank approach that emulates the bug discovering process affianced by developers. The ranking model qualify useful relationships between a bug report and source code files by leveraging field knowledge, such as API description, the syntactic artifact of code, or issue tracking data. Experimental measure on six Java projects show that our approach can find out the relevant files within the top 10 recommendations for over 70 percentage of the bug reports in Eclipse Platform and Tomcat. Moreover, the proposed ranking model outperforms three recent state-of-the-art approaches. Feature valuation research employing greedy backward feature elimination demonstrate that all properties are useful. When coupled with runtime analysis, the feature valuation results can be utilized to select a subset of properties in order to achieve a target trade-off between system accuracy and runtime complexity.

Reference

- [1] Y. Brun and M. D. Ernst, "Finding latent code errors via machine learning over program executions," in Proc. 26th Int. Conf. Softw. Eng., Washington, DC, USA, 2004, pp. 480–490.
- [2] B. Dit, A. Holtzhauer, D. Poshyanyk, and H. Kagdi, "A dataset from change history to support evaluation of software maintenance tasks," in Proc. 10th Working Conf. Mining Softw. Repositories, Piscataway, NJ, USA, 2013, pp. 131–134.
- [3] E. Enslin, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in Proc. 6th IEEE Int. Working Conf. Mining Softw. Repositories, Washington, DC, USA, 2009, pp. 71–80.
- [4] T. Joachims, "Training linear SVMs in linear time," in Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, 2006, pp. 217–226.
- [5] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or Buggy?" IEEE Trans. Softw. Eng., vol. 34, no. 2, pp. 181–196, Mar. 2008.
- [6] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in Proc. 8th Working Conf. Mining Softw. Repositories, New York, NY, USA, 2011, pp. 43–52.
- [7] R. Saha, M. Lease, S. Khurshid, and D. Perry, "Improving bug localization using structured information retrieval," in Proc. IEEE/ ACM 28th Int. Conf. Autom. Softw. Eng., Nov. 2013, pp. 345–355.
- [8] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2014, pp. 689–699.

Author Profile



Ms. S. Gomathi is currently working as Assistant Professor in Department of Computer Science and Engineering. Her area of interest is Neural Networks, Data Mining.



Ms. K. Swathini is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Data Structures.



Ms. M. Suvetha is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Database Management System.



Ms. M. Deepa is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Database Management System.

