ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

A Ranking Model for Instance and Feature Selection Techniques for Improving Bug Localization

C.S.Deepthi[1], P.Gangadhara[2]

M.Tech Student, Dept. of CSE, Shri Shiridi Sai Institute of Science and Engineering, Affiliated to JNTUA, India[1]

Assistant Professor, Dept. of CSE, Shri Shiridi Sai Institute of Science and Engineering, Affiliated to JNTUA, India[2]

ABSTRACT: Once a novel bug report is received, developers generally have to be compelled to be compelled to breed the bug and perform code reviews to hunt out the cause, a way that will be tedious and time overwhelming. A tool for ranking all the provision files with relation to but in all probability they are to contain the rationale for the bug would modify developers to slender down their search and improve productivity. This paper introduces associate degree adaptive ranking approach that leverages project data through purposeful decomposition of computer code computer file, API descriptions of library parts, the bug-fixing history, the code modification history, and so the file dependency graph. Given a bug report, the ranking score of each offer file is computed as a weighted combination of associate degree array of choices, where the weights unit of measurement trained automatically on antecedently solved bug reports using a learning-to-rank technique. we've an inclination to worth the ranking system on six huge scale open offer Java comes, exploitation the before-fix version of the project for every bug report.

I. INTRODUCTION

Word illustration makes an attempt to represent aspects of word meanings. as an example, the illustration of "cell phone" might capture the facts that cellphones are electronic product, that they embrace battery and screen, that they will be accustomed chat with others, and so on. Word illustration may be a important part of the many tongue process systems as word is typically the fundamental process unit of texts. A uncomplicated approach is to represent every word as aone-hot vector, whose length is vocabulary size and only {1} dimension is 1, with all others being zero. However, one hot word illustration solely encodes the indices of words in an exceedingly vocabulary, however fails to capture made relative structure of the lexicon. to unravel this drawback, several studies represent every word as a continual, low-dimensional and real valued vector, conjointly referred to as word embeddings. Existing embedding learning approaches ar totally on the premise of spatial arrangement hypothesis [9], that states that the representations of words ar mirrored by their contexts. As a result, words with similar grammatical usages and linguistics meanings, like "hotel" and "motel", ar mapped into neighboring vectors within the embedding area. Since word embeddings capture linguistics similarities between words, they need been leveraged as inputs or additional word options for a range of tongue process tasks, as well as MT , grammar parsing , question respondent, discourse parsing , etc al. Despite the success of the context-based word embeddings in several natural language processing tasks [14], we have a tendency to argue that they're not effective enough if directly applied to sentiment analysis that is that the analysis. space targeting at extracting, analyzing and organizing the sentiment/opinion (e.g. thumbs up or thumbs down) of texts. the foremost major problem of context-based embedding learning algorithms is that they solely model the contexts of words however ignore the sentiment data of text. As a result, words with opposite polarity, like smart and unhealthy, ar mapped into shut vectors within the embedding area. this can be important for a few tasks like pos-tagging [18] as a result of the 2 words have similar usages and grammatical roles. However, it becomes a disaster for sentiment analysis as they need opposite sentiment polarity labels

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

II. RELATED WORK

1. Title: Feature identification: A novel approach and a case study

Author: G. Antoniol and Y.-G. Gueheneuc,
Feature identification may be a well-known technique to spot subsets of a program ASCII text file activated once workout a practicality. many approaches are projected to spot options. we have a tendency to gift associate degree approach to feature identification and comparison for giant object-oriented multi-threaded programs victimisation each static and dynamic knowledge. we have a tendency to use processor emulation, information filtering, and probabilistic ranking to beat the difficulties of collection dynamic knowledge, i.e., impreciseness and noise. we have a tendency to use model transformations to match and to visualise known options. we have a tendency to compare our approach with a naive approach and a thought analysis-based approach employing a case study on a real-life giant object-oriented multi-threaded program, Mozilla, to indicate the benefits of our approach. we have a tendency to conjointly use the case study to match processor emulation with applied mathematics identification.

2. Title: Feature identification: An epidemiological metaphor,

Author: G. Antoniol and Y.-G. Gueheneuc,
Feature identification may be a technique to spot the ASCII text file constructs activated once workout one among the options of a program. we have a tendency to propose new applied mathematics analyses of static and dynamic information to accurately establish options in giant multithreaded object-oriented programs. we have a tendency to draw inspiration from medical specialty to enhance previous approaches to feature identification ANd develop an medical specialty figure of speech. we have a tendency to build our figure of speech on our previous approach to feature identification, during which we have a tendency to use processor emulation, knowledge-based filtering, probabilistic ranking, and meta modeling. we stock out 3 case studies to assess the quality of our figure of speech, victimisation the "save a bookmark" feature of net browsers as AN illustration. within the 1st case study, we have a tendency to compare our approach with 3 previous approaches (a naive approach, a plan analysis-based approach, and our previous probabilistic approach) in characteristic the feature in MOZILLA, a large, real-life, multithreaded object-oriented program. within the second case study, we have a tendency to compare the implementation of the feature within the FIREFOX and MOZILLA net browsers. within the third case study, we have a tendency to establish identical feature in 2 additional net browsers, Chimera (in C) and ICEBrowser (in Java), and another feature in JHOTDRAW and XFIG, to focus on the generalizability of our figure of speech

3. Title: Debugadvisor: A recommender system for debugging,

Author: B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala,

In giant software package development comes, once a computer user is assigned a bug to repair, she usually spends lots of your time looking (in associate ad-hoc manner) for instances from the past wherever similar bugs are debugged, analyzed and resolved. Systematic search tools that enable the computer user to specific the context of the present bug, and search through various information repositories related to giant comes will greatly improve the productivity of debugging This paper presents the planning, implementation and knowledge from such a quest tool known as DebugAdvisor.

4. Expectations, outcomes, and challenges of modern code review Author: A. Bacchelli and C. Bird,

Code review could be a common software system engineering apply used each in open supply and industrial contexts. Review these days is a smaller amount formal and additional “lightweight” than the code inspections performed and studied within the 70s and 80s. we have a tendency to through

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

empirical observation explore the motivations, challenges, and outcomes of tool-based code reviews. we have a tendency to determined, interviewed, and surveyed developers and managers and manually classified many review comments across numerous groups at Microsoft. Our study reveals that whereas finding defects remains the most motivation for review, reviews ar less regarding defects than expected and instead give extra edges like information transfer, inflated team awareness, and creation of other solutions to issues.

5. Title: Leveraging usage similarity for effective retrieval of examples in code repositories,

Author: S. K. Bajracharya, J. Ossher, and C. V. Lopes,

Developers usually learn to use arthropod genus (Application Programming Interfaces) by observing existing samples of API usage. Code repositories contain several instances of such usage of arthropod genus. However, typical info retrieval techniques fail to perform well in retrieving API usage examples from code repositories. This paper presents Structural linguistics compartmentalisation (SSI), a way to associate words to ASCII text file entities supported similarities of API usage. The heuristic behind this method is that entities (classes, methods, etc.) that show similar uses of arthropod genus ar semantically connected as a result of they are doing similar things. we have a tendency to appraise the effectiveness of SSI in code retrieval by scrutiny 3 SSI primarily based retrieval schemes with 2 typical baseline schemes. we have a tendency to appraise the performance of the retrieval schemes by running a group of twenty candidate queries against a repository containing 222,397 ASCII text file entities from 346 jars happiness to the Eclipse framework. The results of the analysis show that SSI is effective in up the retrieval of examples in code repositories.

III. EXISTING SYSTEM

Recently, researchers have developed methods that concentrate on ranking source files for given bug reports automatically.
Saha et al. syntactically parse the source code into four document fields: class, method, variable, and comment. The summary and the description of a bug report are considered as two query fields.
Kim et al. propose both a one-phase and a two-phase prediction model to recommend files to fix. In the one-phase model, they create features from textual information and metadata (e.g., version, platform, priority, etc.) of bug reports, apply Na€ıve Bayes to train the model using previously fixed files as classification labels, and then use the trained model to assign multiple source files to a bug report.
Rao and Kak apply various IR models to measure the textual similarity between the bug report and a fragment of a source file

DISADVANTAGES OF EXISTING SYSTEM:
Their one-phase model uses only previously fixed files as labels in the training process, and therefore cannot be used to recommend files that have not been fixed before when being presented with a new bug report.
Existing methods require runtime executions..

IV. PROPOSED SYSTEM

The main contributions of this paper include: a ranking approach to the problem of mapping source files to bug reports that enables the seamless integration of a wide diversity of features; exploiting previously fixed bug reports as training examples for the proposed ranking model in conjunction with a learning-to-rank technique; using the file dependency graph to define features that capture a measure of code complexity; fine-grained benchmark datasets created by checking out a before-fix version of the source code package for each bug report; extensive evaluation and comparisons with existing state-of-the-art methods; and a thorough evaluation of the impact that features have on the ranking accuracy.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

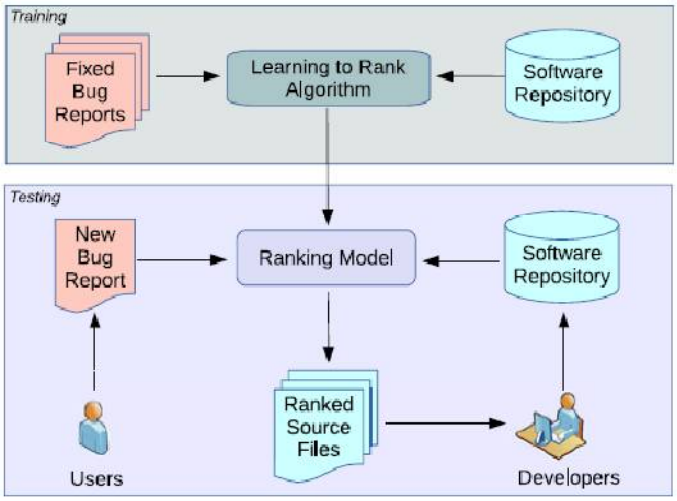
ADVANTAGES OF PROPOSED SYSTEM:

Our approach can locate the relevant files within the top 10 recommendations for over 70 percent of the bug reports in Eclipse Platform and Tomcat.

Furthermore, the proposed ranking model outperforms three recent state-of-the-art approaches.

Feature evaluation experiments employing greedy backward feature elimination demonstrate that all features are useful.

SYSTEM ARCHITECTURE



V. IMPLEMENTATION

System Construction Module:

In the first module, we develop the system with the entities required to evaluate our proposed model. When a new bug report is received, developers usually need to reproduce the bug and perform code reviews to find the cause, a process that can be tedious and time consuming. So In This paper introduces an adaptive ranking approach that leverages project knowledge through functional decomposition of source code, API descriptions of library components, the bug-fixing history, the code change history, and the file dependency graph. Given a bug report, the ranking score of each source file is computed as a weighted combination of an array of features, where the weights are trained automatically on previously solved bug reports using a learning-to-rank technique.

We propose to approach it as a ranking problem, in which the source files (documents) are ranked with respect to their relevance to a given bug report (query). In this project we apply three entities namely User, Developer, Admin. If User has an error in a source code then user send the error message to the Admin. Then Admin analysis the errors and ranking the reports and send to the Developers. And Developers find the solutions of the errors**.**

Ranking Function:

The ranking function is defined as a weighted combination of features, where the features draw heavily on knowledge specific to the software engineering domain in order to measure relevant relationships between the bug report and the source code file. While a bug report may share textual tokens with its relevant source files, in general there is a significant inherent mismatch between the natural language employed in the bug report and the programming language used in the code.

Ranking methods that are based on simple lexical matching scores have suboptimal performance, in part due to lexical mismatches between natural language statements in bug reports and technical terms in software systems. Our system contains features that bridge the corresponding lexical gap by using project specific API documentation to connect natural language terms in the bug report with programming language constructs in the code.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

Source Code files may contain a large number of methods of which only a small number may be causing the bug. Correspondingly, the source code is syntactically parsed into methods and the features are designed to exploit method level measures of relevance for a bug report. It has been previously observed that software process metrics (e.g., change history) are more important than code metrics (e.g., size of codes) in detecting defects.

Feature Representation:

The proposed ranking model requires that a bug report - source file pair (r,s) be represented as a vector of k features. We distinguish between two major categories of features.

Query dependent: These are features that depend on both the bug report r and the source code file s. A query dependent feature represents a specific relationship between the bug report and the source file, and thus may be useful in determining directly whether the source code file s contains a bug that is relevant for the bug report r.

Query independent. These are features that depend only on the source code file, i.e., their computation does not require knowledge of the bug report query. As such, query independent features may be used to estimate the likelihood that a source code file contains a bug, irrespective of the bug report.

We hypothesize that both types of features are useful when combined in an overall ranking model.

Collaborative Filtering Score:

It has been observed in that a file that has been fixed before may be responsible for similar bugs. This collaborative filtering effect has been used before in other domains to improve the accuracy of recommender systems, consequently it is expected to be beneficial in our retrieval setting, too. Given a bug report r and a source code file s, let br(r , s) be the set of bug reports for which file s was fixed before r was reported. The feature computes the textual similarity between the text of the current bug report r and the summaries of all the bug reports in br(r , s). This feature is query-dependent.

In a Class Name Similarity a bug report may directly mention a class name in the summary, which provides a useful signal that the corresponding source file implementing that class may be relevant for the bug report. Our hypothesis is that the signal becomes stronger when the class name is longer and thus more specific.

In a File Revision History the source code change history provides information that can help predict fault-prone files. For example, a source code file that was fixed very recently is more likely to still contain bugs than a file that was last fixed long time in the past, or never fixed..

VI. CONCLUSION AND FUTURE WORK

To find a bug, developers use not solely the content of the bug report however additionally domain data relevant to the package project. we have a tendency to introduced a learning-to-rank approach that emulates the bug finding method used by developers. The ranking model characterizes helpful relationships between a bug report and ASCII text file files by investing domain data, like API specifications, the syntactical structure of code, or issue chase knowledge. Experimental evaluations on six Java comes show that our approach will find the relevant files at intervals the highest ten recommendations for over seventy % of the bug reports in Eclipse Platform and Felis catus. what is more, the projected ranking model outperforms 3 recent progressive approaches. Feature analysis experiments using greedy backward feature elimination demonstrate that every one options ar helpful. once plus runtime analysis, the feature analysis results is used to pick out a set of options so as to attain a target trade-off between system accuracy and runtime complexness. The projected adaptational ranking approach is mostly applicable to package comes that there exists a sufficient quantity of project specific data, like a comprehensive API documentation (Section three.1.2) associated an initial range of antecedently mounted bug reports (Section half-dozen.1). what is more, the ranking performance will get pleasure from informative bug reports and well documented code resulting in a higher lexical similarity (Section three.1.1), and from ASCII text file files that have already got a bug-fixing history (Section three.2). In future work, we are going to leverage further sorts of domain data, like the stack traces submitted with bug reports and also the file amendment history, likewise as options antecedently utilized in defect prediction systems. we have a tendency to additionally arrange to use the ranking SVM with nonlinear kernels and more assess the approach on comes in alternative programming languages.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

REFERENCES

[1] G. Antoniol and Y.-G. Gueheneuc, "Feature identification: A novel approach and a case study," in Proc. 21st IEEE Int. Conf. Softw. Maintenance,Washington, DC, USA, 2005, pp. 357–366.
[2] G. Antoniol and Y.-G. Gueheneuc, "Feature identification: An epidemiological metaphor," IEEE Trans. Softw. Eng., vol. 32, no. 9, pp. 627–641, Sep. 2006.
[3] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "Debugadvisor: A recommender system for debugging," in Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng., New York, NY, USA, 2009, pp. 373–382.
[4] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 712–721.
[5] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2010 pp. 157–166.
[6] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for bugs in all the right places," in Proc. Int. Symp. Softw. Testing Anal., New York, NY, USA, 2006, pp. 61–72.
[7] N. Bettenburg, S. Just, A. Schr€oter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2008, pp. 308–318.
[8] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in Proc. 15th Int. Conf. Softw. Eng., Los Alamitos, CA, USA, 1993, pp. 482–498.
[9] D. Binkley and D. Lawrie, "Learning to rank improves IR in SE," in Proc. IEEE Int. Conf. Softw. Maintenance Evol., Washington, DC, USA, 2014, pp. 441 445.
[10] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," J. Mach. Learn. Res., vol. 3, pp. 993–1022 Mar. 2003.
[11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in Proc. ACM Conf. Comput. Supported Cooperative Work, New York, NY, USA, 2010, pp.301–310.