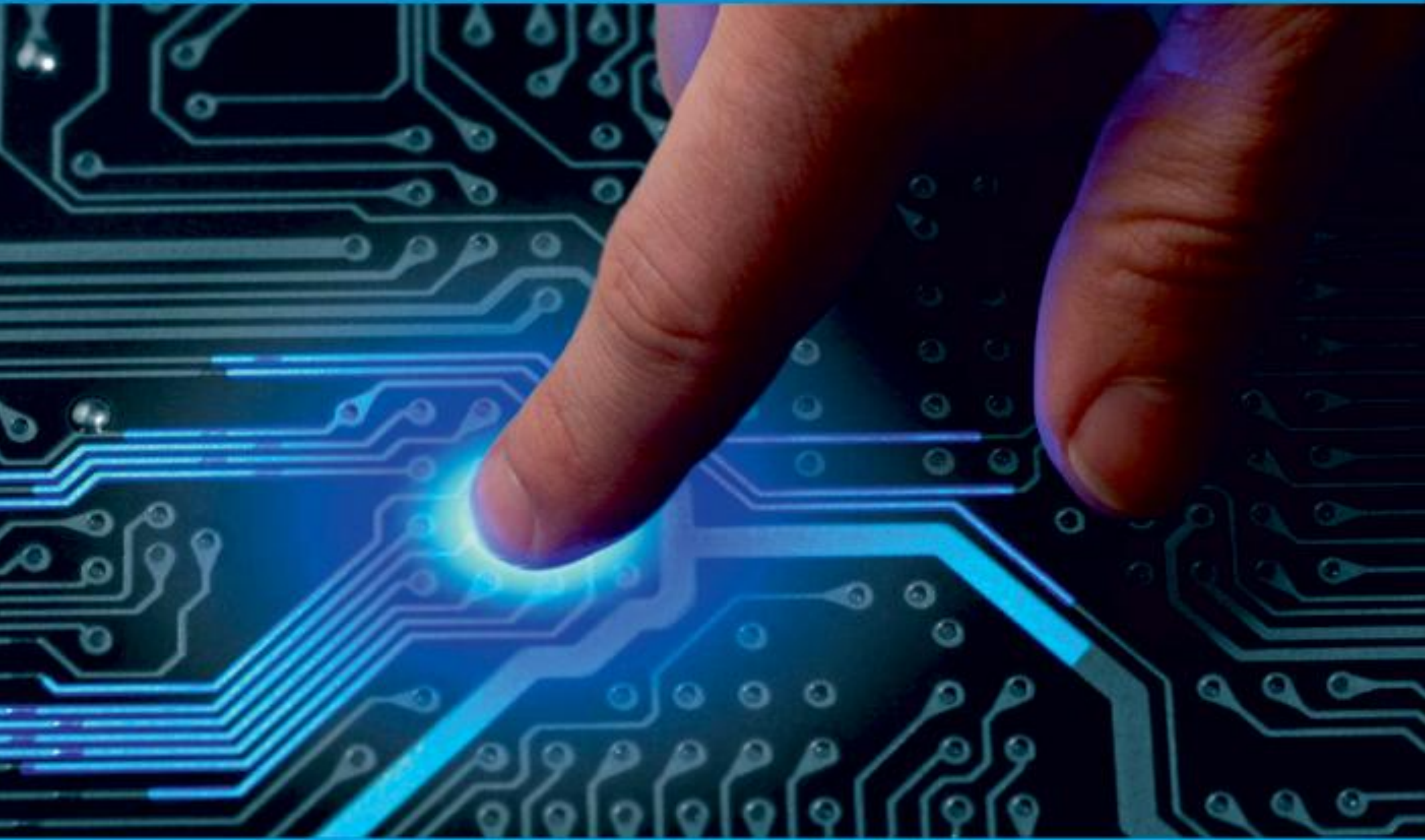
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 9, Issue 6, June 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.542

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.542 |

|| Volume 9, Issue 6, June 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0906237 |

Automated Software Bug Triage System With A Machine Learning Approach

Thilak KG, Raju K, Rakesh M, Mohammed Farzan, Dr. Ananda Babu J

UG Student, Dept. of Information Science and Engineering, Malnad College of Engineering, Hassan, Karnataka, India

UG Student, Dept. of Information Science and Engineering, Malnad College of Engineering, Hassan, Karnataka, India

UG Student, Dept. of Information Science and Engineering, Malnad College of Engineering, Hassan, Karnataka, India

UG Student, Dept. of Information Science and Engineering, Malnad College of Engineering, Hassan, Karnataka, India

Associate Professor, Dept. of Information Science and Engineering, Malnad College of Engineering, Hassan, Karnataka, India

ABSTRACT: For popular software systems, the number of daily yield bug reports is high. Triaging these incoming reports is a time-consuming task. Part of the bug triage is the assignment of a report to a developer with the appropriate expertise. In this paper, we present an approach to automatically suggest developers who have the suitable expertise for handling a bug report, based on the identified component obtained from the short description of the bug report. Our work is the first to examine the impact of multiple machine learning dimensions (classifiers and training history) along with the ranked list of developers for prediction accuracy in bug assignment. We validate our approach on Eclipse covering 2,868,000 bug reports consisting of 253 components. We demonstrate that our techniques can achieve up to 80.05% prediction accuracy in bug assignment and significantly reduce the aberrant assignment of bugs. We contrast the prediction time for our dataset using various algorithms such as Naive Bayes Text Classifier, Multinomial Naive Bayes, and Linear SVM. We concluded that SVM provides higher accuracy and less learning time.

KEYWORDS: Bug, Bug triage, Bug repository, Developer, Feature information, Folding

I. INTRODUCTION

To bug is human, to debug is divine. Software evolution has high associated costs and effort. A survey by the National Institute of Standards and Technology estimated that the annual cost of software bugs is about $59.5 billion. Some software maintenance studies indicate that maintenance costs are at least 50% and sometimes more than 90%, of the total costs associated with a software product. These surveys suggest that making the bug fixing process more efficient would reduce evolution effort and lower software production costs.

Most software projects use bug trackers to organize the bug fixing process and facilitate application maintenance. For sample, Bugzilla is a popular bug tracker used by many large projects, such as Mozilla, Eclipse, KDE, and Gnome. These applications receive hundreds of bug reports a day; ideally, each bug gets assigned to a developer who can fix it in the least amount of time. This process of assigning bugs, known as bug task, is complicated by several factors: if done manually, assignment is labor-intensive, time-consuming, and fault-prone; moreover, for open source projects, it is difficult to keep track of active developers and their professionalism. Identifying the right developer for fixing a new bug is further aggravated by growth, e.g., as projects add more components modules, developers and testers, the number of bug reports submitted daily increases, and manually recommending developers based on their professionalism becomes difficult.

Reports indicate that, on average, the Eclipse project takes about 40 days to assign a bug to the first developer, and then it takes an additional 100 days or more to reassign the bug to the second developer. Similarly, in the Mozilla project, on average, it takes 180 days for the first sample and then an additional 250 days if the first assigned developer is unable to fix it. These numbers indicate that the lack of effective, automatic assignment and toss reduction techniques results in considerably high effort associated with bug resolution. Reassigning a bug to another developer if the previous assignee is unable to resolve it is known as Bug Tossing. It can be inferred from the dataset of Eclipse that almost 90% of all "Fixed" bugs have been tossed at least once.

II. RELATED WORK

Cubranic et al. [1] were the first to propose the idea of using text classification methods (similar to methods used in machine learning) to semi-automate the process of bug assignment. They used keywords extracted from the title and description of the bug report, as well as developer IDs as attributes, and trained a Naive Bayes classifier. When presented with new bug reports, the classifier suggests one or more potential developers for fixing the bug. Their method used bug reports for Eclipse for training and reported a prediction accuracy of up to 30%. While we use classification as a part of our approach, in addition, we employ incremental learning and tossing graphs to reach higher accuracy Anvik et al. [2] improved the machine learning approach proposed by Cubranic et al. by using filters when collecting training data: (1) filtering out bug reports labeled invalid, won't fix, or works for me, (2) removing developers who no longer work on the project or do not contribute significantly, and (3) filtering developers who fixed less than 9 bugs. They used three classifiers, SVM, Naive Bayes, and C4.5. They observed that SVM (Support Vector Machines) performs better than the other two classifiers and reported prediction accuracy of up to 64Lin et al. [3] conducted machine learning-based bug assignment on a proprietary project, Soft PM. Their experiments were based on 2,576 bug reports. They report 77.64% average prediction accuracy when considering module ID (the module a bug belongs to) as an attribute for training the classifier; the accuracy drops to 63% when module ID is not used. Their finding is similar to our observation that using product-component information for classifier training improves prediction accuracy Jeong et al. [4] introduced the idea of using bug tossing graphs to predict a set of suitable developers for fixing a bug. They used classifiers and tossing graphs (Markov-model-based) to recommend potential developers. Pamela Bhattacharya et al.[5] proposed a technique for automated bug assignment using machine learning and tossing graphs. They used a classifier and a tossing graph to automatically assign a bug to a developer. Initially, they used a training data set of fixed bugs that contain information regarding the developers to whom it was assigned and the reassignment to other developers.

III. PROPOSED ALGORITHM

A. *Design Considerations:*

- Snowball Stemming Algorithm
- Naive Bayes Text Classifier Algorithm
- Multinomial Naive Bayes Algorithm
- Tf- idf Weight Calculation
- Linear Support Vector Machine
- Folding

B. *Bug Repository Maintenances*

The Bug Repository data set is a collection of models and metrics of software systems and their histories. The goal of such a data set is to allow people to compare different bug prediction approaches and to evaluate whether a new technique is an improvement over existing ones. In particular, the data set contains the data needed.

- The prediction technique is run based on source code metrics and/or historical measures and/or process information (logs data).
- Compute the performance of the prediction by comparing its results with a set, i.e., the number of post-release defects reported in the bug tracking system.

Bug prediction is performed at the class level using the designed data set. However, class data is aggregated to derive the package or subsystem information, since each class specifies the package that contains it.

B. Bug Report Analysis

An automatic bug triaging system is presented by recommending one experienced developer for each new bug report. The following steps are performed.

1. Representation Framework

We have a collection of bug reports, B = {b1, b|B|}. Each bug report has a collection of term, T = {t1… t|T|}, and a

class label (developer), c C = {c1, …, c|C|}.

2. Term selection methods

The high dimensionality of term space is reduced using term selection by selecting the most discriminating terms for classification tasks. The methods give a weight for each term in which terms with higher weights are assumed to contribute more for classification task than terms with lower weights

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.542 |

|| Volume 9, Issue 6, June 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0906237 |

3. Chi-Square (X2)

In statistics, the X2 test is used to examine the independence of two events. The events, X and Y, are assumed to independent if P (XY) = P(X)P(Y) term selection, the two events are the occurrence of the term and the occurrence of the class.

4. Term Frequency Relevance Frequency (TFRF)

The basic behind the TFRF method is that the more high frequency for a term in the positive category than in the negative category, the more contributions it makes in selecting the positive instances from negative instances.

IV. PSEUDOCODE

```
Step 1:  Consider the two bugs to be compared
Step 2:  Find the list of key Phrases in Bug A1
Step 3:  Find the list of key Phrases in Bug A2
Step 4:  Find the intersection set between List of Key Phrases in Bugs A1 and List of Key Phrases in Bugs A2
Step 5:  Find the union set between List of Key Phrases in Bugs A1 and List of Key Phrases in Bugs A2.
Step 6:  Start from index1 till the end of key phrases in the intersection set
    a)  Obtain the kth key phrase K
    b)  Measure the text frequency of Bugs A1 for  K
    c)  Measure the text frequency of Bugs A2 for K
    d)  if tf(k,A1)>=tf(k,A2) measure the intersection sum as below
        intersection sum =intersection sum +tf(k,A1)
        else
        intersection sum =  intersection sum+tf(k,A2)
    e)  k=k+1
    f)  Repeat the process from step a to step e until all tokens in the intersection set are exhausted

Step 7: Start from index1 till the end of key phrases in the union set.

    a)  Obtain the kth key phrase K
    b)  Measure the text frequency of Bugs A1 for  K
    c)  Measure the text frequency of Bugs A2 for K
    d)  if tf(k,A1)<tf(k,A2) measure the intersection sum as below
        union sum = union sum +tf(k,A1)
        else
        union sum =  union sum +tf(k,A2)
    e)  k=k+1
    f)  Repeat the process from step a to step e until all tokens in the union set are exhausted


    g)  Measure Similarity =Intersection Sum/Union Sum

Step 8: End.
```

V. SIMULATION RESULTS

The simulation studies involve We used Mozilla and Eclipse bugs to measure the accuracy of our proposed algorithm. We analyzed the entire life span of both applications. We divided our bug data sets into 10 folds and executed 9 iterations to cover all the folds. Data collection. We used the bug reports to collect four kinds of data: 1. Keywords: we collect keywords from the bug description and comments in the bug report. 2. Bug source: we retrieve the product and component the bug has been filed under from the bug report. 3. Temporal information: we collect information about when the bug has been reported and when it has been fixed. 4. Developers assigned: we collect the list of developer IDs assigned to the bug

In our experiments, we varied the size of the test set, the size of the vocabulary. The below graph shows the classification accuracy as a function of the train/test set split when the full vocabulary V of words found in bug reports is used.

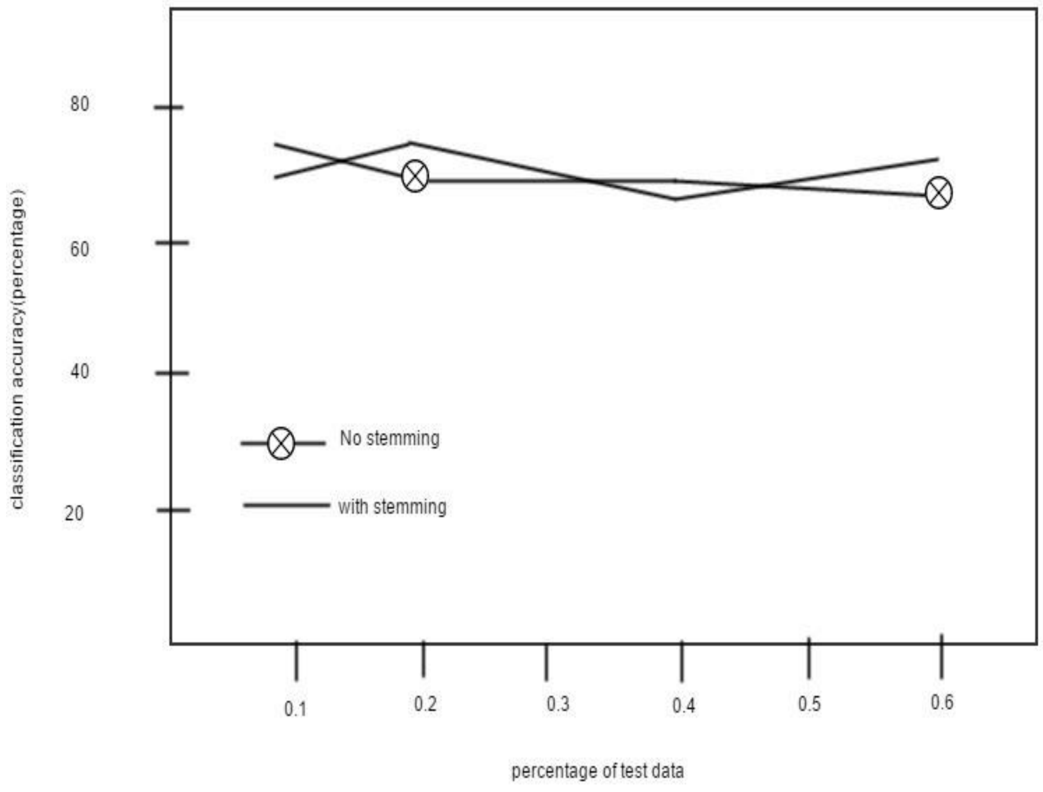
As we can see, the algorithm correctly assigns just under 75% of the bugs, when 90% of the document corpus is used as training and 10% as the test set. The accuracy slowly declines to 65% as the test sets size is increased to 50the corpus.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.542 |

|| Volume 9, Issue 6, June 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0906237 |

The graph also shows the results when the vocabulary was created using stemming, which identifies most grammatical variations of a word such as see, sees, seen, for



Example and treats them as a single term. The results are virtually unchanged, and any differences between the two conditions are within about one standard deviation at each data point.

Finally with 10 fold cross-validation accuracy of nearly 78% is achieved and with linear SVM an accuracy of 80% is achieved

VI. Conclusion and Future Work

The Machine learning and tossing graphs have proved to be promising for automating bug assignment. In this paper we lay the foundation for future work that uses machine learning techniques to improve automatic bug assignment by examining the impact of multiple machine learning dimensions learning strategy, attributes, classifiers on assignment accuracy. We used a broad range of text classifiers and found that, like many problems which use specific machine learning algorithms, we could able to select a specific classifier for the bug assignment problem. We validated our approach on two large, long-lived open-source projects; in the future, we plan to test how our current model generalizes to projects of different scale and lifespans. In particular, we would like to find if the classifier preference should change as the project evolves and how the source code familiarity of a developer could be used as an additional attribute for ranking developers. Similarly, when we assign tossing probabilities, we only consider the developer who could finally fix the bug. However, it is common that developers contribute partially to the final patch in various ways. For example, when a bug is assigned to a developer, he might provide insights and add notes to the bug report instead of fixing the bug; in fact, some contributors provide useful discussions about a bug in the

.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?” in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.

[2] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36, no. 4, pp. 474–494, Jul./Aug. 2010.

[3] J. Anvik and G. C. Murphy, “Reducing the effort of bug report triage: Recommenders for development-oriented decisions,” ACM Trans. Soft. Eng. Methodol., vol. 20, no. 3, Bugs 10, Aug. 2011.

[4] C. C. Aggarwal and P. Zhao, “Towards graphical models for text processing,” Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.

[5] Bugzilla, (2014). [Online]. Available: http://bugzilla.org/

[6] K. Balog, L. Azzopardi, and M. de Rijke, “Formal models for expert finding in enterprise corpora,” in Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, Aug. 2006, pp. 43–50.

[7] P. S. Bishnu and V. Bhattacherjee, “Software fault prediction using quad tree-based k-means clustering algorithm,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 6, pp. 1146–1150, Jun. 2012.

[8] H. Brighton and C. Mellish, “Advances in instance selection for instance-based learning algorithms,” Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.

[9] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, “Information needs in bug reports: Improving cooperation between developers and users,” in Proc. ACM Conf. Compute. Supported Cooperative Work, Feb. 2010, pp. 301–310.

[10] V. Bol_on-Canedo, N. S_anchez-Maro~no, and A. Alonso-Betanzos, "A Bug of feature selection methods on synthetic data," Knowl. Inform. Syst., vol. 34, no. 3, pp. 483–519, 2013.

[11] V. Cerver_on and F. J. Ferri, “Another move toward the minimum consistent subset: A tab search approach to the condensed nearest neighbor rule," IEEE Trans. Syst., Man, Cybern., Part B, Cybern., vol. 31, no. 3, pp. 408–413, Jun. 2001.

[12] D. _Cubrani_c and G. C. Murphy, “Automatic bug triage using text categorization,” in Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng., Jun. 2004, pp. 92–97.

[13] Eclipse. (2014). [Online]. Available: http://eclipse.org/

[14] B. Fitzgerald, “The transformation of open source software,” MIS Quart., vol. 30, no. 3, pp. 587–598, Sep. 2006.

[15] A. K. Farahat, A. Ghodsi, M. S. Kamel, “Efficient greedy feature selection for unsupervised learning,” Knowl. Inform. Syst., vol. 35, no. 2, pp. 285–310, May 2013.