ISSN:0975-9646

A survey on Reducing Features to Improve Bug Prediction by Using Cos-triage Algorithm

Veena Jadhav, Prof. Vandana Gaikwad

Computer Engineering Department,
BVDUCOE Pune-43(India)

***Abstract*- Bugs are nothing but Software defects, present a serious challenge for system consistency and dependability. It is very difficult task to predict bugs. To detect the bugs from the software bug prediction is useful way. Machine learning classifiers have emerged newly as a way to envisage the existence of a bug in a change made to a source code. The machine learning classifier is first skilled on software history data and then it is used to predict bugs. Two main drawbacks of existing classifier-based bug prediction are insufficient accuracy for practical use and deliberate prediction time because of a large number of machine learned features. In this paper we have proposed mainly two techniques cos-triage algorithm which tries to utilize both accuracy and cost of bug prediction and feature selection techniques which discard less important features until optimal classification performance is reached. Reducing the feature improve the quality of knowledge extracted and also enhance the speed of computation.**

***Keywords*— Reliability, Bug prediction, Machine learning, Feature selection, Accuracy**

I. INTRODUCTION

Machine learning Classifiers, when skilled on historical software project information, it can be used to guess the existence of a bug in an individual file-level software change, as verified in previous work by the second and fourth authors[1].First cos-triage algorithm is used to fixing bug and that record is stored in historical data or in log record and then classifier is trained on information found in historical log record and it can be used to classify a new change as being either buggy (predicted to have a bug) or clean (predicted to not have a bug).

Recently, we have formed a prototype displaying server-computed bug predictions within the Eclipse IDE [3]. A bug prediction system must also provide highly specific predictions. If software engineers are to faith a bug prediction system, it must provide a small number of false changes that are predicted to be buggy but which are very clean.[4] If large numbers of clean changes are falsely predicted to be buggy, developers won't have faith in the bug prediction

Bug prediction service must also provide accurate predictions. If engineers are to faith a bug prediction service, it must provide very few "forged alarms," changes that are predicted to be pram but which really clean [16].If as well many clean changes are incorrectly predicted to be buggy, developers will lose trust in the bug prediction system.

Kim et al.[1] developed the former change classification bug prediction approach and similar work done by Hata et al[2]. Which employ the extraction of "features" (in the machine learning sense, which differs from software features) from the history of changes made to a software project. They include everything divided by whitespace in the code that was included or excluded in a change. Thus, all variables,comment words, mathematical operators, name of methods, and programming language keywords are used as features to instruct the SVM classifier which is present in this paper.

Price of large feature set is extremely high. Because of composite interactions and noise classifiers cannot handle such a large feature set. As well as number of features increases time also increases, rising to several seconds per classification for tens of thousands of features, and minutes for large project data histories. This will affects the scalability of a bug prediction service.

This paper uses multiple feature selection techniques to develop classifier performance. Although many classification methods could be working, this paper focuses on the use of cos-triage algorithm and SVM.

This paper contributes three aspects:

1. Study of multiple feature selection techniques to classify bugs in software code changes.
2. Use of cos-triage algorithm to utilize accuracy and cost of bug prediction.

The rest of this paper is organized as follows: In Section 2 primary steps involved in performing change classification is presented. Also this section discuss about feature selection techniques in more detail. Section 3 discuss prior work. Section 4 contains system overview for proposed system. Finally a conclusion is made in section 5.

II.CHANGE CLASSIFICATION

Following steps are concerned in performing change classification on single project.

Creating a Corpus:

1) Change deltas are extracting from the log records of a project, as stored in its SCM repository.
2) For each file bug fix changes are recognized by examining keywords in SCM change log messages.

3) The buggy and clean changes at the file level are recognized by tracing backward in the revision log record from bug fix changes.
4) Features are extracted from all changes, which include both buggy and clean. All expressions in Complete source code contains features , the lines tailored in each change and change meta-data such as author and change time also. Complexity metrics are calculated at this step
5)Combination of wrapper and filter methods execute to calculate a reduced set of features.Gain Ratio, Chi-Squared, Significance, and Relief-F feature rankers are used by filter method. The wrapper methods are depends on the SVM classifiers.
6) classification model is skilled by using reduced set.
7) Trained classifier is set to use. Classifier, verified whether a new change is more similar to a buggy change or a clean change.

A. Finding Buggy and Clean Changes

For bug prediction training data is set and used. Mining change log records is used to discover bug introducing changes and to recognize bug fixes. There are two approaches we use: searching for keywords in log records such as “Set” “Bug” or other keywords likely to emerge in a bug fix, and searching for another references to bug.

B. Feature Extraction

Using support vector machine algorithm, a classification model must be skilled by using buggy and clean changes which is used to organize software changes.
Everything in the source code file divided by whitespace or a semicolon is used as a feature which is nothing but variable name , method name, function name, keyword, comment word, and operator.

C. Feature Selection Techniques

To perform classification large feature sets need longer training and prediction times, also need large amounts of memory. Feature selection is general solution to this problem. In which only the subset of features that are mainly useful for making classification decisions are actually used.

D. Feature Selection Process

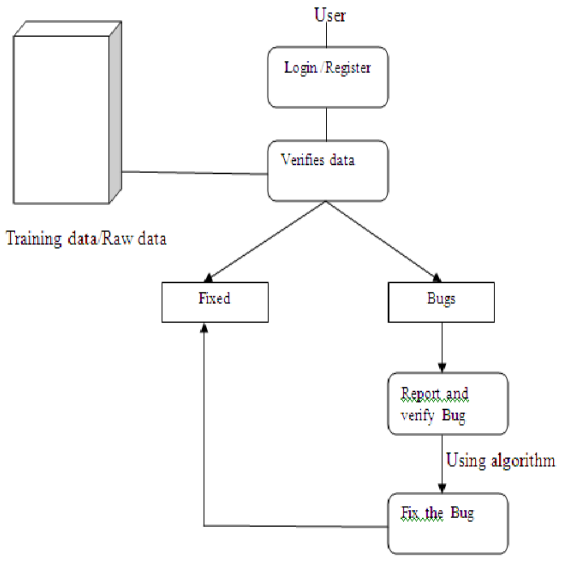
An iterative process of selecting incrementally less significant sets of features is done by using Filter and wrapper methods . This process starts by cutting the initial feature set in half which reduces memory and processing requirements for the rest of the process

A classification model is skilled by using the reduced feature set.Then classifier whether a new change is more related to a buggy change or a clean change.

III. RELATED WORK

Khoshgoftaar and Allen developed model to list modules according to software quality factors such as future fault density using stepwise multiregression [5],[6],[7]. Ostrand et al. explored the top 20 percent of problematic files in a project [11] using future fault predictors and a linear regression model.
Totally Ordered Program Units could be transformed into a partially ordered program list, e.g by presenting the top *N*% of modules as presented by Ostrand et al. Hassan and Holt presented a caching algorithm to calculate the set of fault-prone modules, called the top-10 list[9] .Kim et al.presented the bug cache algorithm to predict future faults based on preceding fault localities [10].
Gyimothy et al. [11] presented fault classes of the Mozilla project across several releases. With the help of decision trees and neural networks that utilize object-oriented metrics as features.
Hall and Holmes [12] studied six different feature selection techniques when using the Naive Bayes and the C4.5 classifier [13]. Each dataset studied has about 100 features.Many of the feature selection techniques developed by Hall and Holmes are used in this paper.
Song et al. [14] develop a general defect prediction framework which contain a data preprocessor, feature selection methods, and machine learning algorithms. They also consider that small changes to data representation can have a high impact on the results of feature selection and bug prediction.
Gao et al. [15] propose several feature selection algorithms to predict buggy software modules for a large legacy telecommunications software service. Filter-based methods and three subset selection search algorithms are used.

IV. SYSTEM DESCRIPTION



.

Figure 1. System overview

Above diagram fig 1. shows general overview of the system.

Initially user login/register into the system. It also contains training data/raw data which maintains the logs or history. Then cos-triage algorithm is applied on that registerd code.Cos-triage algorithm helps us for fixing bug or defects. Output contains some bugs. If developer wants to fix bug then he fixes it else program ends .Then at the back end that output with bug is used as input to support vector machine(SVM) classifier .SVM classifier uses different feature selection methods which is given in above section(2)..After that by reducing the features we gets final output without bug.SVM classifier works on trained data/raw data which is stored in log record.

A.UML DIAGRAM

In following section we are providing use case diagram and class diagram of system

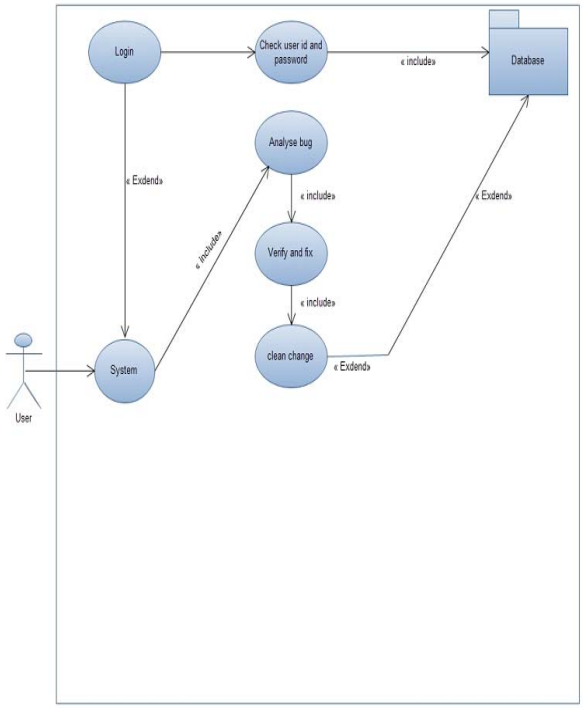


Fig 2.Usecase Diagram

V. CONCLUSION AND FUTURE WORK

This paper has implemented cos-triage algorithm which helps to exploit the cost and accuracy of bug fixing or bug prediction. This paper has also implemented the feature selection technique which reduces the number of features used by a machine learning classifier for bug prediction.

In the future, when software developers have sophisticated bug prediction technology fixed into their software development environment, the use of classifiers with feature selection will allow rapid, exact, more accurate bug predictions. Also many algorithm will come in future which will enhance precision of bug prediction.

REFERENCES

[1] S. Kim, E. W. Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?" *IEEE Trans. Software Eng.*, vol. 34, no. 2, pp. 181–196,2008.

[2] H. Hata, O. Mizuno, and T. Kikuno, "An Extension of Fault-prone Filtering using Precise Training and a Dynamic Threshold," *Proc. MSR 2008*, 2008.

[3] J. Madhavan and E. Whitehead Jr., "Predicting Buggy Changes Inside an Integrated Development Environment," Proc. OOPSLA Workshop Eclipse Technology eXchange, 2007.

[4] A.Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C.Henri-Gros, A. Kamsky, S. McPeak, and D.R. Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," Comm. ACM, vol. 53, no. 2, pp. 66-75, 2010.

[5] T. Khoshgoftaar and E. Allen, "Predicting the Order of Fault-Prone Modules in Legacy Software," *Proc. 1998 Int'l Symp. on Software Reliability Eng.*, pp. 344–353, 1998.

[6] T. Khoshgoftaar and E. Allen, "Ordering Fault-Prone Software Modules,"*Software Quality J.*, vol. 11, no. 1, pp. 19–37, 2003.

[7] R. Kumar, S. Rai, and J. Trahan, "Neural-Network Techniques for Software-Quality Evaluation," *Reliability and Maintainability Symposium*,1998.

[8] T. Ostrand, E. Weyuker, and R. Bell, "Predicting the Location and Number of Faults in Large Software Systems," *IEEE Trans. Software Eng.*, vol. 31, no. 4, pp. 340–355, 2005.

[9] A.Hassan and R. Holt, "The Top Ten List: Dynamic Fault Prediction,"*Proc. ICSM'05*, Jan 2005.

[10] S. Kim, T. Zimmermann, E. W. Jr., and A. Zeller, "Predicting Faults from Cached History," *Proc. ICSE 2007*, pp. 489–498, 2007.

[11] T. Gyim´othy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.*, vol. 31, no. 10, pp. 897–910, 2005.

[12] M. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," IEEE Trans. Knowledge and Data Eng., vol. 15, no. 6, pp. 1437-1447, Nov./Dec. 2003.

[13] J. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann,1993.

[14] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A General Software Defect-Proneness Prediction Framework," IEEE Trans. Software Eng., vol. 37, no. 3, pp. 356-370, May/June 2011.

[15] K. Gao, T. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing Software Metrics for Defect Prediction: An Investigation on Feature Selection Techniques," Software: Practice and Experience,vol. 41, no. 5, pp. 579-606, 2011.

[16] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C.Henri-Gros, A. Kamsky, S. McPeak, and D.R. Engler, "A FewBillion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," Comm. ACM, vol. 53, no. 2, pp. 66-75, 2010.