PeerJ Preprints NOT PEER-REVIEWED

Impact of Restricted Forward Greedy Feature Selection Technique on Bug Prediction

K Muthukumaran, N L Bhanu Murthy
BITS Pilani Hyderabad Campus
Shameerpet, RR District, AP 500078
{p2011415, bhanu }@hyderabad.bits-pilani.ac.in

ABSTRACT
Several change metrics and source code metrics have been introduced and proved to be effective in bug prediction. Researchers performed comparative studies of bug prediction models built using the individual metrics as well as combination of these metrics. In this paper, we investigate the impact of feature selection in bug prediction models by analyzing the misclassification rates of these models with and without feature selection in place. We conduct our experiments on five open source projects by considering numerous change metrics and source code metrics. And this study aims to figure out the reliable subset of metrics that are common amongst all projects.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement- *Version Control*. D.2.8 **[Software Engineering]:** Metrics - *Performance Measures, Process Metrics, Product*

General Terms
Management, Measurement, Reliability

Keywords
Feature selection, Bug prediction, Software Quality

1. INTRODUCTION
Bug prediction models have become a popular method to enhance quality by identifying and fixing buggy files prior to release. Several source code metrics and change metrics have been introduced as predictors of bugs by researchers and they investigated the efficacy of these metrics by building prediction models. The bug prediction models can be built using classification algorithms like J4.5, Naïve Bayes Classifier etc. or regression techniques. There are several research papers in which classification algorithms have been explored by researchers to build bug prediction models by using features from source code metrics or change metrics or combination of these metrics [1] [3] [4].

However, bug prediction models have been built by considering all metrics under the study as features. There has been limited study in the direction of figuring out whether a subset of these metrics might improve the performance of prediction models as compared to model with all features.

In this paper, we investigate whether subset of the predictors improve performance of the prediction model or not by implementing prominent feature selection algorithms. If so, we ask whether the predictors in the subset are consistent across projects. There are comparative studies in literature that show that change metrics are better than source code metrics [5]. We consider combination of change metrics (50) and source code metrics(17) to check if the best feature subset has metrics only from change metrics or not.

This paper is organized as follows. We explain the related work and motivation in Section 2, various source code and change metrics that have been considered as features in this study are discussed in Section 3. We describe feature selection algorithm in Section 4 and discuss results, findings in Section 5.

2. RELATED WORK
Moser et al. considered 18 change metrics and built cost-sensitive classification models for three releases of the Eclipse. The results are very promising and clearly outperform predictors based on static code attributes for the Eclipse project [5].

Moser et al. in an another work analyzed the reliability of a subset of the above mentioned 18 change metrics for bug prediction and they showed that 3 out of 18 change metrics contain most information about software defects across three releases of Eclipse project [6]. They also show that prediction accuracy is not too much affected by using a subset of 3 metrics. It is worthwhile to note that their work is not towards identifying common predictors across projects but to find common predictors across different versions of the same project.

Krishnan et al. find that change metrics are consistently good and incrementally better predictors across the evolving products in Eclipse and according to them there is also some consistency regarding which change metrics are the best predictors [7]. At the same time Menzies et al argues that static code attributes or source code metrics also have significant role in prediction of faults and identify the best predictors among source code metrics [3]. Hence, we consider change and source code metrics to know their influence in faults through feature selection methods.

Most of the researchers in this field put efforts to predict whether a source file or binary is bug prone or not. But Shivaji et al. find that whether a change request is bug prone or not based on history of change requests [8]. They consider distinct leximes in the churned source code, which are quite huge in number, as features and extract them from churned source code by bag-of-words approach (BOW) [11]. They also consider other features from change metadata, source code complexity metrics. They applied feature selection algorithm on all these features to build bug prediction model for change requests and show that feature selection makes a huge improvement in prediction accuracy.

Though feature selection is extensively used in gene selection from microarray data and text categorization problems, it is not thoroughly explored in bug prediction research.

Some of the following works in other fields inspire us to explore the feature selection methods closely. In bio informatics, Huiqing Liu et al show that feature selection improves the classification accuracy significantly in their comparative study on feature selection and classification methods [10].

3. METRICS

In our study, we have conducted experiments on metrics set consisting of prominent source code metrics and change metrics. And the following sections describe all these metrics.

3.1 Source Code Metrics

CK Metrics [13] and object oriented metrics have been considered under source code metrics category and the following table describes details about these metrics.

Table 1: Source Code Metrics

CK Metrics	
WMC	Weighted Method Count
DIT	Depth of Inheritance Tree
RFC	Response For Class
NOC	Number Of Children
CBO	Coupling Between Objects
LCOM	Lack of Cohesion in Methods
OO Metrics	
Fan-In	Number of other classes that reference the class
Fan-Out	Number of other classes referenced by the class
NOA	Number of attributes
NOPA	Number of public attributes
NOPRA	Number of private attributes
NOAI	Number of attributes inherited
LOC	Number of lines of code
NOM	Number of methods
NOPM	Number of public methods
NOPRM	Number of private methods
NOMI	Number of methods inherited

3.2 Change Metrics

We have used some of the change metrics that are used by Moser et al. for our experiments [5] [15] [16] [17]. All these metrics are described in Table 2.

3.3 Entropy of Changes

Hassan introduces metrics that capture the complexity of code changes and shows that these metrics are better predictors than other well-known predictors like prior modifications and prior faults [12]. He proposes four variants of this metric and Ambrose et al. define three more variant of the metric [2]. Ambrose et al. perform experiments with these metrics on the same projects as what we have considered and show that the variant, Weighted History of Complexity Metric (WHCM), is better predictor than others. We have used this metric for our experimentation

Table 3: Entropy Metrics

Metric Name	Definition
WHCM	Weighted History of Complexity Metric

We are not providing the complete description of this metric here as it might distract reader's focus from the current topic and refer reader to the original papers by Hasan et al. and Ambrose et al. for better understanding of these metrics [12] [2] . And this comment holds good for metrics that will be discussed in next sections i.e., 3.3 and 3.4.

Table 2: Change Metrics

Metric Name	Definition
REVISIONS	Number of revisions of a file
REFACTORINGS	Number of times a file has been refactored
BUGFIXES	Number of times a file was involved in bug-fixing2
AUTHORS	Number of distinct authors that checked a file into the repository
LOC_ADDED	Sum over all revisions of the lines of code added to a file
MAX_ LOC_ADDED	Maximum number of lines of code added for all revisions
AVE_ LOC_ADDED	Average lines of code added per revision
LOC_DELETED	Sum over all revisions of the lines of code deleted from a file
MAX_ LOC_DELETED	Maximum number of lines of code deleted for all revisions
AVE_ LOC_DELETED	Average lines of code deleted per revision
CODECHURN	Sum of (added lines of code – deleted lines of code) over all revisions
MAX_ CODECHURN	Maximum CODECHURN for all revisions
AVE_ CODECHURN	Average CODECHURN per revision
AGE	Age of a file in weeks (counting backwards from a specific release)
WEIGHTED_AGE	$\frac{\sum_{i=1}^{N} Age(i) * LOC\_ADDED(i)}{\sum_{i=1}^{N} LOC\_ADDED(i)}$ *Age(i)* is the number of weeks starting from the release date for revision *i* and *LOC_ADDED(i)* is the number of lines of code added at revision *i*.

3.4 Churn of Source Code Metrics

Using churn of source code metrics to predict post release defects is novel. The intuition is that higher-level metrics may better model code churn than simple metrics like addition and deletion of lines of code. The churn of source code metrics is defined for all CK and OO metrics in Table 2.

Ambrose et al. define five variants of this metrics and performed experiments with these metrics on the same projects as what we have considered and showed that WCHU is better predictor than others. We have used this metric for our experimentation. It is to be noted that there will be 17 metrics under this category.

Table 4: Churn Metrics

Metric Name	Definition
WCHU_m for each metric, m, in Table 1	Weighted Churn of Source Code Metric of metrics m (say CBO).

3.5 Entropy of Source Code Metrics

Ambrose et al. extended the concept of code change entropy [10] to the source code metrics listed in Table 1 with the aim of measuring the complexity of the variants of a metric over subsequent sample versions.

They define five variants of this metrics and performed experiments with these metrics on the same projects as what we have considered and showed that LDHH is better predictor than others. We have used this metric for our experimentation. It is to be noted that there will be 17 metrics under this category.

Table 5: Source Code Entropy Metrics

Metric Name	Definition
LDHH_m for each metric, m, in Table 1	Linearly Decayed Entropy of Source Code Metrics, m.

4. FEATURE SELECTION

We have considered 67 metrics, as discussed in previous section, to know whether prediction accuracy of bug prediction models can be improvised by reducing or eliminating some of these metrics.

There are several techniques to find the subset of features that can optimize performance of prediction model. They include filter method for example correlation based selection or wrapper method i.e., forward selection or backward elimination using any kind of classification algorithm. We adopt the second approach and implement Restricted Forward Selection Greedy Algorithm [9] and the algorithm is discussed below.

Restricted Forward Selection Algorithm

Split the data randomly into three partitions and considers two partitions as outer train set and remaining partition as outer test set. And 70% of outer train set is taken as inner train set and remaining 30% as inner test set or cross validation set.

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_i, \ldots, x_m\}$ be the set of all metrics that are under consideration for bug prediction problem and $\mathcal{F} = \{\emptyset\}$ be the set containing features selected after every step.

1. Initially consider each metric one at a time and build prediction models with the metric over inner train set and calculate its prediction accuracy on inner test set. Thus we have m values, prediction accuracies of m models, corresponding to each attribute. Select attribute and model with the best prediction accuracy out of these m attributes and models, subtract it from $\mathcal{X}$ and add it to $\mathcal{F}$ by storing the corresponding prediction accuracy.

2. Couple the set $\mathcal{F}$ with each attribute in $\mathcal{X}$ and calculate prediction accuracies of respective models and do this by selecting one attribute at a time for all attributes in $\mathcal{X}$.

3. Find the attribute and model that gives out the best prediction accuracy and add it to $\mathcal{F}$ by subtracting it from $\mathcal{X}$.

4. Repeat Steps 2 and 3 till $\mathcal{X} = \{\emptyset\}$ and $|\mathcal{F}| = m$

5. Select the model with best prediction accuracy out of all models and the corresponding subset.

6. Find out the prediction accuracy of the model using optimal subset over the outer test set.

Now we will discuss the measure that is to be used as prediction accuracy in the above algorithm. The prominent information retrieval measures – Recall and Precision - are described in Table 6 and any one of them can be used as a measure for ***prediction accuracy*** of the model. We have used recall as prediction accuracy as it is considered to be more valuable than precision for bug prediction problem by researchers in this area. We describe some of these reasons below.

There are two types of misclassifications for any binary classification learning problem like bug prediction.

Positive Misclassification - Actual buggy file predicted as non-buggy file

Negative Misclassification - Non buggy file predicted as buggy file

The cost associated to fix a post-release bug that is not caught during bug prediction is much more than the cost of performing quality assurance activities like code review or unit testing on a file which is predicted to be buggy but not really buggy. Hence positive misclassification is considered to be much more severe than negative misclassification and lesser the number of positive misclassifications larger the recall value. Finally we have to find the best subset of features that can minimize misclassification costs or equivalently maximize recall.

Table 6: Confusion Matrix

		Observed Output	
Confusion Matrix		**Positive**	**Negative**
Predicted Output	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

$$Recall = \frac{TP}{TP + FN}$$

$$Precission = \frac{TP}{TP + FP}$$

5. EXPERIMENT AND RESULTS

We have conducted all our experiments on five open source projects namely Eclipse JDT, Eclipse PDE, Lucene, Mylyn, Equinox and data pertaining to metrics and post release bugs of these projects is publicly made available by Ambrose et al. [2]. The details about number of classes, number of defects etc. of five projects are shown in Table 7.

PeerJ Preprints NOT PEER-REVIEWED

Table 7: Projects

Projects	Pred Rel	#classes	#post release defects
Eclipse JDT Core	3.4	997	463
Eclipse PDE UI	3.4.1	1562	401
Apache Lucene	2.4.0	691	103
Mylyn	3.1	2196	677
Equinox framework	3.4	439	279

The source code metrics and change metrics that are considered for this study are discussed in Section 3 and the data of these metrics are obtained from the above mentioned repositories for our experiments.

We will find optimal feature subset by implementing the Restricted Forward Selection Greedy Algorithm that is described in section 3.

The prediction models with optimal features as well as all features are built with training data for each project and their performance is evaluated by implementing these models on testing data. The Recall and Precision of these models are shown in Table 8. With feature selection in place, there is surge in recall value for all projects with an average increase of 12.26% and maximum increase of 21.21%.

The higher recall is being achieved with a little hit on precision but this is accepted for bug prediction problem as Menzies et al. say that, for software engineering data sets with large neg/pos ratios, it is often required to lower precision to achieve higher recall [14].

Table 8 : Recall and Precision

Projects	Before Feature selection		After Feature selection	
	R	P	R	P
Equinox	**0.7575**	0.5952	**0.9696**	0.4444
Lucene	**0.0833**	0.6666	**0.2083**	0.5000
Mylyn	**0.0348**	0.4285	**0.0930**	0.5333
PDE	**0.1538**	0.7058	**0.2051**	0.4102
JDT	**0.3055**	0.6111	**0.4722**	0.5964

The metrics that are selected in optimal feature subset for each project is shown in the Table 9. There is no metric that is present in optimal feature subsets of all projects. And hence, this work makes a step forward to the generic hypothesis that there may not be common predictors across projects.

There have been research studies that establish change metrics are better predictors than source code metrics [5] [16]. But it is interesting to note that the optimal feature subset of Equinox project contains only source code metrics and for the remaining four projects the optimal feature set contain change metrics as well as source code metrics. And this observation indicates that source code metrics should not be considered as inferior predictors as compared to change code metrics.

Table 9: Metrics in optimal feature subset

Projects	Features
Equinox	1. numberOfAttributes
Lucene	1. linesRemovedUntil 2. numberOfAttributes
Mylyn	1. avgLinesRemovedUntil 2. codeChurnUntil 3. WCHU_numberOfLinesOfCode 4. LDHH_numberOfPublicMethods 5. LDHH_numberOfPublicAttributes 6. LDHH_numberOfPrivateAttributes 7. CvsWEntropy 8. avgCodeChurnUntil 9. rfc 10. numberOfMethodsInherited
PDE	1. WCHU_rfc 2. LDHH_numberOfLinesOfCode 3. LDHH _fanIn, 4. numberOfFixesUntil 5. LDHH_wmc 6. weightedAgeWithRespectTo 7. maxLinesRemovedUntil 8. avgLinesAddedUntil
JDT	1. WCHU_noc 2. LDHH_numberOfAttributesInherited 3. LDHH_numberOfPrivateAttributes 4. avgCodeChurnUntil

6. CONCLUSION

This paper has emphasized the significance of feature selection in bug prediction by showing considerable improvement in prediction accuracies. The experiments conducted on five open source projects reveal that there is an average increase of 12.26% with maximum increase of 21.21% in prediction accuracies after applying Restricted Forward Selection Greedy Algorithm for feature selection. We also confirm that these is no common predictor across five projects and this point takes a step closer to general hypothesis that there may not be any common predictors across projects. The optimal feature subset for one project contain only source code metrics and for all other projects it is mix of source code and change metrics which indicate that source code metrics are equally capable as change metrics in predicting bugs.

7. REFERENCES

[1] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pp. 9–9, May 2007.

[2] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," *Mining software Repositories (MSR)*, 2010.

[3] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE …*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[4] E. Giger, M. D. Ambros, M. Pinzger, and H. C. Gall, "Method-Level Bug Prediction," *Proceedings of the ACMIEEE international symposium on Empirical software engineering and measurement (2012)*, pp. 171–180, 2012.

[5] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," *Proceedings of the 13th international conference on Software engineering ICSE 08*, no. April, pp. 181–190, 2008.

[6] R. Moser, W. Pedrycz, and G. Succi, "Analysis of the Reliability of a Subset of Change Metrics for Defect Prediction," *Proceedings of the Second ACMIEEE international symposium on Empirical software engineering and measurement ESEM 08*, pp. 309–311, 2004.

[7] S. Krishnan, C. Strasburg, R. R. Lutz, and K. Goševa-Popstojanova, "Are change metrics good predictors for an evolving software product line?," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 7:1–7:10.

[8] S. Shivaji, E. J. Whitehead, and R. Akella, "Reducing Features to Improve Code Change-Based Bug Prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, Apr. 2013.

[9] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," in *Pattern Recognition*, 1994, vol. 129, no. 8, pp. 121–129.

[10] H. Liu, J. Li, and L. Wong, "A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns." *Genome informatics. International Conference on Genome Informatics*, vol. 13, no. 0919–9454 LA - eng PT - Journal Article RN - 0 (Proteome) SB - IM, pp. 51–60, Jan. 2002.

[11] S. Scott and S. Matwin, "Feature Engineering for Text Classification," *Representations*, vol. 6, no. April, pp. 379–388, 1999.

[12] A. E. Hassan, "Predicting faults using the complexity of code changes," *2009 IEEE 31st International Conference on Software Engineering*, no. 2009, pp. 78–88, 2009.

[13] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[14] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to 'Comments on "Data Mining Static Code Attributes to Learn Defect Predictors,"'" *IEEE Transactions on Software Engineering*, vol. 33, pp. 637–640, 2007.

[15] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[16] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," *Software Engineering, 2005. ICSE 2005. ...*, pp. 284–292, 2005.

[17] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp.340–355,2005.