ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/372902947

Determining the Most Significant Metadata Features to Indicate Defective Software Commits

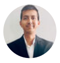
Conference Paper · May 2023

DOI: 10.1109/SERA57763.2023.10197721

CITATIONS
0

READS
27

3 authors, including:


Rupam Kumar Dey
University of Tennessee at Knoxville
5 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Anahita Khojandi
University of Tennessee at Knoxville
148 PUBLICATIONS **1,501** CITATIONS
SEE PROFILE

All content following this page was uploaded by Rupam Kumar Dey on 10 May 2024.

The user has requested enhancement of the downloaded file.

Determining the Most Significant Metadata Features to Indicate Defective Software Commits

Rupam Kumar Dey
Industrial & Systems Engineering
The University of Tennessee
Knoxville, USA
rdey1@vols.utk.edu

Anahita Khojandi
Industrial & Systems Engineering
The University of Tennessee
Knoxville, USA
khojandi@utk.edu

Kalyan Perumalla
Industrial & Systems Engineering
The University of Tennessee
Knoxville, USA
kperuma3@utk.edu

***Abstract*—Defects are largely inevitable in the software development life cycle. Since we cannot avoid them during the development process, we can only desire to fight back with our limited resources in terms of time and monetary investment. Like in many other fields, machine learning models can be of help to mitigate the problem of defects by predicting both bug frequency and defective modules at different granularity levels. However, machine learning models are as good as the quality of the pre-selected set of features under consideration. Therefore, importance must be given while selecting only the necessary features from the original set of features. In this study, we compared various machine learning models with varying feature selection techniques and found the superiority of random forest-based machine learning techniques with wrapper methods. Random forest-based models with the wrapper method were able to detect all the buggy classes successfully on the validation data set.**

***Index Terms*—Bugs, Software Development Life Cycle, Feature Selection, Random Forest Model, Wrapper Method**

I. Introduction

A. Overview

Bugs are an unavoidable part of the software development process. However, dealing with them sooner than later stages of the development cycle is preferred due to the huge cost involved with bug fixing [1]. Also, it is important to prioritize which section to be dealt with due to limited resources. Therefore, the prediction of bugs helps directly identify and prioritize the buggy modules according to relevant granularity. However, the accuracy of various machine learning models depends on the number of feature selections. Too many or too few features can hurt the models equally. Hence, the number of features should be optimum. To deal with this issue, we examined the effect of selecting different feature subsets along with different machine learning models considering broadly three types of feature selection methods (filter, wrapper, and embedded) on a change metrics data set from the Eclipse JDT Core software system.

Feature selection methods play a vital role in selecting important features and those features could be used later on to combat bugs with limited resources. There are three types of well-known feature selection methods, namely, filter, wrapper, and embedded methods.

In the filter method [2], features are selected based on the statistical score on correlation with the target variable. This method is independent of any particular machine learning algorithm. A correlation matrix with a heat map is one such method that is used in our study here.

Starting with a subset of features and judging the outcome of training with the subset, the addition or elimination of features is decided in the wrapper method [3]. The optimal set of features could be obtained by performing the wrapper method on a specific machine-learning algorithm. Forward feature selection and backward feature elimination are two of the well-known wrapper methods. In forward feature selection, the starting feature set is empty and one new feature (the most significant feature) is added at each iteration from the remaining features. In backward feature elimination, the starting feature set consists of all the existing features and at each iteration one of the existing feature (the least significant feature) is eliminated. For both forward feature selection and backward feature elimination, iterations could be stopped when the addition or elimination of features does not improve model performance at a significant level.

Embedded methods [4] determine features based on the contribution during the training phase at each iteration. Random forest top ten feature selection is one such method.

B. Problem Statement

Every software needs to go through the whole development life cycle before it can be delivered to the end customer. Rigorous testing is needed to ensure defect-free software for the consumer market. Due to the nature of the process and the huge workforce involved at each stage of the software development cycle, defects are almost always an undeniable part of it. At the same time, it is not possible and practical to check each and every module of a huge software. That's where the machine learning model comes into play. If we can use machine learning techniques to detect buggy modules and prioritize our limited resources accordingly, it will save both time and monetary investment at the same time. However, machine learning models are as good as the selected feature subsets. Therefore, careful selection of features is paramount for the success of the model. Some of the well-known feature selection methods like filter, wrapper, and embedded could be used to select important feature sets effectively and efficiently.

C. Organization

The rest of the paper is organized in the following manner. Previous related works are described in Section II. The methodology is listed in Section III. Section IV portrays the analysis and key takeaways. While Section V summarizes all the important findings, our future work is listed in Section VI.

II. Background

In the past, software defect prediction was done considering numerous approaches. Some of the well-known approaches are based on the decision tree induction method, relational association-based rule, neural networks, deep forest, bandit algorithms, etc. Feature reduction techniques like the filter, wrapper, and embedded were also explored in the process.

A. Rule and Association-based Models

In one of the studies [5], the researchers used a rule-based decision tree induction method to select 15 features out of 94 and considered 18 classifiers on class-level data set KC1 from the promise repository to determine if a particular class had any defects or not. They found their feature selection method was superior compared to the other two methods (RELIEF features and Support Vector Machine features) in terms of error metric and area under the receiver operating characteristic curve (AUROC). They only explored filter methods for feature selection and it would be interesting to see how other multivariate feature selection methods would perform with different classifiers.

Another study [6] used a relational association-based rule to detect software defects. They used relations or correlations among various attributes of a large data set to determine if a module is defective or not. They achieved superior results in favor of their algorithm while applying to 10 NASA data sets (CM1, KC1, KC3, PC1, JM1, MC2, MW1, PC2, PC3, and PC4) compared to other classifiers (CBA2, 1R classifier, Bagging classifier, evolutionary decision rules for subgroup discovery classifier) in terms of accuracy.

B. Optimization-based Models

Taking fewer features into consideration while predicting software defects sometimes affect adversely in terms of model performance. On the other hand, taking many features into consideration hurt us in terms of computation resources. Considering these two inversely related phenomena, this study [7] proposed a multi-objective feature selection (MOFES) method that will optimize not only the number of features but also model performance. Considering four classifiers (J48- a decision tree classifier algorithm, K-nearest neighbor, Logistic Regression, and naive Bayes) with five Pareto-based multi-objective optimization algorithms (PAES- Pareto archived evolution strategy, NSGA II- non-dominated sorting genetic algorithm II, SPEA2- strength Pareto evolutionary algorithm 2, EMOA- evolutionary multi-objective optimization algorithm, MOcell- A cellular genetic algorithm for multi-objective optimization) on two data sets (RELINK and PROMISE), they found NSGA II as the best Pareto based multi-objective optimization algorithm and their method MOFES achieved better performance on many occasions when compared with 22 states of the art feature selection methods.

C. Neural Network-based Models

The intervention of neural networks in other areas is remarkable. Deep neural networks are also a popular choice nowadays for various classification and regression tasks related to software defect prediction. Researchers from this article [8] used a deep neural network-based method to predict software defects and found their method's superiority when compared with three state-of-the-art methods (support vector regression, fuzzy support vector regression, and decision tree regression model) on two data sets (medical imaging system data set and NASA PROMISE (KC2)) in terms of lower mean squared error (MSE) and higher coefficient of determination (R^2).

Another study [9] combined layer-by-layer training methods from deep learning and random forest classifier from ensemble learning method to create a better model for defect prediction than any one of them alone. Their approach exhibited superior results (a 5% increase) in terms of AUC value when compared with six other classifiers (deep forest model (gcForest), deep belief network, random forest, naive Bayes, logistic regression, and support vector machine) using four renowned data sets (NASA, PROMISE, AEEEM, and Relink) from 25 open source projects.

Not only within project but also cross-project defect predictions are possible using neural networks. This study [10] predicted software defects both within project and cross-project boundary on ten open source projects using semantic features. The semantic features were produced using deep belief network. Their results showed promise that was based on precision, recall and F-1 score.

D. Feature Reduction-based Models

Different feature reduction technique yields different level of accuracy for different models. To select the best feature reduction technique for a defect prediction model, one of the studies [11] used the bandit algorithm. Applying the bandit algorithm on 14 data sets (PROMISE and from D'Ambros [12]) using a combination of three types of prediction models (control(recommended by previous studies), Ad hoc (selected arbitrarily), and experiment (showed the highest accuracy among candidates)) and four feature reduction techniques (correlation-based feature selection, consistency-based feature selection, Akaike information criterion step-wise feature selection, and Bayesian information criterion step-wise feature selection) showed that their approach performed similar or better than the state of the art techniques.

Considering different approaches into consideration while selecting important features from available feature pool also proved useful. One of the studies [13] showed this case where the author considered the average of chi-squared, information gain, and Pearson correlation coefficient to re-rank the available features. Random forest classifiers using those re-ranked features brought superior results in terms of AUC value on six projects (CM1, KC3, MC1, MC2, MW1, PC1) from NASA data sets.

Sometimes combining two of the methods yield better output. For example, one of the articles [14] found ensemble learning with feature selection very useful in classifying software defects. The authors examined 11 NASA data sets considering 11 metrics with four ensemble algorithms (sequential minimal optimization, multi-layer perceptron, k-nearest neighbor, and J48). They did not consider all of the multivariate feature selection methods.

E. Miscellaneous

The importance of regularization is shown in this study [15]. This study performed linear and Poisson regression while predicting the number of defects. While doing so, they used regularization methods like ridge, lasso, and ElasticNet and found up to 50% less mean squared error along with less variance in the results from five open-source Java systems (Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, and Mylyn).

Feature-oriented defect prediction also showed promise. One of the studies [16] created a novel defect data set at the granularity level of features and found precision and recall values as high as 0.85 while predicting faulty features considering two feature-based metric sets (consisting of eight and six features respectively). The study used seven commonly used classifiers (J48 decision trees, k-nearest-neighbor, logistic regression, naive Bayes, artificial neural network, random forest, and support vector machine) and four evaluation metrics (precision, recall, F-1 score, and AUROC) while doing so. Random forest regression and neural network-based approach showed superiority.

Although feature selection methods were applied for software defect prediction previously, prior studies did not take into account the effect of considering the full span of multivariate feature selection methods for change metric data sets with respect to baseline. Also, the selected features were often described as top-k features rather than considering minimum and adequate number of features. In this study, we filled this gap through our research.

F. Our Contribution

Our unique contributions in this study are as follows:

- We showed the impact of choosing the filter, wrapper, and embedded feature selection methods on prediction models on a change metrics data set from the Eclipse JDT Core software system. Accordingly, we identified a minimum and adequate set of features to facilitate defect prediction with a low RMSE value;
- We showed the efficacy of the feature selection approaches in predicting the class-wise number of defects (experiment one). At the same time, the model was also able to detect if a module was defective or not (experiment two).

III. Methodology

In this section, we describe the data set and its features, and discuss data pre-processing, model selection, threshold criteria, experiments, and reproducibility considerations.

A. Data Set

Bug Prediction Data set contains data on five different systems (Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, and Mylyn). We considered data from Eclipse JDT Core, specifically the change metrics data set that could be downloaded from here. The data set consists of 15 features that are essentially 15 change metrics. Details about the data set columns are provided in table I. It has 997 class-wise entries of which 206 are faulty classes (or buggy classes).

TABLE I
FEATURE DETAILS

Feature Name	Description
numberOfVersionsUntil	number of versions
numberOfFixesUntil	number of bug fixes
numberOfRefactoringsUntil	number of times a file refactored
numberOfAuthorsUntil	number of distinct authors
linesAddedUntil	summation of lines of code added to a class across all versions
maxLinesAddedUntil	maximum number of lines added to a class
avgLinesAddedUntil	average number of lines added to a class
linesRemovedUntil	number of lines removed from a class
maxLinesRemovedUntil	maximum number of lines removed from a class
avgLinesRemovedUntil	average number of lines removed from a class
codeChurnUntil	difference between added lines of code and deleted lines of code
maxCodeChurnUntil	maximum difference between added lines of code and deleted lines of code
avgCodeChurnUntil	average difference between added lines of code and deleted lines of code
ageWithRespectTo	age in weeks (backward) from a particular release
weightedAgeWithRespectTo	age in weeks (backward) from a particular release with respect to lines of codes added
bugs	number of bugs

The type of all the columns in the data set is numeric.

Table II gives us a rough idea about the range of each column's value. It is to be noted that the '-' sign at the start of a numeric value denoted a negative value whereas the same '-' sign appearing in between two numbers denoted the range.

B. Data Processing

The initial .csv file contains 21 columns. We got rid of five columns, namely, 'classname', 'nonTrivialBugs', 'majorBugs', 'criticalBugs', and 'highPriorityBugs' as those columns were

TABLE II
RANGE OF COLUMNS

Feature Name	Value Range
numberOfVersionsUntil	1-709
numberOfFixesUntil	0-166
numberOfRefactoringsUntil	0-2
numberOfAuthorsUntil	1-15
linesAddedUntil	0-65571
maxLinesAddedUntil	0-7452
avgLinesAddedUntil	0-222.22
linesRemovedUntil	0-59724
maxLinesRemovedUntil	0-7452
avgLinesRemovedUntil	0-206.19
codeChurnUntil	-1745 - (+ 10624)
maxCodeChurnUntil	0-2768
avgCodeChurnUntil	-39.86 - (+121.4)
ageWithRespectTo	7.86-367
weightedAgeWithRespectTo	0-227.63
bugs	0-9

irrelevant to our study. And the 'bugs' column was the target. The overall data set was divided into three subdivisions, namely, training, validation, and test set. The split ratio was 60:20:20 for the training-validation-test set. The shape of the training, validation and test set are (598, 15), (199, 15), and (200, 15) respectively. Out of 199 entries in the validation set, 41 of them are buggy classes. We removed the 'class name' column from the data set and the target feature is 'bugs'. Therefore, we were considering 15 features as predictor variables. We built eight models for prediction purposes and followed three types of feature selection methods (filter, wrapper, and embedded) including baseline models with all features. The models include A. linear regression with all features (LRAF), B. linear regression with features based on Pearson correlation coefficient (LRCC), C. linear regression with the forward feature selection (LRFFS), D. linear regression with backward feature elimination (LRBFE), E. Random Forest regression with all features (RFAF), F. Random Forest regression with top few features (RFTF), G. Random Forest regression with forward feature selection (RFFFS), and H. Random Forest regression with backward feature elimination (RFBFE).

The threshold value considered for the Pearson correlation coefficient was 0.49 in model B. The initially selected features were further examined for the multicollinearity case. Considering all these, the final feature set was obtained.

On top of that, we conducted two experiments. Experiment one can be stated this way 'for a particular class, if the actual number of bugs is one or more and the predicted number of bugs is more than zero, then those entries will not have any contribution in the calculation of the root mean squared error (RMSE) value. If the actual number of bugs is zero and the model predicted some number other than zero, then those entries were considered while calculating the RMSE value'. We tagged the obtained RMSE value from experiment one as the revised RMSE. On the other hand, experiment two is about finding the optimal feature subsets to detect the total number of buggy classes on the validation data set based on the result from the corresponding bug prediction models.

C. Metric Selection

To facilitate comparison among various model results and to maintain consistency, the metrics used here are mean squared error (MSE) and root mean squared error (RMSE) for the optimum number of features selection and evaluation of different model performances. Since the sci-kit learn package provided the negative of the MSE value, we had to get rid of the negative sign to obtain the actual MSE value. We also captured the % negative mean squared error reduction value with the addition/ elimination of one feature at each iteration. The related equations and mathematical notations are listed below for convenience.

If we get n predictions based on the n actual values, and if $\hat{y}_i$ is the predicted value for the actual value y_i for each i, the equation for MSE and RMSE are as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = [\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2]^{\frac{1}{2}}$$

We settled for a combination of the lower RMSE values and the number of defective module predictions to select better models.

D. Reproducibility

All the analyses done are provided in the form of jupyter notebook-based script. After downloading the change metrics data set from the Eclipse JDT core software system of the bug prediction dataset, the script can be used to reproduce the results found in this study. The script can be provided on request. Libraries used in this script like sci-kit learn, mlxtend, pandas, NumPy, etc. could be installed following commands provided in the script.

IV. Analysis

In this section, we showed the performance of eight different models (some with various feature selection methods) in terms of the usual RMSE value, revised RMSE value, and the number of buggy class detection. We also captured the minimum and sufficient set of features for each model for the purpose of bug predictions.

A. LRAF

This model considered all the available features while predicting the number of bugs and detecting buggy modules. This model returned an RMSE value of 0.72 for the validation set and 0.78 for the test set with all features. Performing experiment one, the RMSE value for the validation set went down to 0.5. The model successfully captured 40 buggy classes out of 41 in the validation set.

B. LRCC

We used the Pearson correlation coefficient to select important features for the number of bugs prediction. Initially, we got four such features (number of versions until, lines added until, lines removed until, code churn Until). Some of the features are highly inter-correlated with each other (known as ‘multicollinearity’). Considering multicollinearity, finally, we selected two features, namely, the number of versions until and code churn until for prediction purposes. The RMSE value applying this model for validation and test set were 0.71 and 0.82. Upon performing experiment one, we got the revised RMSE value of 0.47 and it captured 37 out of 41 buggy classes within the validation set. Fig. 1 shows the Pearson correlation coefficient values among different predictor variables.

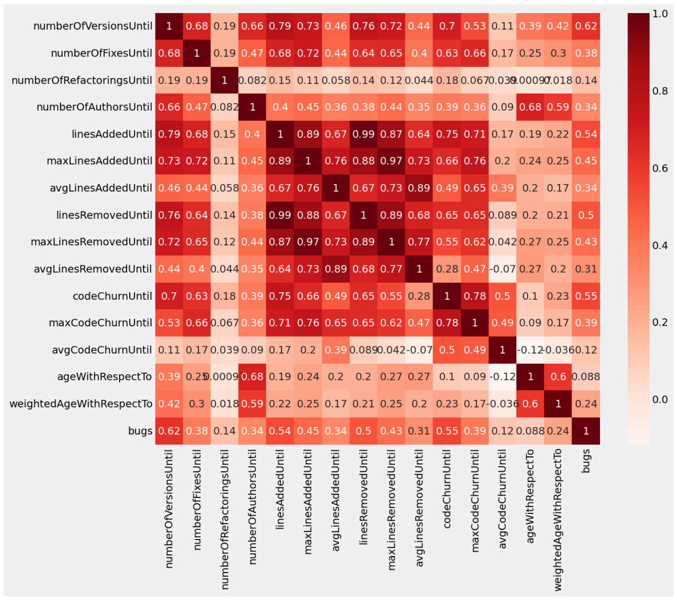


Fig. 1. Heat-map with Pearson correlation coefficient with all features.

C. LRFFS

In this model at each iteration the feature that gives the best performance score for the model is added. To select an optimum number of features for experiment two, we added one feature at each iteration and captured % negative mean squared error reduction with the addition of each new feature. From Fig. 2 it is evident that with only four features (minimum feature set) we covered most of the higher positive % error reduction values (leaving 0.13 since it is negligible compared to other positive values). We chose an adequate number of features around seven, and after that % error reduction got higher negative values. The four-feature set consists of features like the number of versions until, code churn until, average code churn until, and age with respect to. On the other hand, the seven-feature set consists of the number of versions until, number of fixes until, average lines removed until, code churn until, average code churn until, age with respect to, and weighted age with respect to. The four feature set returns RMSE value of 0.73 and 0.81 for validation and test set

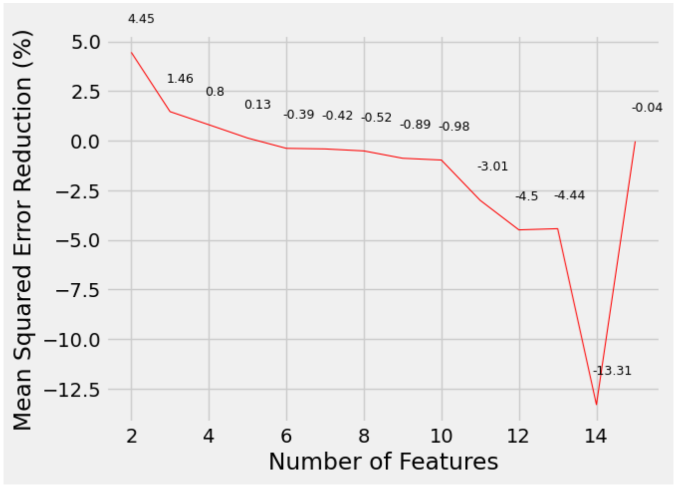


Fig. 2. Mean squared error reduction in percentage with respect to the addition of new features.

respectively. We got the revised RMSE value of 0.43 and all the buggy classes were captured with experiment one on the validation set. With the seven-feature set, RMSE values were 0.73 and 0.78 for the validation and test sets respectively. The revised RMSE value was 0.48 and 40 out of 41 buggy classes were captured on the validation set.

D. LRBFE

In this model at each iteration the feature that is least significant is eliminated. Starting with all the features, we eliminated one feature at each iteration for experiment two. From Fig. 3 it can be inferred that with only four features (minimum feature set) we did not lose much in terms of % error reduction. The adequate number of features is around

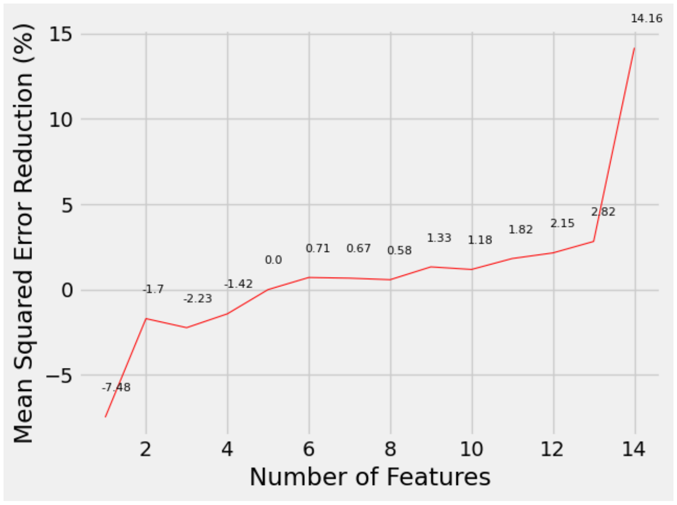


Fig. 3. Mean squared error reduction in percentage with respect to the elimination of existing feature.

seven after which % error reduction is not significant. The four-feature set consists of the number of authors until, code churn until, average code churn, and age with respect to. On the other hand, the seven-feature set consists of the number of authors until, lines added until, lines removed until, code churn until, average code churn until, age with respect to, and weighted age with respect to. The four feature set returns RMSE value of 0.77 and 0.93 for validation and test set respectively. We got the revised RMSE value of 0.53 and 38 out of 41 buggy classes were captured with experiment one on the validation set. With the seven-feature set, RMSE values were 0.81 and 0.87 for the validation and test sets respectively. The revised RMSE value was 0.51 and 39 out of 41 buggy classes were captured on the validation set.

E. RFAF

RFAF considers all 15 features with 1000 trees while predicting the number of defects. The RMSE values incurred for this model were 0.71 and 0.85 on validation and test set respectively. Performing experiment one, this model returned a revised RMSE value of 0.46 and captured all buggy classes on the validation data set.

F. RFTF

RFTF was built with the top ten features from all feature sets (15 features) based on feature importance value. Table III shows the feature importance value in descending order for the top ten features.

TABLE III
TOP TEN FEATURES WITH DESCENDING FEATURE IMPORTANCE VALUE

Feature Name	Importance Value
numberOfVersionsUntil	0.35
codeChurnUntil	0.16
linesAddedUntil	0.08
ageWithRespectTo	0.06
avgLinesRemovedUntil	0.05
weightedAgeWithRespectTo	0.05
maxLinesAddedUntil	0.04
avgLinesAddedUntil	0.04
linesRemovedUntil	0.04
numberOfFixesUntil	0.03

From the top-ten feature model, we got RMSE values of 0.72 and 0.84 for validation and test set respectively. Upon performing experiment one, the model returned 0.48 as the revised RMSE value and captured 40 out of 41 buggy classes on the validation set.

G. RFFFS

In this model at each iteration the feature that gives the best performance score for the model is added. We selected all 15 features as the number of features to be added one by one at each iteration for experiment two. From Fig. 4 it is evident that with only four features (minimum feature set) we covered most of the higher values of % error reduction. The adequate number of features is around seven after which the % error reduction value went negative by a significant amount. The four-feature set consists of the following features: number of versions until, max lines added until, average code churn until, and age with respect to. On the other hand, the features included in the seven-feature set are the number of versions until, number of fixes until, number of refactorings until, max lines added until, max lines removed until, average code churn until, and age with respect to. The four feature set returns RMSE value of 0.82 and 0.83 for validation and test set respectively. We got the revised RMSE value of 0.46 and all the buggy classes were captured with experiment one on the validation set. With the seven-feature set, RMSE values were 0.81 and 0.88 for the validation and test sets respectively. The revised RMSE value was 0.47 and the number of buggy classes captured was the same as the four-feature set with experiment one on the validation set.

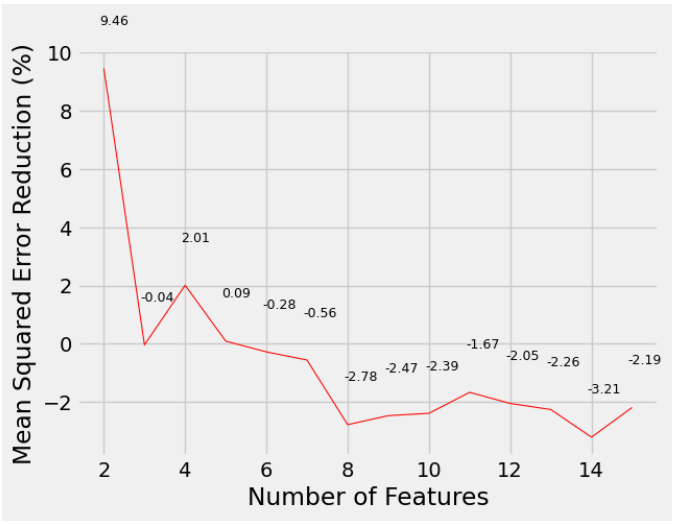


Fig. 4. Mean squared error reduction in percentage with respect to the addition of new features.

H. RFBFE

In this model at each iteration the feature that is least significant is eliminated. We selected one feature as the number of features to be kept after eliminating one feature at each iteration for experiment two. From Fig. 5 it is clear that with only four features (minimum feature set) we did not lose much in terms of % error reduction. The adequate number of features is around seven after which % error reduction is not significant. The four-feature set consists of the following features: number of versions until, max lines added until, average code churn until, and age with respect to. On the other hand, the features in the seven-feature set include the number of versions until, number of fixes until, number of refactorings until, max lines added until, max lines removed until, average code churn until, and age with respect to. The four feature set returns RMSE values of 0.82 and 0.83 for the validation and test sets respectively. We got the revised RMSE value of 0.46 and all the buggy classes were captured with experiment one on the validation set. With the seven-feature set, RMSE values were 0.81 and 0.88 for the validation and test sets respectively. The revised RMSE value was 0.47 and all the buggy classes were captured on the validation set. The LRFFS model with seven feature set along with the LRAF model provided us with the

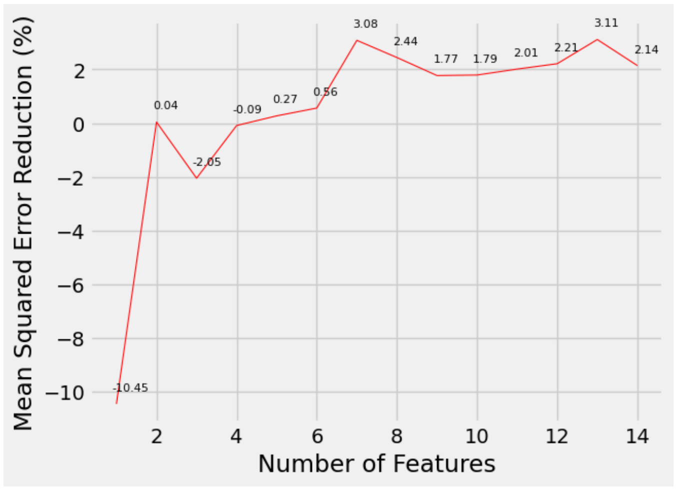


Fig. 5. Mean squared error reduction in percentage with respect to the elimination of existing feature.

lowest RMSE value (0.78) on the test data set. Although both RFAF and LRCC models provided us with the lowest RMSE value (0.71) for the test data set, LRFFS with the feature set was not far too in terms of RMSE value (0.73). On the other hand, LRFFS with four feature sets yielded the lowest RMSE value (0.43). Considering the nature of experiment one, it can be inferred that the model predicted some number when the actual class was buggy (one or more bugs) and a small number compared to other models when the class was not buggy (zero bugs).

V. Summary of Findings

We applied three types of feature selection methods (filter, wrapper, and embedded) to select the best set of features for bug prediction models. In the process, we also included the baseline performance by selecting all the features for both linear and random forest regression. Table IV shows the performance of different models in terms of RMSE value and captured buggy classes. Although baseline models are performing great in terms of RMSE value and capturing buggy classes, the LRCC model under the filter method with only two features also achieved very similar results except for the fact that it could not capture all the buggy classes like one of the baseline models (RFAF). With wrapper methods, as low as four features were sufficient to detect all of the buggy classes in random forest-based feature selection models, and the increase in the number of features up to seven captured more buggy classes compared to four feature sets for linear regression-based models. Considering more than seven features did not significantly increase model performance for both linear and random forest-based models. Applying the embedded method for the RFTF model gave better performance than the filter method but it was not as good as some of the wrapper methods (random forest-based feature selection methods) in terms of buggy class detection. Model-wise feature selection can be found in table V. As we can see from table V, the top three most chosen features by various models were age with respect to, number of versions until, and avg code churn until.

VI. Conclusion and Future Works

In this study, we developed machine-learning techniques to prioritize time and monetary resource allocation in ensuring secure and bug-free software development. In the process, we identified public software repository data set and utilized it with various feature selection methods to identify top features to be considered while predicting bug frequency and bug localization within the space of class-level granularity. Our study showed promise in both tasks.

One of the interesting facts about machine learning models is that they often act quite differently with a larger data set or with different techniques used for a particular task. Therefore, one of our immediate future works would be to find out how the findings from different machine learning models used in this study evolve for a bigger data set with many more available features. At the same time, we need to obtain results from other types of feature selection techniques, compare those results with ours and document key findings.

Another point is that the data set on which we worked was imbalanced and skewed towards non-buggy classes. About 20% of the class-wise entries were faulty or buggy classes for the entire data set. Almost the same ratio exists for the validation data set too. Due to this imbalance in the data set, machine learning models may provide biased results. To overcome this, we need to come up with some techniques that deal with class imbalance i.e., the Synthetic Minority Oversampling Technique (SMOTE). It would be interesting to observe how the outcome changes with a more balanced data set.

VII. Acknowledgment

This research work was funded by Cisco University Research Program Fund (Grant #2022-244141).

References

[1] H. Krasner, “The cost of poor quality software in the us: A 2018 report,” https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/, accessed: 03-11-2023.

[2] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artificial Intelligence*, vol. 97, no. 1, pp. 273–324, 1997, relevance. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437029700043X

[3] ——, *The Wrapper Approach*. Boston, MA: Springer US, 1998, pp. 33–50. [Online]. Available: https://doi.org/10.1007/978-1-4615-5725-8_3

[4] I. R. Subramanian and A. Elisseeff, “An introduction to variable and feature selection,” *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[5] N. Gayatri, S. Nickolas, and A. V. Reddy, “Feature selection using decision tree induction in class level metrics dataset for software defect predictions,” in *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, USA, 2010, pp. 124–129.

[6] G. Czibula, Z. Marian, and I. G. Czibula, “Software defect prediction using relational association rule mining,” *Information Sciences*, vol. 264, pp. 260–278, 2014, serious Games. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513008876

[7] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, “An empirical study on pareto based multi-objective feature selection for software defect prediction,” *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300573

TABLE IV
PERFORMANCE OF DIFFERENT MODELS IN TERMS OF RMSE VALUE AND CAPTURED BUGGY CLASSES

	Baseline		Filter	Wrapper								Embedded
	LRAF	RFAF	LRCC	LRFFS		LRBFE		RFFFS		RFBFE		RFTF
				4-F	7-F	4-F	7-F	4-F	7-F	4-F	7-F	
RMSE[b]	0.72	0.71	0.71	0.73	0.73	0.77	0.81	0.82	0.81	0.86	0.81	0.72
RMSE[a]	0.78	0.85	0.82	0.81	0.78	0.93	0.87	0.83	0.88	0.83	0.88	0.84
Revised RMSE[b] (exp-1)	0.5	0.46	0.47	0.43	0.48	0.53	0.51	0.46	0.47	0.46	0.47	0.48
Bug Class[b] (exp-2)	40	41	37	41	40	38	39	41	41	41	41	40

[a]value obtained on test data set.
[b]value obtained on validation data set.

TABLE V
SELECTED FEATURES IN DIFFERENT MODELS

	Filter	Wrapper								Embedded
	LRCC	LRFFS		LRBFE		RFFFS		RFBFE		RFTF
		4-F	7-F	4-F	7-F	4-F	7-F	4-F	7-F	
age with respect to		✓	✓	✓	✓	✓	✓	✓	✓	✓
number of versions until	✓	✓	✓			✓	✓	✓	✓	✓
avg code churn until		✓	✓	✓	✓	✓	✓	✓	✓	
code churn until	✓	✓	✓	✓	✓					✓
max lines added until						✓	✓	✓	✓	✓
number of fixes until			✓				✓		✓	✓
weighted age with respect to			✓		✓					✓
number of refactoring until							✓		✓	
number of authors until				✓	✓					
lines added until					✓					✓
lines removed until					✓					✓
max lines removed until							✓		✓	
avg lines removed until			✓							✓
avg lines added until										✓
max code churn until										

[8] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, 2020.

[9] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Information and Software Technology*, vol. 114, pp. 204–216, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584919301466

[10] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 297–308. [Online]. Available: https://doi.org/10.1145/2884781.2884804

[11] M. Tsunoda, A. Monden, K. Toda, A. Tahir, K. E. Bennin, K. Nakasai, M. Nagura, and K. Matsumoto, "Using bandit algorithms for selecting feature reduction techniques in software defect prediction," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 670–681.

[12] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Softw. Engg.*, vol. 17, no. 4–5, p. 531–577, aug 2012. [Online]. Available: https://doi.org/10.1007/s10664-011-9173-9

[13] L. Jia, "A hybrid feature selection method for software defect prediction," *IOP Conference Series: Materials Science and Engineering*, vol. 394, no. 3, p. 032035, jul 2018. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/394/3/032035

[14] M. A. Mabayoje, A. O. Balogun, A. O. Bajeh, and B. A. Musa, "Software defect prediction: Effect of feature selection and ensemble methods," *FUW Trends in Science & Technology Journal*, vol. 3, no. 2A, pp. 518–522, 2018.

[15] H. Osman, M. Ghafari, and O. Nierstrasz, "Automatic feature selection by regularization to improve bug prediction accuracy," in *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, 2017, pp. 27–32.

[16] S. Strüder, M. Mukelabai, D. Strüber, and T. Berger, "Feature-oriented defect prediction," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, ser. SPLC '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3382025.3414960

View publication stats