A Project Report

On

Software Fault Prediction

BY

R ADARSH

2019A7PS0230H

Under the supervision of

Dr. Lov Kumar

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F366/367: LABORATORY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(May 2022)

ACKNOWLEDGMENTS

I would like to thank Dr. Lov Kumar and Dr. N L Bhanumurthy for their constant support and guidance throughout the duration of the project. This would not have been possible without their insights into the project area and their direction for the research.

I would like to extend my gratitude to my collaborator Ms. Sonika Rathi for her help and contributions in writing the paper. I would also like to thank my other collaborators Kshitij Upadhyay and Arjun Prasad for their contribution in industry project with Nucleus .

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Software Fault Prediction**" submitted by Mr. R ADARSH (ID No. 2019A7PS0230H) in partial fulfillment of the requirements of the course CS F366/367, Laboratory Project Course, embodies the work done by him under my supervision and guidance.

Date: **(Dr. Lov Kumar)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Software Fault Prediction (SFP) is one of the applications of the Data Mining and Machine Learning approach in the Software Engineering domain. Software Fault Prediction helps in building quality software with minimal testing efforts by detecting faulty components in the initial stage of Software Development. The application of Software Fault Prediction in product/software development has evolved and the availability of several different prediction models has influenced development outcomes**.** In general, the software metrics do contain redundancy, correlation, irrelevant features, and imbalanced distribution. The performance of the SFP model gets impacted due to redundancy, correlation, and irrelevance between features. We have applied 8 different sampling techniques along with 10 feature selection algorithms against 56 open-source projects. The comparative analysis is performed by applying 10 different classifiers to predict the faulty class. We have considered Accuracy, FMeasure, and AUC performance metrics to compare the performance of different models developed using the classification algorithm.

CONTENTS

INTRODUCTION

For software organizations, software quality is an important facet of maintaining software consistency. When a software system scales in size and complexity it is more common to generate software with faults, hence preserving software consistency becomes essential as well as difficult. A software engineer's principal goal is to deliver error-free software to end-users. And to ensure that software is free of flaws, it must be properly tested. Furthermore, the faults existing in the software system affect the quality and dependability of the software. Faults are considered as overhead for the system and most organizations and companies want to predict the defects in the code or identify fault-prone modules in the early phase of software development systems.

If the faults are detected early they can be fixed within time and post-production bugs can be minimized, thereby allowing for on-time distribution to customers. The ML techniques can help to predict if a class is bug-prone or not. This will help the software testers to give more attention to bug-prone modules and thus saving a lot of time and cost. Which leads to minimizing the maintenance cost and an increase in customer satisfaction. We employed the eight most prominent machine learning classifiers provided in the maximum current comprehensive literature review in this investigation. In this study, we used GNB (Gaussian Naïve Bayes), LOGR (Logistic Regression), DT (Decision Tree), RF (Random Forest) SVCR (SVM with RBF kernel function), SVCL (SVM with linear kernel function), BAG (Bagging) ADAB (AdaBoost) to train Software Fault Prediction (SFP) model and predict the faulty class in the datasets. All of the chosen classification techniques are tested on a variety of open-source datasets related to software defect prediction.

At present, object-oriented software methodologies are widely used in software development. Modern software is built using an object-oriented (OO) approach as OO source code has high code reusability, modularity, and data hiding which reduces the development time of the product and thus helps in the timely delivery of software to its customer. As software metrics are used for developing fault prediction models, their performance will also depend on these source code metrics. Hence, selecting proper subsets of software metrics will affect the overall performance of the prediction model.

The purpose of this study is to identify the impact of feature selection methods against fault predictability. The performance of the software fault prediction (SFP) models depends on these metrics, it is essential to identify the right set of metrics to optimize the performance of proposed models to detect the faulty classes. Thus we have applied feature ranking and feature subset selection techniques over a variety of datasets to identify the best software metrics/features that are useful to train the SFP models. And we have built these SFP models using the stated eight classifiers technique.In addition, to ensure fair predictions, the study conducts statistical analysis of the engendered software fault prediction models. The performance and reliability of these models were evaluated using the performance parameters: accuracy, F-measure, and AUC.

RELATED WORK

Several research has been performed in the field of software fault prediction. The main purpose of software fault prediction models is to classify a class as faulty or non-faulty. Predicting faults in a software module helps the software test resources to be utilized more effectively and efficiently in quality assurance. Ruchika M et al. proposed the machine learning methods & statistical method to predict the faulty class [4]. Haijin et al. [5] suggested a weighted Naïve Bayes on 10 different datasets and the result shows the proposed model performs significantly better. Sequential ensemble model on PROMISE and ECLIPSE repository datasets is studied by Monika Mangla et al. [6]. Other proposed software defect prediction models are SVM by (Vinod Kumar et al. [7] & Hou & Li [8]), Random forest classifier based approach by (Tianchi et al.; Khanh et al. [9,10]), Multilayer Perceptron,J48, two hybrid search based algorithms by Rhmann et al. [11], Neural network approach by Zong et al.[12], etc.

Gao et al. has applied some feature selection techniques for fault prediction on large legacy software systems in telecommunications [13]. Wang et al. evaluated various ensembles of feature selection algorithms [14]. It was shown that sometimes ensembling some algorithms can prove to be better than ensembling all algorithms. Rodriguez et al. evaluated different filter approaches and wrapper approaches [15], and showed that wrapper methods involve higher computation cost but they outperformed filter methods. A comprehensive inspection of feature selection algorithms is published by Liu et al. [16]. Issam et al. [17] recommended Software defect prediction using ensemble techniques on selected features and the result shows the remarkable performance of the proposed model. Hall et al. [18] reported a comparison of six attribute selection strategies that creates feature ranked lists. The findings of the experiments suggest that attribute selection improves the performance of learning algorithms in general. Shivaji et al. [19] examined multiple feature selection techniques using SVM & Naïve Bayes classifiers.

To construct the software failure prediction model, Yohannese et al. used feature selection approach and SMOTE.In addition, they utilized an ensemble approach to predict the faulty class by integrating the outputs of all trained classifiers. Weiss et al. studied methods to deal with highly skewed class distribution and non-uniform classification costs. As a part of this study methods to reduce costs and sampling (oversampling and undersampling) were investigated. Lina et al. to solve the class imbalance problem they proposed the K-Means Cluster-based Over-sampling with noise filtering approach and revealed their proposed model to obtain better Recall and bal values. Ruchika M et al. performed the study for improving software defect prediction by handling the imbalanced data using different oversampling techniques and also proposed the SPIDER3 sampling technique. They compared the performance and impact of different oversampling techniques over software fault prediction models. A. Joon et al. presented the optimized software fault prediction by combining basic noise removal, imbalanced class distribution, and software metrics selection strategies. They employed the SMOTETomek resampling technique in their proposed study.

RESEARCH BACKGROUND

Feature selection techniques

Following attribute selection techniques have been implemented and compared as a part of this study:

oneR: oneR is a rule based algorithm that generates a rule for predictor variables and ranks the features based on classification rates. The attributes having minimum error rates/ greater classification rates are considered important and chosen for classification tasks.

Relief: Relief algorithm takes a filter based approach to feature subset selection. It is sensitive to feature correlations and the fundamental idea behind the algorithm is to estimate the quality of features based on feature interactions. The algorithm makes use of feature weights as foundation for selecting attributes.

Gain Ratio: It is a modification of information gain which helps in reduction of bias. Gain ratio accounts for the intrinsic information of a split. Gain ratio considers the size of branches as well as its number while selecting an attribute.

Info-Gain: Information gain indicates the reduction in entropy.

Chi-square: Chi-square test is used to test the independence of events.The chi-square statistic value can be used to rank the feature according to significance. Higher value indicates greater degree of dependence of feature on response and thus more significance.

Ada-Boost: Ada-boost fits a classifier on a dataset and assigns weights to the classifier. Based on this, it finds a set of best weak classifiers for given training data.

CFS: CFS gives a metric to evaluate the efficacy of a feature subset. This method of feature selection chooses a subset of features that are greatly correlated with the class.

Consistency-based: Focuses more on inconsistency measures of feature subset instead of goodness of feature subset. A feature subset is considered inconsistent if there exists at least 2 instances having same feature values but different class labels.

Logistic Regression Analysis: Univariate logistic regression is used to determine the importance of attributes. The model selects the features based on their p-values. Features having p-value < 0.05 are considered to be of more relevance.

NaiveBayes: Naive Bayes is a supervised algorithm and is based on Bayes theorem and is mainly used for solving classification tasks.It classifies based on probability of an object. The model works by determining conditional probabilities and unconditional probabilities of features to determine the class with highest probability.

Sampling techniques:

SMOTE (Synthetic Minority Over Sampling Technique): SMOTE is a sampling method that solves the class-imbalance problem by oversampling the examples from minority classes. It is one of the most effective types of data augmentation. SMOTE works by first selecting a minority class sample from a data set and then synthesizing a new instance by choosing one of the k-nearest neighbors of the chosen minority class instance.

BSMOTE (Borderline SMOTE): Borderline-SMOTE is a variation of the SMOTE. BSMOTE generates the synthetic instances of minority classes by choosing nearest neighbors that lie on the decision boundary between classes.

SSMOTE (SVM-SMOTE): It is a variation of BSMOTE technique. In this method, instead of choosing from k-nearest neighbors, the technique uses the SVM algorithm to identify misclassification boundaries.

SMOTEE (SMOTE- Edited): SMOTEE combines oversampling and undersampling techniques for imbalanced classification. It uses SMOTE for oversampling and Edited Nearest Neighbor (ENN) for undersampling. **ENN** removes samples having different class labels than that of the majority of their k nearest neighbors**.**

SMOTETO (SMOTE with Tomek's link): In this method, first SMOTE oversampling is applied and then Tomek Links examples from the majority classes removed. Tomek links identifies a pair of nearest neighbors belonging to different classes. The decision boundary in the train dataset tends to become less noisy/ambiguous when one or both examples from these pairs are removed.

USAM: Upsampling technique duplicates records from minority classes and injects into the dataset to prevent the model from inclining towards majority class.

RSAM: Under Random sampling, every element of the population has an equal probability of getting selected.

DSAM: Down sampling refers to training on a disproportionately low subset of majority class examples.

Classification Techniques:

GNB : Gaussian Naive Bayes is a variant of Naive Bayes that makes an assumption that every class follows a normal distribution.

LOGR: Logistic Regression is a supervised learning technique used to predict the probability of a target variable.

DT: Decision trees come under supervised learning. They are constructed in a manner that data is continuously split based on some parameter.

RF: Random forest utilizes ensemble learning. It consists of a collection of decision trees. Bagging or Bootstrap aggregation is used to train the forest generated.

SVCR: SVM algorithm intends to find a hyperplane in N-dimensional space that distinctly classified data points. SVCR uses SVM with a radial basis function which is a kernel function to find non-linear classifier/regression boundary.

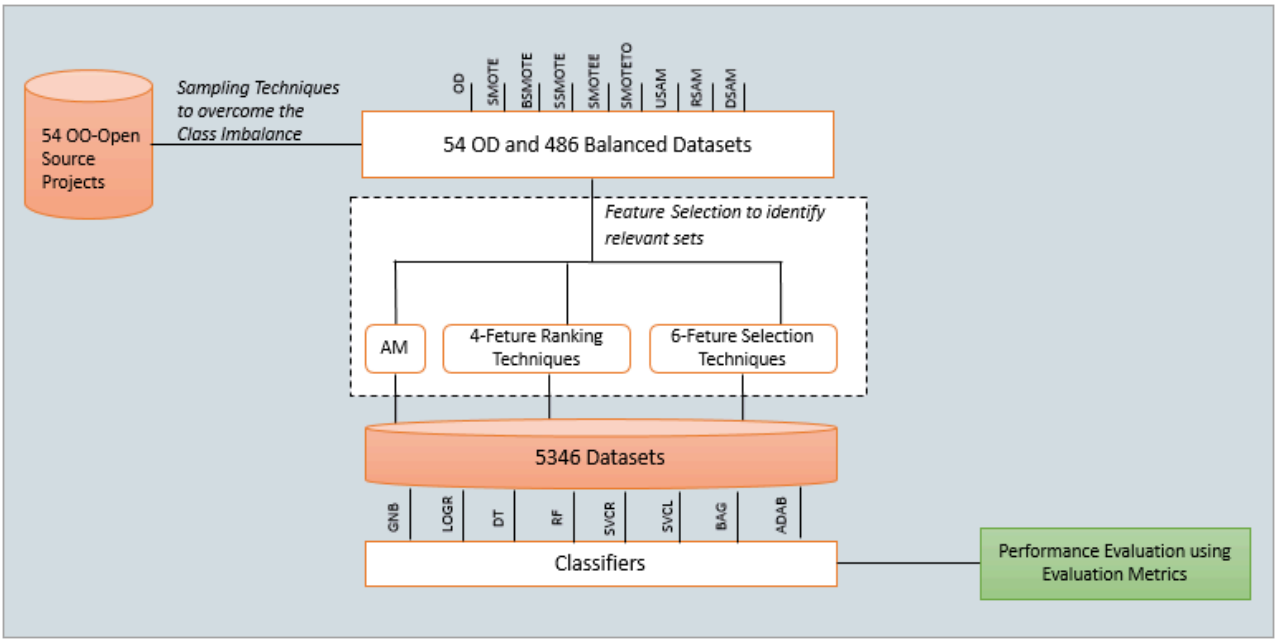
SVCL: SVM algorithm intends to find a hyperplane in N-dimensional space that distinctly classified data points. SVCR uses SVM with a linear basis function which is a kernel function to find linear classifier/regression boundary

BAG: Bootstrap Aggregation (also known as Bagging) is a simple and revolutionary outfit technique. Bootstrap Aggregation is a common method that can be used to reduce the variation of calculations for a large variance.

ADAB: AdaBoost employs a lot of classifiers for increasing classifier accuracy. AdaBoost is a program that allows you to create iterative ensembles.

WORKING OF MODEL

The overall Software Fault Prediction model framework is shown in figure 1, using software metrics as input. The faulty software metrics data were collected for 54 open-source object-oriented software projects.



Against the balanced datasets, we considered applying Feature Selection Techniques as illustrated in the above framework, to identify relevant sets and minimize the impact of class irrelevance by removing correlated and redundant features. During Feature Selection implementation we followed Feature Ranking and Feature Subset Selection methods. In the Feature Ranking method, we applied four techniques i.e. oneR, Relief, Gain Ratio, and Info-Gain to extract the top log2(N) features useful for the classification task whereas for Feature Subset Selection we applied Chi-square, Ada-Boost, CFS, Consistency, Logistic and Naive Bayes to develop a subset of best features from each method. For comparative analysis and validating the predictive power of developed SFP models we do consider relevant features with all feature metrics as well. This results in a total of 5346 datasets to analyze and measure the performance of SFP models developed using 8 different classifiers.

A comparative analysis was also carried out to evaluate the performance of the various sampling techniques and feature selection methods. We have applied the most commonly used standard classifiers [references] like GNB (Gaussian Naïve Bayes) LOGR (Logistic Regression), DT (Decision Tree), RF (Random Forest) SVCR (), SVCL (), BAG (Bagging) ADAB (AdaBoost) to predict the faulty class in the datasets. Consequently, the performance and reliability of these models were evaluated using the performance parameters: accuracy, F-measure, and AUC.

RESULTS: Comparative Study

Comparison of feature selection techniques using descriptive statistics and boxplots:

Figure 5 depicts the boxplot for the Accuracy and AUC of 10 feature selection techniques applied on balanced data and for data without any feature selection techniques applied. The following conclusions can be drawn from Fig. 5 :

- All the models give reasonable accuracies ranging between 70-80 % and AUC values between 0.75-0.85.
- The median accuracy % and median AUC value for FS1 are higher than other techniques.
- The model trained after applying FS1 performs better than the rest of the techniques.

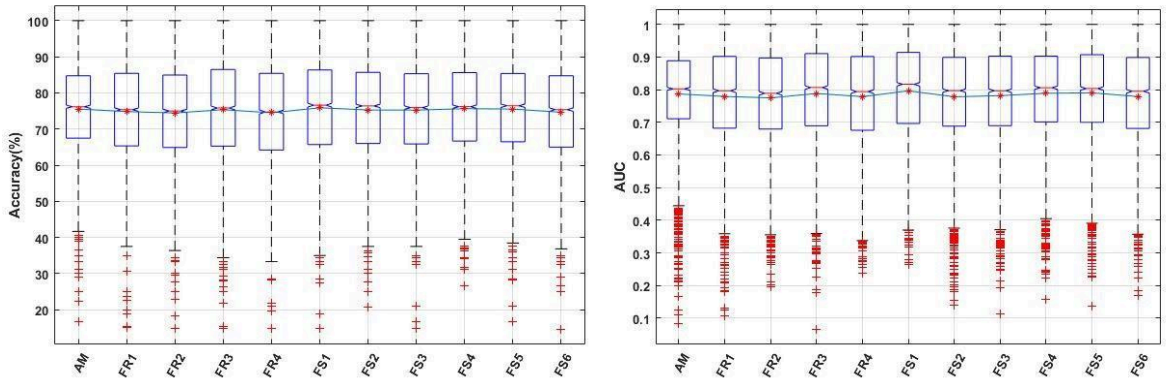


Figure 5: Accuracy and AUC Boxplots of Feature Selection Techniques

Comparison of the feature selection technique using statistical tests:

For Wilcoxon Sign Rank Test considering the null hypothesis (H0) shows that there is no significant difference in the performance of alternative feature selection methods. Reject H0 only if $p \leq 0.05$. p-values shows that most pairs can reject the null hypothesis. Therefore, the feature selection techniques used to reduce the dimensions of balanced source code metrics appear to be very different. Therefore, all ten feature selection techniques should be considered when developing the best-performing predictive model for identifying defective classes.

From the Friedman test results ,we observe the lowest mean rank is 4.96 for the relevant features where all the feature metrics (AM) were considered. At the same time, we found the mean rank of FS4 with reduced dimensionality of balanced source code dataset is 5.37 which is closer to the mean rank of AM. And the mean rank for other feature selection techniques is also in the fair range. We deduced from all of this that different feature selection techniques impact the performance of fault prediction models. As a result, to build the software fault prediction models, all feature selection techniques must be used.

Comparison of sampling techniques using descriptive statistics and boxplots:

Figure 6 shows the boxplot for the Accuracy and AUC of eight sampling techniques applied on 54 open source projects datasets along with a box plot for original data without any sampling techniques applied. From Fig 5, we deduce the following observations:

- The median accuracy % and median AUC value for the SMOTEE method are higher than other techniques.
- The model trained after applying SMOTEE performs better than the rest of the techniques.

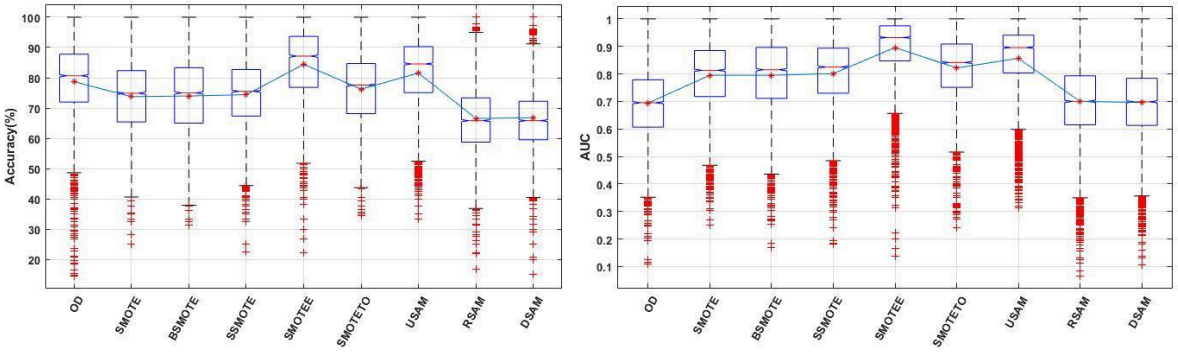


Figure 6: Accuracy and AUC Boxplots of Sampling Techniques

Comparison of the sampling technique using statistical tests:

To compute statistically significant differences between the various pairings of sampling techniques performance, we employed the Wilcoxon signed-rank test in this study. The Wilcoxon Sign Rank test's null hypothesis (H0) is that "there is no significant difference in the performance of different sampling strategies." If $p \leq 0.05$, we reject the null hypothesis. p values show that the null hypothesis can be rejected for all pairs of sampling methods, with the exception of (SMOTE, BSMOTE), (BSMOTE, SSMOTE), and (DSAM, RSAM), which all have p values > 0.05. As a result, we conclude that the performance of the majority of sampling approaches appears to differ significantly.

The Friedman test assists us in finding the most appropriate sampling strategy. Friedman Test findings show that the SMOTEE approach has the lowest mean rank of 1.87. We can infer that SMOTEE has the best overall performance in resolving the problem of class imbalance.

Comparison of classifier models using descriptive statistics and boxplots:

The boxplot of the eight distinct classifiers used to train the Software Fault Prediction models with the Accuracy and AUC performance parameters is shown in Figure 7. The following conclusions can be drawn from Fig. 7:

- The median accuracy % and median AUC value for the RF (Random Forest) classifier is higher than other techniques.
- From the descriptive statistics we observed that the Random Forest classifier performs better than the rest of the other classifiers.

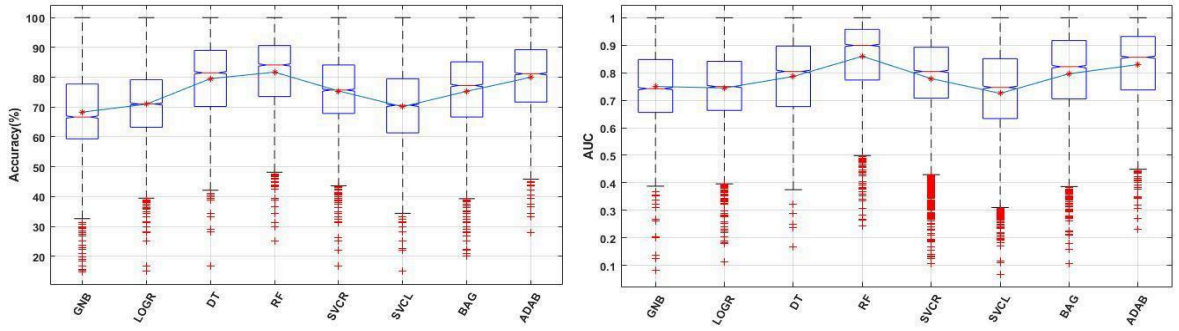


Figure 7: Accuracy and AUC Boxplots of employed eight classifiers

Comparison of the classification techniques using statistical tests:

.For the Wilcoxon Sign Rank, the null hypothesis (H0) states that there's no great difference in overall performance trained software fault prediction models using various classifiers. If the value of $p \leq 0.05$, we reject the null hypothesis. In reference to p-values, we derived the null hypothesis that can be rejected for all pairs of classifiers except for (SVCR & BAG) and (DT & ADAB), which has a p-value > 0.05. As a result, we concluded the performance of various classifier algorithms differs statistically.

From the Friedman test results, we observe the lowest mean rank is 1.93 for the Random Forest (RF) classifier. In this study, we have determined the RF classifier has shown significant performance in comparison to other classifiers.

*INDUSTRY ENGAGEMENT : NUCLEUS SOFTWARE

Past Work

OO Metrics

In the previous semester , we had generated the object oriented metrics from the defect data provided by the organization. We used CKJM extended tools to compute the OO metrics. The CKJM extended tool calculated 19 size and structure software metrics by processing the bytecode of compiled Java files. Apart from this, the MetricsReloaded plugin was used to generate the CK metrics and MOOD metrics. MetricsReloaded is a plugin for IntelliJ IDEA that automatically generates source code metrics for multiple languages including Java.

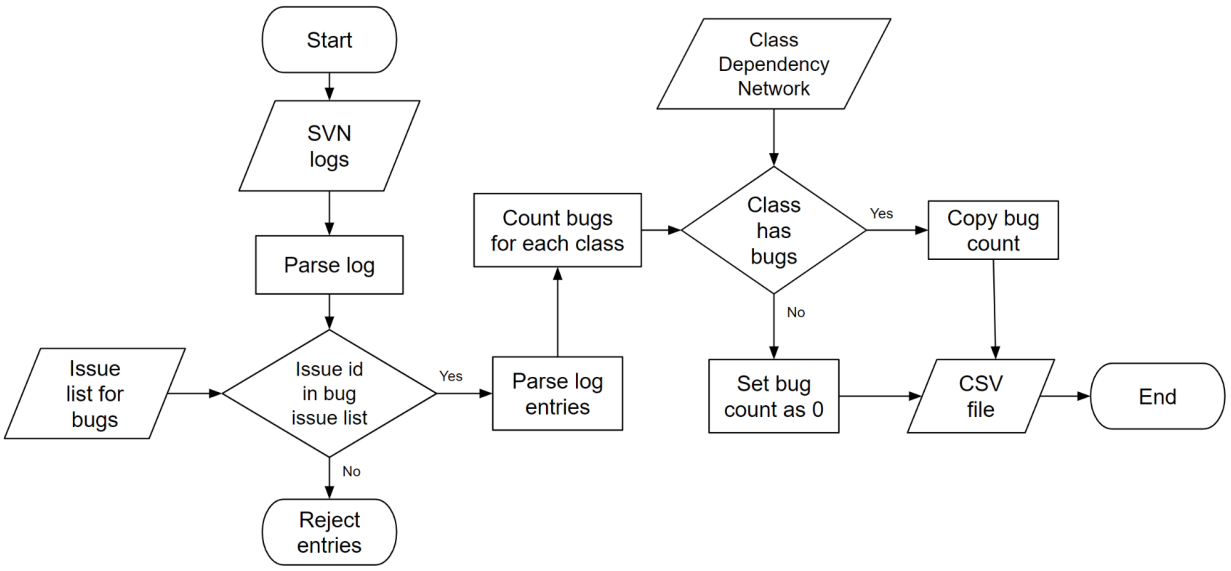
Network Embeddings

Network embeddings were obtained from the Class Dependency Network according to the steps described in the *node2defect* GitHub repository (https://github.com/quyutest/node2defect). An edgelist and a node mapping are created. The edge list is used to create network embeddings as described in the *OpenNE* GitHub repository (https://github.com/thunlp/OpenNE) and the *ProNE* GitHub repository (https://github.com/THUDM/ProNE).

Defect Data

Steps:

1. The SVN log is parsed and entries are separated.
2. Entries are filtered according to issue ID. If the issue ID is present in the issue list for bugs, it is included. Otherwise, it is excluded.
3. The changes in the included entries are parsed and a list of classes changed is obtained.
4. The defect counter for each class is incremented for every time it is changed.
5. The bug data is compiled and merged with the complete list of classes from the class dependency network.



We experimented with a lot of different models and ensembles and the best performance was obtained with the **RUSBoost** algorithm. RUSBoost is an algorithm to handle class imbalance problems in data with discrete class labels. It uses a combination of RUS (random under-sampling) and the standard boosting procedure AdaBoost, to better model the minority class by removing majority class samples.

We also performed various feature selection techniques like **Principal Component Analysis (PCA)** and using only **significant features (SIGF)** on our feature sets.

Among all these different variations of the dataset, the best performance(F1 score) was obtained when we considered **all the features** alongside the **Prone-Sparse technique** of **embedding size 32**.

Current Work

In the current semester, we aimed to extract metrics and defect data at method-level so that we can predict the defect prone modules of the code with finer granularity. Based on the literature survey, it seemed that **ChangeDistiller** tool would be an ideal fit to extract change-metrics at method level. However, due to some bugs in the tree-differentiating algorithm implemented by the tool , we could not extract the change metrics satisfactorily. Also generating defect data at method level seemed to be a difficult task. Hence we switched track and decided to work with previous semester data and decided to study the effect of class-level change metrics on performance of the model.

Change Metrics Extraction and Dataset Preparation

1. The Nucleus Team had provided the revision-wise SVN diff files , which contain information about the lines added and lines deleted in a file in a particular revision.
2. We parsed the diff files to generate the required change metrics : **locAdded , locRemoved , maxlocAdded , maxlocRemoved , avglocAdded , avglocRemoved , code churn , number of revisions**. The file names along with the metrics were stored in a CSV file.

3. We compare the dataset from the previous semester with our current generated dataset and find the intersection of files between the 2 datasets so that we can append the change metrics to the embeddings generated last time.
4. The final dataset can be used for modeling and obtaining the performance parameters.

After building the model and generating the results, it would be an interesting task to compare the performance of models and study the impact of change metrics in defect prediction.

CONCLUSION

Experimental results showed that class imbalance and class irrelevance adversely affected the performance of predictive models. The analysis of experimental outcome indicates oversampling technique (SMOTEE- proposed new method of SMOTE**) delivers good results in comparison with undersampling techniques, as undersampling techniques drop out the useful instances in the dataset randomly, which condenses the performance. Also, these experimental results showed that feature selection methods reduce computational effort and improves the performance of the overall classification.

REFERENCES

[1]T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, “Defect prediction from static code features: Current results, limitations, new approaches,” Automated Software Engineering, vol. 17, no. 4, pp. 375–407, 2010, doi: 10.1007/s10515-010-0069-5

[2]Y. Singh, R. Malhotra, Object Oriented Software Engineering, PHI Learning, 2012.

[3]K. Bhandari, K. Kumar and A. L. Sangal, "A Study on Modeling Techniques in Software Fault Prediction," 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC), 2021, pp. 6-11, doi: 10.1109/ICSCCC51823.2021.9478119.

[4] Ruchika Malhotra and Ankita Jain (2012). Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality. Journal of Information Processing Systems, 8(2), 241-262. DOI: 10.3745/JIPS.2012.8.2.241.

[5]Ji, Haijin & Huang, Song & Wu, Yaning & Hui, Zhanwei & Zheng, Changyou. (2019). A new weighted naive Bayes method based on information diffusion for software defect prediction. Software Quality Journal

[6]Mangla, Monika, Nonita Sharma, and Sachi Nandan Mohanty. "A sequential ensemble model for software fault prediction." Innovations in Systems and Software Engineer-ing (2021): 1-8.

[7]Kulamala, V.K., Kumar, L. & Mohapatra, D.P. Software Fault Prediction Using LSSVM with Different Kernel Functions. Arab J Sci Eng 46, 8655–8664 (2021). https://doi.org/10.1007/s13369-021-05643-2 [8] Hou S, Li Y (2009) Short-term fault prediction based on support vector machines with parameter optimization by evolution strategy. Expert Syst Appl 36(10):12383–12391

[9]Tianchi Zhou, Xiaobing Sun, Xin Xia, Bin Li, Xiang Chen, Improving defect prediction with deep forest, Information and Software Technology, Volume 114, 2019

[10]Khanh DH, Pham T, Wee NS, Tran T, Grundy J, Ghose A, Kim T, Kim C-J (2018) A deep tree-based model for software defect prediction

[11]Rhmann, Wasiur, Babita Pandey, Gufran Ansari, and Devendra Kumar Pandey. "Software fault prediction based on change metrics using hybrid algorithms: An empirical study." Journal of King Saud University-Computer and Information Sciences 32, no. 4 (2020): 419-424.

[12]P. Zong, Y. Wang and F. Xie, "Embedded Software Fault Prediction Based on Back Propagation Neural Network," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2018, pp. 553-558, doi: 10.1109/QRS-C.2018.00098.

[13]K. Gao, T. M. 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

[14] H. Wang, T. Khoshgoftaar, and A. Napolitano, “A comparative study of ensemble feature selection techniques for software defect prediction,” in Proceedings of International Conference on Machine Learning and Applications, 2010, pp. 135–140.

[15] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, “Detecting fault modules applying feature selection to classifiers,” in Proceedings of International Conference on Information Reuse and Integration, 2007, pp. 667– 672.

[16] H. Liu and L. Yu, “Toward integrating feature selection algorithms for classification and clustering,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, pp. 491–502, 2005.

[17] Issam H. Laradji, Mohammad Alshayeb, Lahouari Ghouti, Software defect prediction using ensemble learning on selected features, Information and Software Technology, Volume 58, 2015

[18] M. A. Hall and G. Holmes, “Benchmarking attribute selection techniques for discrete class datamining,” IEEE Transactions on Knowledge and Data Engineering, vol. 15, pp.1437 – 1447, 2003

[19] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," The 9th International Conference on Machine Learning and Applications, IEEE, Washington, DC, 2010.