PROCEEDINGS OF SCIENCE

Software defect prediction: A study on software metrics using statistical and machine learning methods

Marco Canaparo,[a,*] **Elisabetta Ronchieri**[a,b] **and Gianluca Bertaccini**[b]

[a] *INFN-CNAF,*
Viale Berti Pichat 6/2, Bologna, Italy
[b] *Department of Statistical Sciences, University of Bologna,*
Via Belle Arti, 41, Bologna, Italy
E-mail: marco.canaparo@cnaf.infn.it, elisabetta.ronchieri@cnaf.infn.it, gianlucabertaccini2@gmail.com

Software defect prediction aims at identifying defect prone software modules in order to allocate optimal testing resources. The role of testing in software development life cycle is vital especially when software systems are becoming more and more complex, representing a suitable environment for defects. Several researchers have striven to develop models able to determine defective modules with the aim of reducing time and cost of software testing. Such models are typically trained on software measurements, known as software metrics, which describe the characteristics of a software project in terms of e.g., dimension and complexity. These metrics reduce the subjectivity of software quality assessment and can be relied on for decision making, e.g., to decide where to focus software tests.

The aim of our work is to employ both feature selection or construction techniques and machine learning techniques to build software defect prediction models on different kinds of software dataset metrics (derived from various software projects available in the NASA and Eclipse repositories), and assess their performances by considering accuracy, precision, recall and area under the curve. We have used non parametric tests to compute the statistical significance of the obtained results. The collected metrics belong to three main categories: dimension, complexity and object orientation. The involved datasets contain class labels, i.e., information on the defectiveness of the software modules.

To make our study available to research community, we have developed an open source and extensible R application that supports researchers to load the selected kinds of datasets, to filter them according to the their features and to apply machine learning techniques.

International Symposium on Grids & Clouds 2022 (ISGC 2022)
21 - 25 March, 2022
*Online, Academia Sinica Computing Centre (ASGC), Taipei, Taiwan****

*Speaker

PoS(ISGC2022)020

1. Background

Software defects might be the cause of several problems during software maintenance or development, they could lead to major damage and important loss of money. Consequently, researchers have been striven to find ways to mitigate defects' influence on the source code via software testing [1] that is an essential and time-consuming activity carried out during the entire development process. The distribution of defects in software modules may vary, as a result, applying the same effort to test all the software modules of a software project would lead to a cost prohibitive activity and to non optimal results [2]. This explains the motivation behind software defect prediction (SDP). To support testers and developers in the testing phase, SDP helps find models that can be used to identify defective-prone modules [3].

In the last decade, machine learning (ML) techniques have been applied to defect prediction [4] with various approaches, such as Bayesian networks, neural networks, multivariate regression and ensemble methods [5]. Nevertheless, the construction of a SDP model is a challenging activity that has seen researches propose different solutions [6–13]. Despite the high number of approaches the vast majority of SDP models use software metrics as their main source of data. More in detail, previous studies have relied on code metrics, process metrics or previous defects: these measurements guarantee a reduction of subjectivity when evaluating defectiveness and have been relied on for decision making process [14].

In addition to ML, the attention towards feature selection techniques is increasing because of their ability to affect and improve prediction performances [15]. In previous literature, different techniques have been explored, since feature selection has been proven to play a vital role in data preprocessing when it comes to dealing with high dimensionality datasets [16–22]. Independent dataset features may be irrelevant, i.e. they may have no impact on the target features, and/or redundant, i.e. they may be highly correlated one-another, and can be removed [23].

The contribution of our work is three-fold: first it explores different feature selection and construction techniques and perform their comparison; secondly, it carries-out a comparison between different ML-based models; finally it provides an R framework to support researches working on SDP. More in detail, our framework eases datasets loading and software metrics analysis, provides guidelines and examples to simplify the application of ML techniques and allows users to reproduce our work and simplifies the integration and exploration of other techniques.

During our research we have formulated the following research questions:

RQ1 Which feature selection method performs the best?

RQ2 Can unsupervised ML techniques perform better or similarly to supervised ML techniques?

2. Methodology

The main objective of this study has been the development and validation of prediction models for software defectiveness. A generalized workflow is shown in Figure 1 that schematizes the actions that have been undertaken to implement this model. The work starts with the datasets selection, analysis and preprocessing. Then we have worked on the dataset features, built our ML-based prediction models and evaluated both the different employed features selection and extraction techniques as well as ML algorithms.

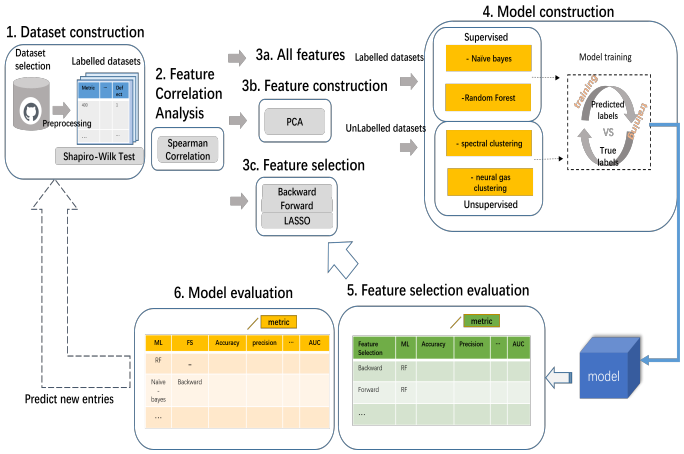

Figure 1: Methodology Overview.

2.1 Dataset Construction

In this study, two souces of data have been considered: the Eclipse repository [24] and the NASA one [25].

Our datasets have undergone a preprocessing activity that included: the removal of observations with missing or not available (i.e., NA) values, the conversion of defective categorical variables and multi-defect labels to one binary facet label. Furthermore, the Eclipse project-level datasets have been combined in a unique one.

2.1.1 Eclipse Datasets

The Eclipse datasets are related to five different software projects:

- Eclipse JDT Core: one of the most popular IDE (integrated development environment) for coding in Java and other programming languages. It is written mostly in Java.
- Equinox Framework: a Java framework for developing and deploying modular software programs and libraries. It is part of the Eclipse project.
- Apache Lucene: a Java library that provides users with indexing and search features.
- Mylyn: a framework for Eclipse providing task management tools for developers.
- Eclipse PDE UI: plug-in development environment (PDE) that provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments and features.

Table 1 shows the properties of the Eclipse datasets. There are 17 variables and a response variable, called defective, which contains the value 1 if the class is defective and 0 otherwise. Each record of the Eclipse dataset contains 6 Chidamber and Kemerer (CK) software metrics, 11 object oriented metrics and a binary defective label. All the variables, including the response, are numeric.

Variables	Information
6 CK Software metrics	CBO: coupling between objects; DIT: depth of inheritance tree; LCOM: lack of cohesion methods; NOC: number of children; RFC: response for a class; WMC: weighted method count.
11 Object oriented software metrics	e.g., number of lines, number of methods, number of attributes.
1 defective label	Binary response variable.

Table 1: The Eclipse datasets properties.

We have analysed the distribution of the values of the software metrics. Figure 2 shows the CBO histogram and the WMC quantile plot of the Eclipse project, highlighting a non-normal distribution of their values. The other metrics' distributions follow similar patterns.

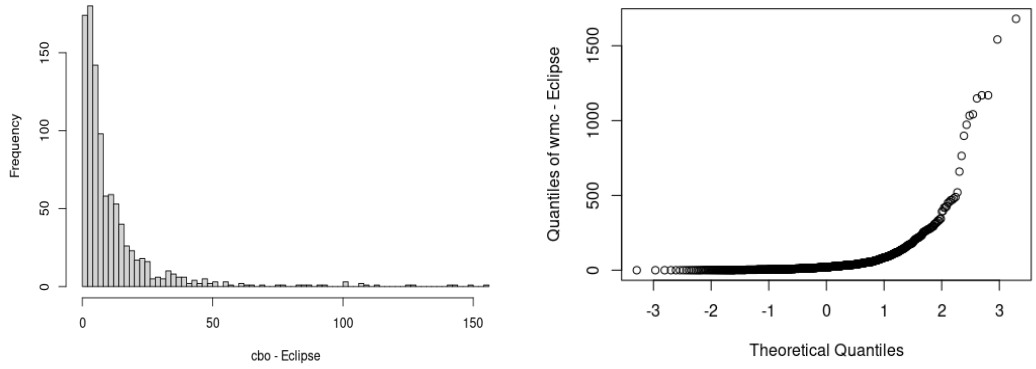

Figure 2: The CBO histogram on the left side and WMC quantile plot on the right side.

Figure 3 shows the Shapiro-Wilk boxplot for the Eclipse datasets. The closer to zero are the values, the more their distribution deviates from normality. This is true for all variables, while LCOM and number of public attributes have the strongest deviation from normality.

The highest percentage of defective classes is reached in the Equinox dataset (39.81%), while the lowest is in the Lucene dartaset (9.26%).

2.1.2 NASA Datasets

The PROMISE repository [25] is owned by NASA and contains twelve datasets, each corresponding to a software project. The included projects are:

- CM1: software developed in C for a NASA spacecraft instrument.
- JM1: software that computes simulations in order to generate real-time predictions about ground systems. Developed in C.

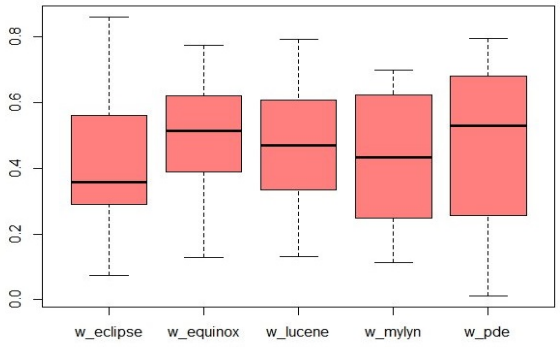

Figure 3: The Shapiro-Wilk boxplot for the Eclipse datasets.

- KC1, KC3: software for storage management developed in C++. It receives and processes ground data. It is split into two projects, where KC1 is the main one and the other is a relative component.
- MC1, MC2: no information available on these projects.
- MW1: no information available on this project.
- PC1, PC2, PC3, PC4, PC5: flight software developed for a satellite orbiting around earth. PC1 is the main project, while the others are minor projects connected to the main one. Written in C.

Table 2 shows the properties of the NASA datasets, containing different number of variables that range from 22 to 40.

Variables	Information
Lines of code metrics	counting the number of lines of code, comments, percentages
McCabe's complexity metrics	metrics computed starting from the McCabe's Cyclomatic Complexity
Halstead complexity metrics	different complexity metrics that are computed starting from the number of operands and operators.
Other complexity metrics	complexity metrics that are not based directly on McCabe's or Halstead's complexity.
Density metrics	complexity metrics with respect to the number of total paths in an instance of code.
1 defective label	binary response variable.

Table 2: The NASA datasets properties.

We have analysed the distribution of the values of the metrics: the values of NASA dataset variables are non-normally distributed, there are very few exceptions. Figure 4 shows the histogram of the ESSENTIAL_COMPLEXITY values on the left side, and the quantile plot of the HALSTEAD_EFFORT variable on the right side.

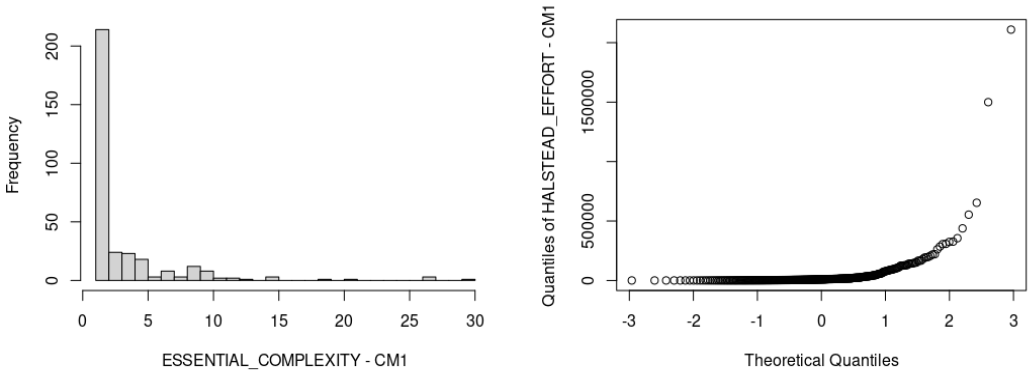

Figure 4: The ESSENTIAL_COMPLEXITY histogram on the left side and HALSTEAD_EFFORT quantile plot on the right side.

Figure 5 shows the Shapiro-Wilk boxplot for the NASA datasets: most variables in the NASA datasets are below 0.9, so they are non-normally distributed.

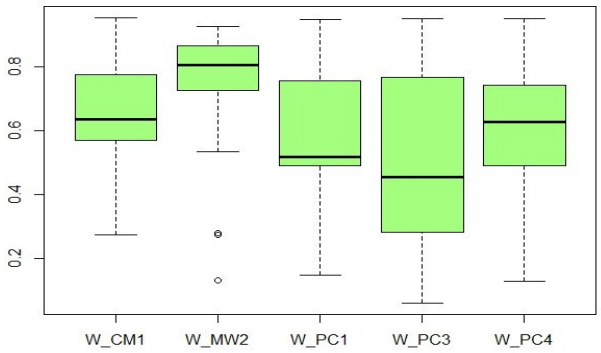

Figure 5: The Shapiro-Wilk on the NASA datasets.

2.2 Feature Correlation analysis

Given the non-normality distribution of our variables, we have conducted a variable correlation analysis. It is known that correlation between variables may negatively affect the performance of a prediction model, since our software metric values have been proved to be skewed, our expections were to notice a high variables correlation. This has been confirmed by the application of Spearman correlation.

Figure 6 shows the existing high correlations for the CK metrics, i.e., lcom, rfc, wmc, and cbo (detailed in Table 1) for the Eclipse software project. Figure 7 shows high correlations for the Halstead metrics in the NASA CM1 software project.

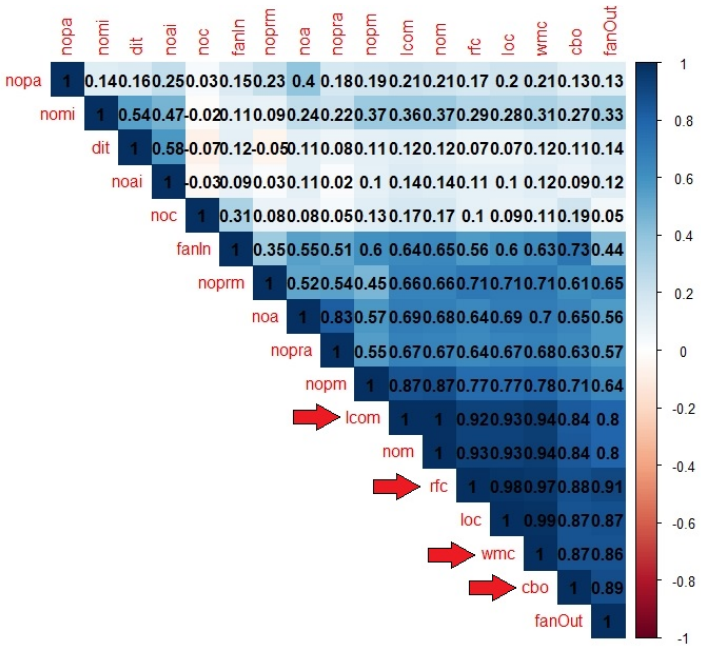

Figure 6: The correlation between software metrics for the Eclipse software project.

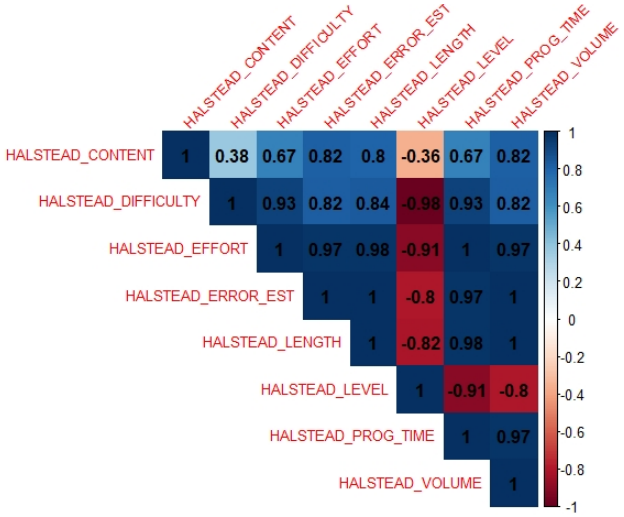

Figure 7: The correlation between software metrics for the NASA CM1 software project.

2.3 Feature Selection and Construction

Correlation between variables can be reduced by applying feature selection and/or feature construction techniques [26]. We have followed two different approaches: in the former we have used Principal Component Analysis (PCA), which creates new features as a combination of the original features provided by the dataset [27]; in the latter, we have analysed three metric selection techniques, i.e. Backward Elimination, Forward Selection and Least absolute shrinkage and selection operator (LASSO) [28].

Backward Elimination [29] is an algorithm that reduces the number of variables in a model.

First step of the algorithm consists of fitting a model containing all the independent variables and the response variable. The second step consists of fitting other models containing a variable less than the original model. For each model the Akaike Information Criterion (AIC) is computed and the best model is selected [30]. From the best model of this step, one variable at a time is removed and other models are evaluated. When do we stop? When the value of AIC does not improve significantly by removing a variable.

Forward Selection [31] works similarly to Backward elimination but in an opposite direction. First step we have a model with the only response variable. Second step, we have many models containing each one variable we have added to the model of the first step. Then a Model evaluation through AIC is done, and selected the best model to which is added another variable and so on. When do we stop? Adding a variable does not significantly improve the fit of the model.

LASSO [32] is a regression method that shrinks the parameters of the regression associated with each variable. In our context it has been used to perform variable selection.

Figure 8 shows how the different feature techniques reduce the number of features involved. Backward and LASSO selection methods tend to select more variables than the other methods, especially in the Eclipse PDE dataset and in many NASA datasets.

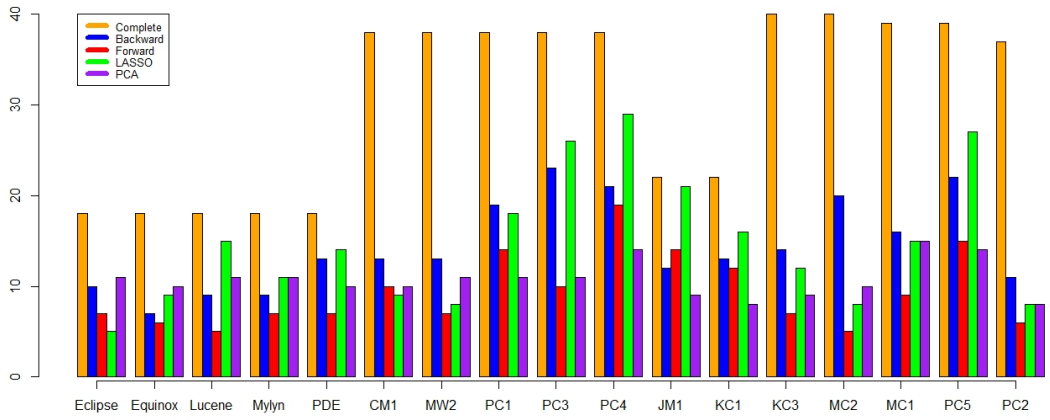

Figure 8: The number of features selected by feature techniques.

2.4 Model Construction

Our models have exploited both supervised and unsupervised techniques. Among the supervised techniques, Naive Bayes [33] and Random Forest [34] have been considered: the former is based on Bayes's theorem; the latter is an ensemble technique based on multiple decision trees. For the unsupervised techniques, Neural Gas Clustering[35] and Spectral Clustering [36] have been considered: the former is part of artificial neural network; the latter is based on connectivity.

Since our datasets are all labeled, supervised techniques would be the best choice, however, in software engineering field, software datasets are more commonly unlabelled because software projects usually lack defect labels and/or historical data, consequently, we have used unsupervised techniques and compared to the supervised ones [37].

To compare the performance of our models, a confusion matrix for every application of each technique has been computed: these applications are on 17 complete datasets, 17 selected by backward elimination, 17 selected by forward selection, 17 selected by LASSO and 17 selected by PCA. Each model has been assessed with the following performance indicators:

- **accuracy** [38] is the number of correct predictions (which are the negatives and positives correctly identified) with respect to the total population;
- **precision** [39] is the proportion of instances that are relevant between the ones that have been considered positive;
- **recall** [4] is the proportion of relevant instances classified correctly;
- **area under the curve** (**AUC**) [40] plots the recall (or true positive rate) against the false positive rate.

2.5 Feature Selection and Construction Evaluation

Figure 9 shows the accuracy values for all the ML techniques, highlighting that Random Forest has obtained the best results in every feature selection method. PCA gives the worst results when used with Spectral Clustering.

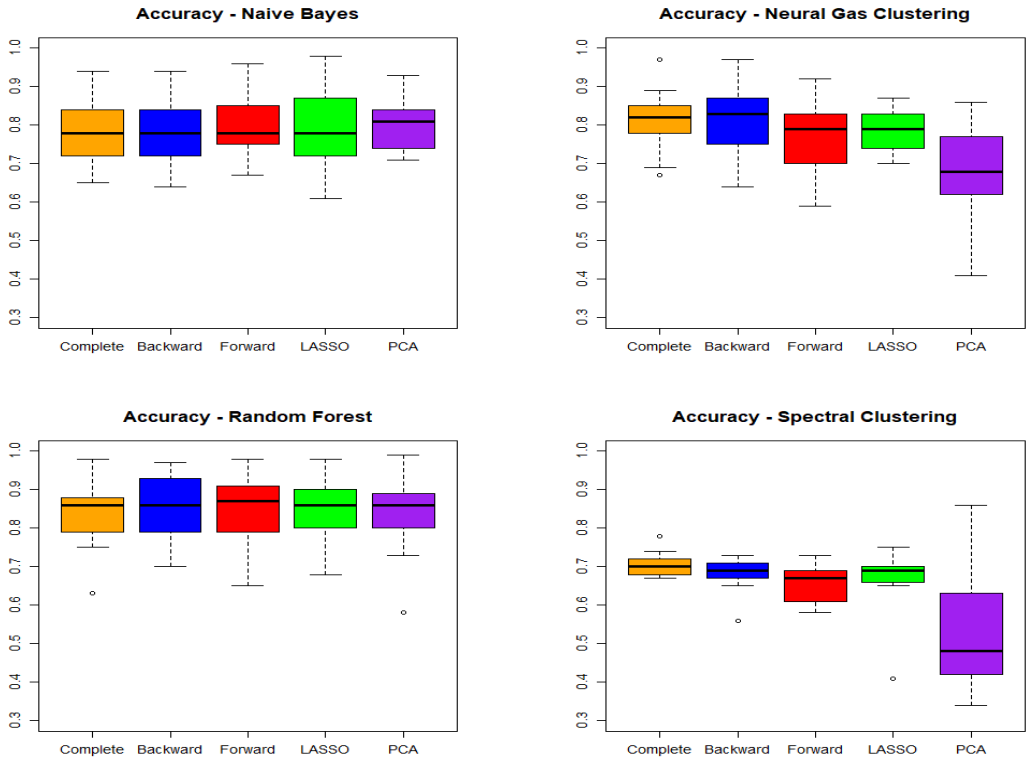

Figure 9: The accuracy performance indicator - ML techniques are reported from the top left to the bottom right starting with the Naive Bayes up to the Spectral Clustering.

Figure 10 shows the precision values for all the ML techniques, highlighting that Random Forest has achieved the best results. On average all methods achieve values greater than 0.8:

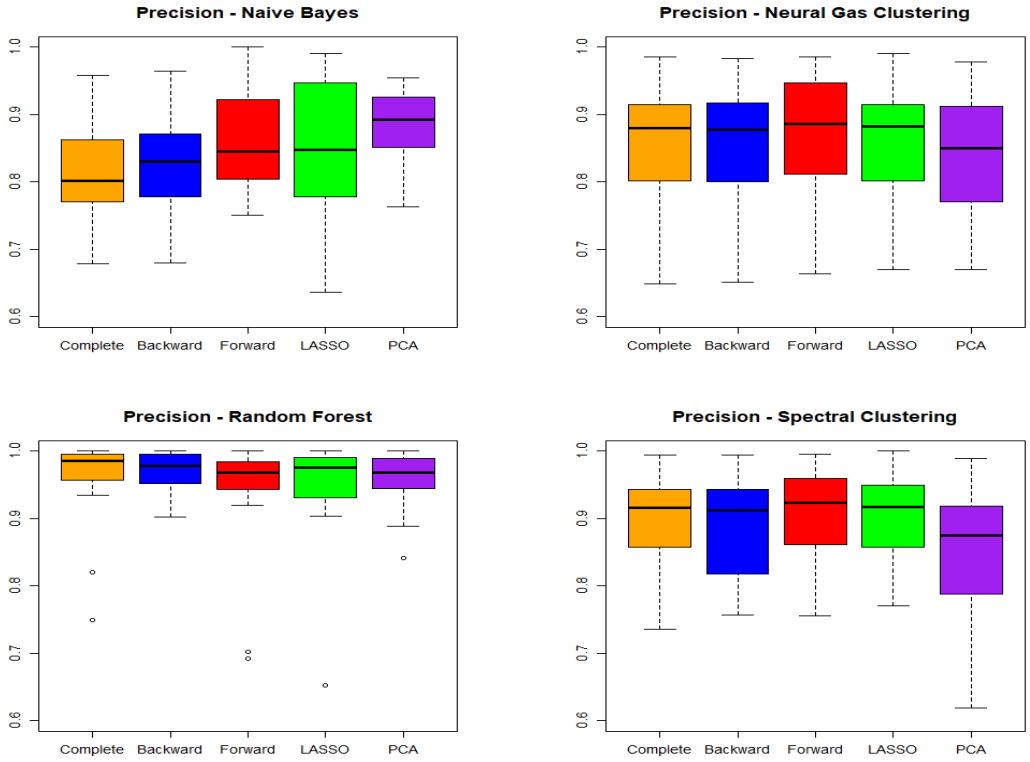

Figure 10: The precision performance indicator - ML technique are reported from the top left to the bottom right starting with the Naive Bayes up to the Spectral Clustering.

Figure 11 shows the recall values for all the ML techniques, highlighting that except Spectral Clustering, recall values are very high (about 0.9). Most defective classes are correctly identified. PCA has a negative influence on unsupervised methods.

Figure 12 shows that AUC values are on average lower that other performance indicators. The Spectral Clustering technique achieves 0.65. PCA performs worse that other methods when combined with unsupervised methods.

Since the results are not so clear-cut, it is necessary to explore them in depth considering all the evaluation measures when necessary (instead of considering e.g. only accuracy or AUC), and using nonparametric methods.

How can we select the best feature selection technique for each ML technique? We have used the average values of our performance indicators. The pairs between feature selection and ML technique are listed in table Table 3. Figure 13 shows the results of the best feature selection technique for each ML technique with respect to AUC.

We have applied the Friedman test [41] on the results of each ML technique. The Friedman test showed that for supervised methods different feature selection methods do not influence the performance of the algorithm; on the contrary, for unsupervised methods, the choice of a feature selection or construction technique has an impact on the algorithms.

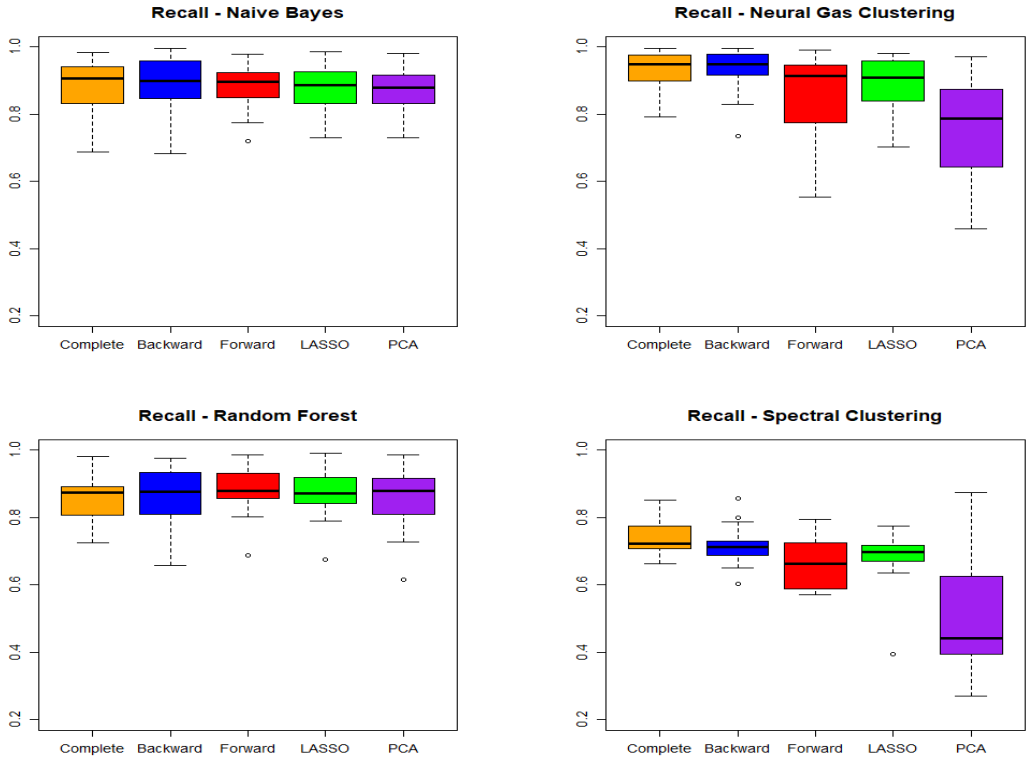

Figure 11: The recall performance indicator - ML technique are reported from the top left to the bottom right starting with the Naive Bayes up to the Spectral Clustering.

Since the Friedman test does not show the best selection method, we have applied the Nemenyi test [42] on the two unsupervised methods. This test has selected Backward Elimination and Forward Selection as the best performing feature selection method for Neural Gas Clustering. As regards Spectral Clustering, the complete set of features or LASSO are the best choices for improve the model performance.

2.6 Model comparison

Unsupervised techniques are the most used in literature when dealing with software defect prediction, because labels are not always available for software systems. As a consequence, we have detailed the AUC results. Spectral Clustering achieves a performance comparable to supervised techniques.

According to the AUC results shown in Figure 13 on the bottom right side, we have applied the Friedman test, that indicated a statistical difference in the results of the methods. Consequently, we have applied the Nemenyi test for multiple comparison that computed a critical distance CD = 1.138, as shown in Figure 14.

Table 4 shows which model - composed by feature selection and ML technique - performs best according to the results obtained by applying the Nemenyi test. Spectral Clustering is the best technique when applied on the whole dataset and there is no significant difference between the

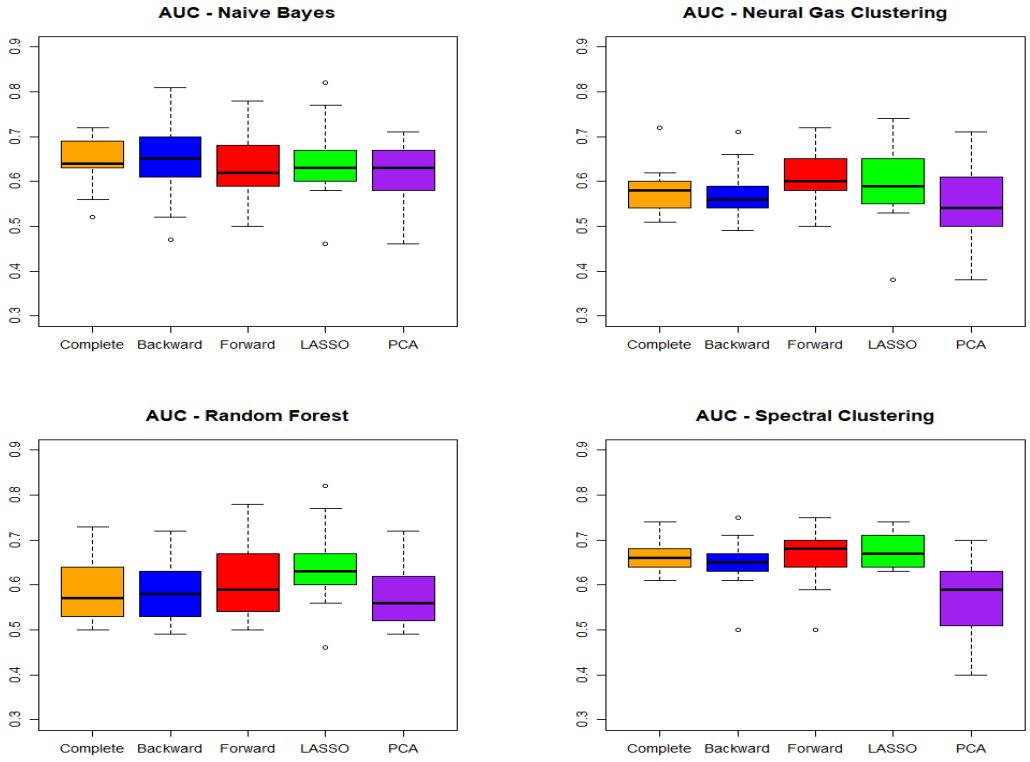

Figure 12: The AUC performance indicator - ML technique are reported from the top left to the bottom right starting with the Naive Bayes up to the Spectral Clustering.

performance of Random Forest, Naive Bayes and Spectral Clustering since their ranking difference is not bigger than CD. The same conclusions can be drawn by analysing Figure 14

In conclusion, our purpose has been to employ both supervised and unsupervised methods and comparing them by using different feature selection and construction techniques. Both Friedman Test and Nemenyi test have been used to assess the overall performance. Spectral Clustering on the all feature datasets has shown better or similar performance to Naive Bayes and Random Forest which use respectively Backward Elimination and Forward Selection. The tests conducted with Nemenyi test for ML techniques show that their difference in performances is not statistical significant. On the contrary, Neural Gas Clustering is found to be the worst performing algorithm to identify defective classes.

3. Results and Discussions

In this section we first provide an answer to our research questions.

RQ1: Which feature selection method performs best? There is no feature selection or construction method that can perform better than others. Using Naive Bayes, Backward Elimination seems the method that achieves the highest score; applying Random Forest, Forward Selection is the technique that performs the best. Regarding unsupervised methods, Neural Gas Clustering works better with Backward Elimination and Spectral Clustering with the complete set of features.

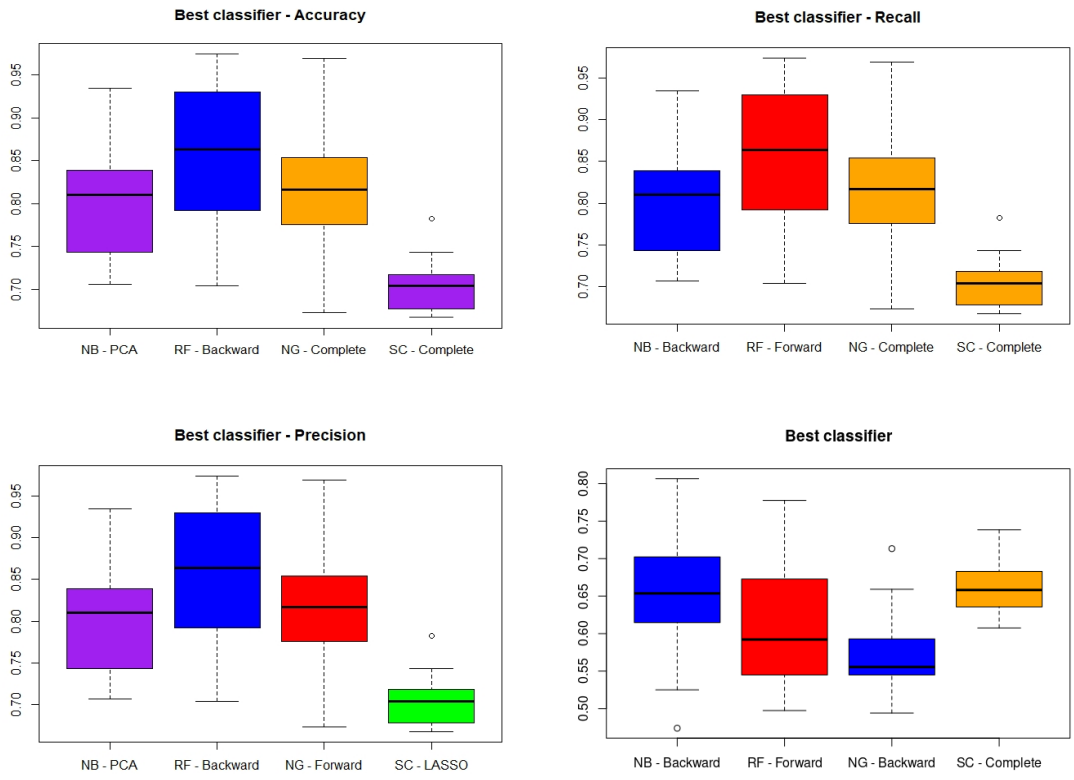

Figure 13: The best feature selection technique for each ML technique with respect to performance indicator - performance indicators are reported from the top left to the bottom right starting with accuracy up to AUC.

RQ2: Can unsupervised ML techniques perform better or similarly to supervised ML techniques? Spectral Clustering is the unsupervised method that has proven to perform similarly to supervised methods. Using all the variables of the datasets, it achieves an average of 0.66 for AUC. Whereas Naive Bayes score 0.65 average AUC with backward selection and Random Forest 0.61 with Forward Selection.

Furthermore, we provide some details on the developed R framework available on github [43]. The main R package is called *rsma* that includes a set of functions to run different statistical learning algorithms over the supported datasets. In addition to Eclipse and NASA software datasets, other software metrics are supported. To make easy its usage, a small demo is available in the *rsma.demo* github project.

4. Conclusions

In this paper we have employed both feature selection or construction techniques and machine learning techniques to build software defect prediction models on the Eclipse and NASA data software metrics, and assess their performances by considering accuracy, precision, recall and area under the curve. We have also used non parametric tests to compute the statistical significance of the obtained results.

Performance Indicator	ML Technique	Feature Selection
Accuracy	Naive Bayes	PCA
Accuracy	Random Forest	Backward elimination
Accuracy	Neural Gas Clustering	Complete feature set
Accuracy	Spectral Clustering	Complete feature set
AUC	Naive Bayes	Backward elimination
AUC	Random Forest	Forward Selection
AUC	Neural Gas Clustering	Backward elimination
AUC	Spectral Clustering	Complete feature set
Precision	Naive Bayes	PCA
Precision	Random Forest	Backward elimination
Precision	Neural Gas Clustering	Forward Elimination
Precision	Spectral Clustering	LASSO
Recall	Naive Bayes	Backward Elimination
Recall	Random Forest	Forward Selection
Recall	Neural Gas Clustering	Complete feature set
Recall	Spectral Clustering	Complete feature set

Table 3: Performance Indicator with respect to ML and feature selection techniques

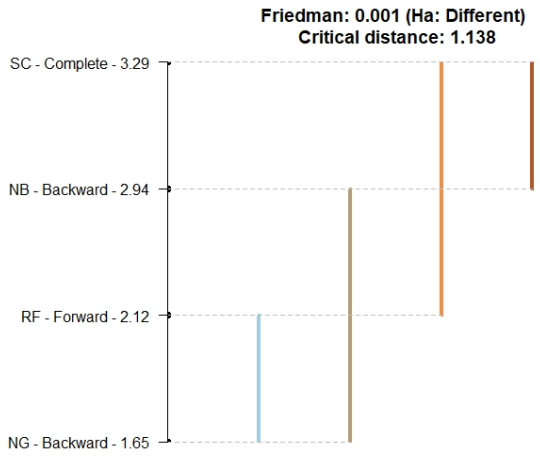

Figure 14: Nemenyi multiple comparison - the best performing classifiers according to AUC.

There is no feature selection or construction method that can perform better than others. Using Naive Bayes, Backward Elimination seems the method that achieves the highest score; applying Random Forest, Forward Selection is the technique that performs the best. Regarding unsupervised methods, Neural Gas Clustering works better with Backward Elimination and Spectral Clustering with the complete set of features.

Spectral Clustering is the unsupervised method that has proven to perform similarly to super-

Rank 1 (worst)	Rank 2	Rank 3	Rank 4 (best)
NG - Backward	RF - Forward	NB - Backward	SC - Complete
1.647	**2.118**	**2.941**	**3.294**

Table 4: Nemenyi test.

vised methods. Using all the variables of the datasets, it achieves an average of 0.66 for AUC. Whereas Naive Bayes score 0.65 average AUC with backward selection and Random Forest 0.61 with Forward Selection.

To make our study available to research community, we have also developed an open source and extensible R application called *rsma* that supports researchers to load the selected kinds of datasets, to filter them according to the their features and to apply machine learning techniques.

References

[1] Geanderson Esteves, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. Understanding machine learning software defect predictions. *Automated Software Engineering*, 27(3-4):369–392, oct 2020.

[2] Lipika Goel, D. Damodaran, Sunil Kumar Khatri, and Mayank Sharma. A literature review on cross project defect prediction. In *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, pages 680–685, 2017.

[3] Alexandre Boucher and Mourad Badri. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology*, 96:38–67, 2018.

[4] Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, and Jin Liu. Dictionary learning based software defect prediction. ICSE 2014, page 414–423, New York, NY, USA, 2014. Association for Computing Machinery.

[5] Meng Yan, Mengning Yang, Chao Liu, and Xiaohong Zhang. Self-learning change-prone class prediction. In *SEKE*, 2016.

[6] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[7] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *2009 IEEE 31st International Conference on Software Engineering*, pages 78–88, 2009.

[8] Ming Li, Hongyu Zhang, Rongxin Wu, and Zhi-Hua Zu. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, 19:201–230, 2012.

[9] Huihua Lu and Bojan Cukic. An adaptive approach with active learning in software fault prediction. In *Proceedings of the 8th International Conference on Predictive Models in*

Software Engineering, PROMISE '12, page 79–88, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. *Information and Software Technology*, 87:206–220, 2017.

[11] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 17–26, 2015.

[12] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. Collective personalized change classification with multiobjective search. *IEEE Transactions on Reliability*, 65(4):1810–1829, 2016.

[13] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. An empirical study of classifier combination for cross-project defect prediction. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 264–269, 2015.

[14] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41, 2010.

[15] Hadeel Alsolai and Marc Roper. A systematic review of feature selection techniques in software quality prediction. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–5, 2019.

[16] Mitja Gradišnik, Tina Beranič, and Sašo Karakatič. Impact of historical software metric changes in predicting future maintainability trends in open-source software development. *Applied Sciences*, 10(13), 2020.

[17] Lov Kumar and Santanu Ku. Rath. Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept. *International Journal of System Assurance Engineering and Management*, 8:1487–1502, 2017.

[18] Issam H. Laradji, Mohammad Alshayeb, and Lahouari Ghouti. Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58:388–402, 2015.

[19] Ahmet Okutan and Olcay Taner Yıldız. Software defect prediction using bayesian networks. *Empirical Softw. Engg.*, 19(1):154–181, feb 2014.

[20] Sonali Agarwal and Divya Tomar. A feature selection based model for software defect prediction. *International journal of advanced science and technology*, 65:39–58, 2014.

[21] Kehan Gao, Taghi M. Khoshgoftaar, and Randall Wald. Combining feature selection and ensemble learning for software quality estimation. In *FLAIRS Conference*, 2014.

[22] Huanjing Wang, Taghi M. Khoshgoftaar, and Amri Napolitano. Software measurement data reduction using ensemble techniques. *Neurocomputing*, 92:124–132, 2012. Data Mining Applications and Case Study.

[23] M. Karagiannopoulos, Dionysios Anyfantis, Sotiris B. Kotsiantis, and Panayiotis E. Pintelas. Feature selection for regression problems. 2007.

[24] https://bug.inf.usi.ch/download.php.

[25] https://github.com/klainfo/NASADefectDataset/tree/master/CleanedData/MDP.

[26] Md Abdullah Al Mamun, Christian Berger, and Jörgen Hansson. Effects of measurements on correlations of software code metrics. *Empirical Software Engineering*, 24(4):2764–2818, may 2019.

[27] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374, 2016.

[28] Huanjing Wang, Taghi M. Khoshgoftaar, Kehan Gao, and Naeem Seliya. High-dimensional software engineering data and feature selection. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 83–90, 2009.

[29] Khadijah, Amazona Adorada, Panji Wisnu Wirawan, and Kabul Kurniawan. The comparison of feature selection methods in software defect prediction. In *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*, pages 1–6, 2020.

[30] Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1998.

[31] D. Asir Antony Gnana Singh, A. Escalin Fernando, and E. Jebamalar Leavline. Experimental study on feature selection methods for software fault detection. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pages 1–6, 2016.

[32] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[33] Irina Rish. An empirical study of the naive bayes classifier. 3(22):41–46, 2001.

[34] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[35] Bernd Fritzke. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7, 1994.

[36] Aris Marjuni, Teguh B. Adji, and Ridi Ferdiana. Unsupervised software defect prediction using median absolute deviation threshold based spectral classifier on signed laplacian matrix. *Journal of Big Data*, 6(1):1–20, 2019.

[37] Jaechang Nam and Sunghun Kim. *CLAMI: Defect Prediction on Unlabeled Datasets (T)*. November 2015.

[38] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

[39] Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. An empirical study of just-in-time defect prediction using cross-project models. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 172–181, New York, NY, USA, 2014. Association for Computing Machinery.

[40] Emanuel Giger, Marco D'Ambros, Martin Pinzger, and Harald C. Gall. Method-level bug prediction. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 171–180, 2012.

[41] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[42] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*. Third edition, July 2015.

[43] https://github.com/rsma-defect-prediction.

5. Appendix

Tables 5, 6, 7 and 8 show the AUC results of each algorithm applied to all the datasets. Table 9 shows the AUC values of our models fopr each dataset. Table 10 shows the p-values obtained by the techniques over each dataset.

Table 5: AUC - Naive Bayes.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.70	**0.67**	0.74	0.67	0.71
Equinox	0.65	**0.65**	0.72	0.77	0.70
Lucene	0.64	**0.63**	0.56	0.60	0.56
Mylyn	0.65	**0.62**	0.57	0.58	0.65
PDE	0.64	**0.59**	0.66	0.65	0.58
CM1	0.69	**0.66**	0.61	0.61	0.46
MW2	0.72	**0.81**	0.62	0.82	0.59
PC1	0.61	**0.70**	0.70	0.64	0.69
PC3	0.64	**0.61**	0.67	0.70	0.66
PC4	0.70	**0.78**	0.68	0.72	0.67
JM1	0.60	**0.62**	0.61	0.63	0.62
KC1	0.64	**0.59**	0.60	0.62	0.63
KC3	0.56	**0.52**	0.53	0.46	0.60
MC2	0.63	**0.72**	0.78	0.58	0.70
MC1	0.52	**0.47**	0.59	0.58	0.48
PC5	0.69	**0.68**	0.65	0.63	0.65
PC2	0.63	**0.77**	0.50	0.66	0.57

Table 6: AUC - Random Forest.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.67	0.72	**0.70**	0.67	0.72
Equinox	0.73	0.70	**0.65**	0.77	0.68
Lucene	0.52	0.56	**0.57**	0.60	0.52
Mylyn	0.57	0.58	**0.59**	0.58	0.59
PDE	0.57	0.55	**0.54**	0.65	0.53
CM1	0.50	0.49	**0.50**	0.61	0.49
MW2	0.54	0.49	**0.69**	0.82	0.54
PC1	0.61	0.59	**0.53**	0.64	0.60
PC3	0.53	0.53	**0.54**	0.70	0.55
PC4	0.65	0.66	**0.75**	0.72	0.69
JM1	0.58	0.57	**0.57**	0.63	0.56
KC1	0.64	0.62	**0.63**	0.62	0.60
KC3	0.54	0.63	**0.78**	0.46	0.62
MC2	0.57	0.61	**0.66**	0.58	0.50
MC1	0.50	0.50	**0.55**	0.56	0.50
PC5	0.65	0.66	**0.67**	0.63	0.67
PC2	0.50	0.50	**0.50**	0.66	0.50

Table 7: AUC - Neural Gas Clustering.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.57	0.52	**0.58**	0.65	0.51
Equinox	0.59	0.59	**0.62**	0.63	0.62
Lucene	0.54	0.55	**0.58**	0.54	0.62
Mylyn	0.56	0.55	**0.59**	0.57	0.49
PDE	0.59	0.59	**0.60**	0.59	0.47
CM1	0.60	0.59	**0.69**	0.61	0.52
MW2	0.72	0.71	**0.72**	0.74	0.70
PC1	0.60	0.66	**0.65**	0.66	0.47
PC3	0.51	0.49	**0.50**	0.55	0.50
PC4	0.55	0.56	**0.58**	0.58	0.71
JM1	0.53	0.55	**0.55**	0.53	0.54
KC1	0.59	0.59	**0.58**	0.59	0.56
KC3	0.62	0.62	**0.70**	0.63	0.61
MC2	0.58	0.58	**0.69**	0.67	0.54
MC1	0.62	0.54	**0.60**	0.68	0.38
PC5	0.53	0.53	**0.60**	0.54	0.59
PC2	0.53	0.53	**0.50**	0.38	0.50

Table 8: AUC - Spectral Clustering.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.71	0.50	0.70	**0.73**	0.49
Equinox	0.68	0.71	0.70	**0.71**	0.70
Lucene	0.64	0.64	0.59	**0.63**	0.57
Mylyn	0.63	0.62	0.65	**0.63**	0.55
PDE	0.66	0.66	0.67	**0.65**	0.60
CM1	0.65	0.63	0.70	**0.64**	0.48
MW2	0.71	0.70	0.70	**0.70**	0.59
PC1	0.68	0.66	0.75	**0.74**	0.54
PC3	0.70	0.66	0.50	**0.71**	0.51
PC4	0.65	0.65	0.68	**0.65**	0.65
JM1	0.64	0.64	0.64	**0.63**	0.59
KC1	0.63	0.63	0.63	**0.63**	0.63
KC3	0.61	0.61	0.69	**0.67**	0.62
MC2	0.68	0.67	0.68	**0.67**	0.63
MC1	0.67	0.68	0.73	**0.71**	0.64
PC5	0.64	0.63	0.61	**0.64**	0.40
PC2	0.74	0.75	0.70	**0.70**	0.50

Table 9: Comparison of ML techniques through AUC values.

Project	**NB - Backward**	**RF - Forward**	**NG - Backward**	**SC - Complete**
Eclipse	0.667	0.698	0.515	0.709
Equinox	0.654	0.648	0.594	0.683
Lucene	0.628	0.566	0.554	0.642
Mylyn	0.619	0.592	0.547	0.628
PDE	0.592	0.545	0.592	0.658
CM1	0.665	0.505	0.589	0.650
MW2	0.807	0.691	0.713	0.711
PC1	0.702	0.528	0.659	0.676
PC3	0.615	0.545	0.494	0.703
PC4	0.776	0.754	0.556	0.649
JM1	0.623	0.574	0.545	0.636
KC1	0.590	0.629	0.593	0.632
KC3	0.525	0.778	0.623	0.608
MC2	0.723	0.659	0.577	0.679
MC1	0.474	0.550	0.545	0.672
PC5	0.681	0.673	0.525	0.636
PC2	0.769	0.497	0.528	0.738

Table 10: Friedman test p-values over each results dataset.

Evaluation Measures	Naive Bayes	Random Forest	Neural Gas Clustering	Spectral Clustering
Accuracy	0.324	0.972	**0.000**	**0.000**
F-score	0.190	0.967	**0.000**	**0.000**
AUC	0.621	0.141	**0.006**	**0.000**
Recall	0.109	0.766	**0.000**	**0.000**
Precision	**0.000**	0.145	**0.001**	**0.002**
Specificity	0.090	**0.023**	**0.000**	**0.000**