Automatic Feature Selection by Regularization to Improve Bug Prediction Accuracy

Haidar Osman, Mohammad Ghafari, and Oscar Nierstrasz
Software Composition Group, University of Bern
Bern, Switzerland
{*osman, ghafari, oscar*}@*inf.unibe.ch*

***Abstract*—Bug prediction has been a hot research topic for the past two decades, during which different machine learning models based on a variety of software metrics have been proposed. Feature selection is a technique that removes noisy and redundant features to improve the accuracy and generalizability of a prediction model. Although feature selection is important, it adds yet another step to the process of building a bug prediction model and increases its complexity. Recent advances in machine learning introduce embedded feature selection methods that allow a prediction model to carry out feature selection automatically as part of the training process. The effect of these methods on bug prediction is unknown.**

In this paper we study regularization as an embedded feature selection method in bug prediction models. Specifically, we study the impact of three regularization methods (Ridge, Lasso, and ElasticNet) on linear and Poisson Regression as bug predictors for five open source Java systems. Our results show that the three regularization methods reduce the prediction error of the regressors and improve their stability.

***Index Terms*—Bug Prediction; Feature Selection; Machine Learning**

I. Introduction

Building a bug predictor is the process of training a machine learning model on software metrics to predict bugs in software entities. In this context, software metrics are called features, independent variables, or explanatory variables.

The quality of the trained model is directly dependent on the quality of the features. Irrelevant and correlated features degrade the performance of prediction models. The more features are fed into a model, the more complex the model is, and the less accurate the model becomes.

Feature selection is the process of selecting relevant features for model training. Feature selection reduces model complexity by eliminating noise and correlated features to reduce the generalization error. There are three main types of feature selection: filters, wrappers, and embedded methods.

Filters apply statistical measures to give scores to features independently of the machine learning model. The features are then ranked based on the score and a subset of the most relevant features is selected based on a certain score threshold. Example filters are Correlation-based Feature Selection (CFS), Information Gain (InfoGain), and Principal Component Analysis (PCA).

The 1st international Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE 2017), Klagenfurt, Austria, co-located with SANER 2017

Wrappers choose a feature subset that gives the best performance of a certain machine learning model. They try different subsets and choose the one that gives the best accuracy of the machine learning model at hand.

Embedded methods learn which features contribute to the prediction accuracy as part of the machine learning model itself. The most common type of embedded methods are regularization or penalization methods. They introduce additional terms to the optimization formula (training algorithm) of a model to penalize complex models and reduce the dimensionality of the input. Example regularization methods are Lasso, Ridge, and ElasticNet.

Each type of feature selection has its pros and cons. Filters are usually fast but less accurate than wrappers because they do not take the underlying machine learning model into account. Applying wrappers, on the other hand, gives better results in terms of accuracy but they are usually computationally intensive. Both filters and wrappers are applied as a separate step before actually training the model, while embedded methods become part of the machine learning model itself making them simpler to adopt. Although regularization methods are fast and accurate, not all machine learning models have or can adopt them.

Unfortunately, the importance of feature selection for building stable and performant prediction models is often overlooked in the bug prediction literature. A recent literature review in bug prediction reveals that 39 out of the 64 reviewed papers do not apply feature selection [1]. Also there is a handful of studies dedicated to feature selection (*e.g.*, [2][3][4][5][6][7][8]) among the vast plethora of bug prediction studies. Besides, these studies focus on filters and wrappers, while the effect of embedded feature selection remains unclear.

Performing feature selection in bug prediction is, although important, yet another step in an already complex pipeline. It requires many experiments in the trial-and-error style. This is exactly what makes regularization methods appealing and interesting. They become part of the machine learning model itself and feature selection is performed during the training phase automatically. In this paper, we study how embedded feature selection by regularization affects bug prediction accuracy. We compare linear and Poisson regressors before and after applying three regularization methods (Lasso, Ridge, and ElasticNet) on five open source Java systems. Our results reveal that the three regularization methods perform statistically similarly with a significant positive impact on the accuracy of the prediction models. The prediction error of linear and Poisson regressions are reduced by up to 16% and 50%, respectively. Based on these findings, we recommend the adoption of regularization as an easy, fast, and effective method to perform feature selection in bug prediction.

II. Technical Background

Regularization, such as Ridge, Lasso, and ElasticNet, work with any kind of regression in the same way. As an example, we explain how they work with linear regression.

Suppose the data is in a p-dimensional space. Mathematically we can represent the linear relationship as:

$$y = \beta_0 + BX \tag{1}$$

β_0 and $B = [\beta_1, \beta_2, ..., \beta_p]$ are known as model coefficients or parameters. Training a linear regression model means estimating the model parameters. These estimates are called $\hat{\beta}_0$ and $\hat{B}$.

For each input vector X_i from the training set, the response estimation is:

$$\hat{y}_i = \hat{\beta}_0 + \hat{B}X_i \tag{2}$$

And the error is:

$$error_i = y_i - \hat{y}_i = y_i - \hat{\beta}_0 - \hat{B}X_i \tag{3}$$

One of the most common methods to estimate model parameters is the *Least Squares* method, which minimizes the *Residual Sum of Squares (RSS)*, which is defined as:

$$RSS = \sum_{i=1}^{n} error_i^2 \tag{4}$$

In other words, model parameters are calculated as:

$$\{\hat{\beta}_0, \hat{B}\} = argmin(RSS) \tag{5}$$

Regularization methods add other terms, called shrinkage penalty, to the minimization Equation 5 to penalize high coefficients as follows:

$$Lasso : \{\hat{\beta}_0, \hat{B}\} = argmin(RSS + \lambda \sum_{j=1}^{p} |\beta_j|) \tag{6}$$

$$Ridge : \{\hat{\beta}_0, \hat{B}\} = argmin(RSS + \lambda \sum_{j=1}^{p} \beta_j^2) \tag{7}$$

$$ElasticNet : \{\hat{\beta}_0, \hat{B}\} = argmin(RSS+ \\ \lambda[(1-\alpha)\sum_{j=1}^{p} |\beta_j| + \alpha \sum_{j=1}^{p} \beta_j^2] : \alpha \in]0, 1[\tag{8}$$

Penalizing high coefficients leads to keeping only the ones with relevant features and shrinking the rest towards zero. In statistical terms, Lasso uses $l1$ penalty ($\sum |\beta_j|$), Ridge uses $l2$ penalty ($\sum \beta_j^2$), and ElasticNet uses a combination of both according to $\alpha \in]0, 1[$. These models have a tuning variable λ that controls the impact of the shrinkage method on the model parameter estimation.

TABLE I
THE BUG PREDICTION DATASET DETAILS, AS REPORTED BY D'AMBROS *et al.* [9]

System	Release	#Classes	% Buggy
Eclipse JDT Core	3.4	997	$\approx$ 20%
Eclipse PDE UI	3.4.1	1,497	$\approx$ 14%
Equinox	3.4	324	$\approx$ 40%
Mylyn	3.41	1,862	$\approx$ 13%
Lucene	2.4.0	691	$\approx$ 9%

TABLE II
THE CK METRICS SUITE [10] AND OTHER OBJECT-ORIENTED METRICS INCLUDED AS THE SOURCE CODE METRICS IN THE BUG PREDICTION DATASET [9]

Metric Name	Description
CBO	Coupling Between Objects
DIT	Depth of Inheritance Tree
FanIn	Number of classes that reference the class
FanOut	Number of classes referenced by the class
LCOM	Lack of Cohesion in Methods
NOC	Number Of Children
NOA	Number Of Attributes in the class
NOIA	Number Of Inherited Attributes in the class
LOC	Number of lines of code
NOM	Number Of Methods
NOIM	Number of Inherited Methods
NOPRA	Number Of PRivate Atributes
NOPRM	Number Of PRivate Methods
NOPA	Number Of Public Atributes
NOPM	Number Of Public Methods
RFC	Response For Class
WMC	Weighted Method Count

III. Empirical Study

A. The Dataset

We run the experiments on the "bug prediction dataset"[1] provided by D'Ambros *et al.* [9] to serve as a benchmark for bug prediction studies. It has been used by many bug prediction studies [12][13][14][15]. This dataset contains software metrics (source code and change metrics) on the class level for five open-source Java systems: Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, and Mylyn. A summary of the studied systems is in Table I and more details can be found in the original paper [9]. In our study, we use all the 17 source code metrics (Table II) and the 15 change metrics (Table III) in the dataset as features and the number of bugs as the response variable for building the prediction models.

B. The Machine Learning Algorithms

We use linear and Poisson regression as the machine learning models. Linear regression is a simple, effec-

[1] http://bug.inf.usi.ch/

TABLE III
THE CHANGE METRICS PROPOSED BY MOSER *et al.* [11] INCLUDED IN THE BUG PREDICTION DATASET [9]

Metric Name	Description
REVISIONS	Number of reversions
BUGFIXES	Number of bug fixes
REFACTORINGS	Number Of Refactorings
AUTHORS	Number of distinct authors that checked a file into the repository
LOC_ADDED	Sum over all revisions of the lines of code added to a file
MAX_LOC_ADDED	Maximum number of lines of code added for all revisions
AVE_LOC_ADDED	Average lines of code added per revision
LOC_DELETED	Sum over all revisions of the lines ofcode deleted from a file
MAX_LOC_DELETED	Maximum number of lines of code deleted for all revisions
AVE_LOC_DELETED	Average lines of code deleted per revision
CODECHURN	Sum of (added lines of code - deleted lines of code) over all revisions
MAX_CODECHURN	Maximum CODECHURN for all revisions
AVE_CODECHURN	Average CODECHURN for all revisions
AGE	Age of a file in weeks (counting backwards from a specific release)
WEIGHTED_AGE	Sum over age of a file in weeks times number of lines added during that week normalized by the total number of lines added to that file

tive, and widely used regression model in bug prediction [16][17][18][19][20][21][22]. Poisson Regression is also used in bug prediction [23][24]. It is called a count model because it predicts "counts", as in our case where we want the model to predict the number of bugs. For regularization, we use three methods: Ridge, Lasso, and ElasticNet [25].

C. Model Selection

For each of the studied projects, we split the dataset into two sets: training set (80%) and validation set (20%). The split maintains a similar response variable distribution between the training set and test set. We use the *CARET* package in R for this purpose.[2]

The training set is used to estimate the model parameters for the two regressors (linear and Poisson), besides the shrinkage parameter λ for the three regularization methods (Ridge, Lasso, ElasticNet) for each regressor. This estimation is done via 10-fold cross-validation on the training set only. For ElasticNet we set α to be fixed at 0.5 to give equal weights to the $l1$ and $l2$ penalization terms. We use the R package *glmnet* for model training.[3] Then, the trained models are tested on the validation set and the root mean squared error (RMSE) is calculated.

[2]https://cran.r-project.org/web/packages/caret/index.html

[3]https://cran.r-project.org/web/packages/glmnet/index.html

This whole process of splitting, training, and testing is repeated 30 times and the RMSE is averaged to avoid any bias that might result from an unfortunate data splitting.

D. Statistical Comparisons

For each project, we compare the RMSEs of the linear regressor without regularization, linear regression with Ridge, linear regression with Lasso, and linear regression with ElasticNet. Similarly, we compare RMSEs for the Poisson regression without regularization, Poisson regression with Ridge, Poisson regression with Lasso, and Poisson regression with ElasticNet.

The results are compared using the two-stage statistical test: ANOVA (Analysis of Variance) and the post-hoc test Tukey's HSD (honest significance difference), both at 95% confidence interval. ANOVA indicates whether there is a statistical difference among the populations. When the ANOVA test passes, the Tukey's HSD post-hoc test can be applied to give the results of the pairwise comparisons among the populations. If the ANOVA test does not pass, post-hoc analysis cannot be run and all populations are considered to be equivalent.

E. Results

Figure 1 summarizes the results of our experiments as boxplots. Red bold frames indicate statistically significant results, where the RMSEs of Ridge, Lasso, and ElasticNet are equivalent among each other and statistically lower than the regression model with no regularization. In the rest of the experiments, the RMSEs of Lasso, Ridge, ElasticNet, and no regularization are statistically equivalent.

As expected, regularization methods improve the accuracy of the underlying regression model. For linear regression, regularization methods decrease the root mean squared error (RMSE) for all projects, but with statistical significance only in Eclipse JDT Core. For Poisson regression, regularization methods significantly decrease the RMSE for all projects. This means that regularization affects different regressors differently, but always decreases model error. Also we notice that the dispersion of the RMSE values is less in regularized models, as indicated by the sizes of the boxes in Figure 1. This means that applying regularization not only improves the accuracy of the regressor, but also increases its stability.

Another observation is that the three feature selection algorithms perform similarly. There is no perceptible difference between the RMSEs of Ridge, Lasso, and ElasticNet, for both linear and Poisson regression. This result is important because it gives researchers and practitioners the freedom to select any regularization method they want.

F. Threats to Validity

Threats to internal validity: The quality of our results is directly dependent on the quality of the used dataset and the implementation of the used R Packages. Any error in them can introduce a systematic bias in the results.

Threats to external validity: The used dataset contains source code and change metrics from five open-source Java

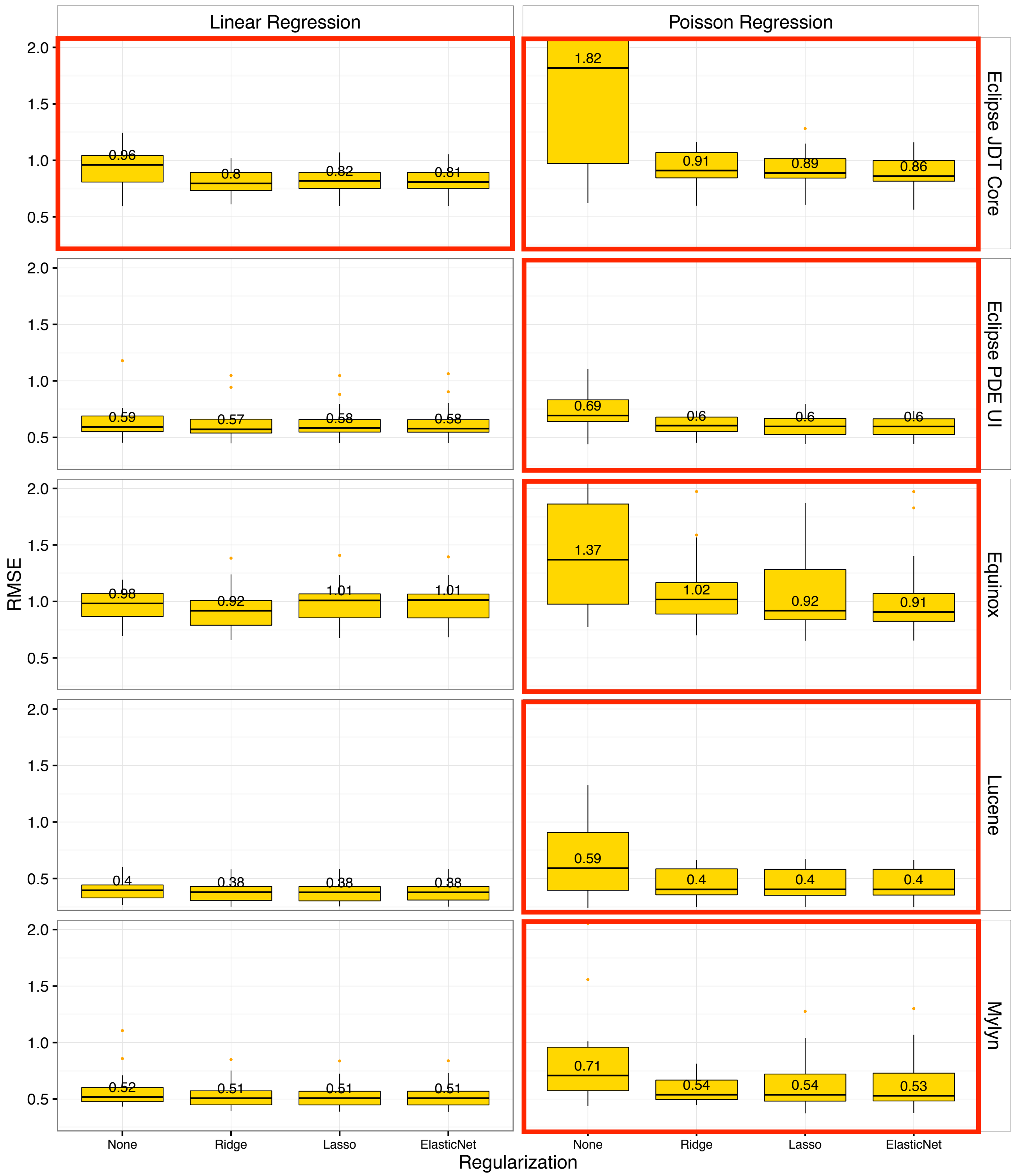

Fig. 1. Boxplots of all the experiments in our empirical study. The y-axis represents the root mean squared error (RMSE). For each project/model, we examine four regularization configurations: None, Ridge, Lasso, and ElasticNet. We carried out the analysis of variance (ANOVA) at the 95% confidence interval for each project/model to see if there is a difference between the four regularization configurations. Red bold frames indicate the statistically significant results, where the regularization methods are equivalent among each other and significantly reduce the RMSE of the base model.

systems. Our results may not generalize to industrial systems or systems written in other programming languages.

IV. Related Work

The importance of feature selection is often undermined in bug prediction studies. A recent systematic literature review in the field of bug prediction reveals that 60% of the studies do not apply feature selection at all. Among the studies that apply feature selection, filters are the most used such as correlation-based feature selection (CFS) [26][27][28][29][30][31][32], principal component analysis (PCA) [33][34][35], consistency based selection (CBS) [36], and InfoGain [37][38][39][40]. Wrapper feature selection methods are rarely applied [41][42] and regularization methods are either never used or never reported.

Few studies investigate feature selection in the field of bug prediction. Wang *et al.* [6] report that feature selection improves classification accuracy. Catal and Diri [7] explore which machine learning algorithm performs best before and after applying feature reduction. Shivaji *et al.* [2] report a significant accuracy enhancement of Naïve Bayes and Support Vector Machines classifiers when feature selection is applied. Challagulla *et al.* [3] report that correlation-based feature selection (CFS) and consistency-based subset evaluation (CBS) increase the prediction accuracy of the classifiers, while principal component analysis (PCA) before training the models does not. Gao *et al.* [4] report that classification models are either improved or remain unchanged while 85% of the original features were eliminated.

All previous studies on feature selection treat bug prediction as a classification problem, where the response variable is the class of a software entity as buggy or clean. In this study we consider bug prediction as a regression problem, where the response variable is the number of bugs in a software entity. Also, to the best of our knowledge, we are the first to study the effect of applying regularization methods on the accuracy bug prediction.

V. Conclusions and Future Work

Feature selection is a necessary step when building a bug prediction model. By reducing the number of features, it reduces model complexity, eliminates feature multicollinearity, and improves model understanding. Very little research has been dedicated to this field in general and no study has investigated regularization as an embedded feature selection in bug prediction.

In this paper, we provide an empirical evidence on the positive effect of feature selection by regularization on the performance of bug predictors. We compare the mean squared error of Poisson regression and linear regression before and after applying three regularization methods: Ridge, Lasso, and ElasticNet. We show that regularization improves the performance of both regressions and reduces the root mean squared error by up to 50%, while also increasing the stability of the prediction. Based on these findings, we recommend the adoption of regularization in regression models, when possible, as a convenient and effective technique for feature selection.

In the future, we plan to investigate the impact of more embedded methods on more machine learning models and compare embedded feature selection with filter and wrapper methods.

VI. Acknowledgments

The authors gratefully acknowledge the funding of the Swiss National Science Foundations for the project "Agile Software Analysis" (SNF project No. 200020_162352, Jan 1, 2016 - Dec. 30, 2018).[4]

The authors also gratefully acknowledge the helpful discussions and remarks of Dr. Bernadetta Tarigan.

References

[1] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[2] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, 2013.

[3] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," *International Journal on Artificial Intelligence Tools*, vol. 17, no. 02, pp. 389–400, 2008.

[4] 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.

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

[6] P. Wang, C. Jin, and S.-W. Jin, "Software defect prediction scheme based on feature selection," in *Information Science and Engineering (ISISE), 2012 International Symposium on*. IEEE, 2012, pp. 477–480.

[7] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[8] B. Turhan and A. Bener, "Analysis of naive bayes' assumptions on software fault data: An empirical study," *Data & Knowledge Engineering*, vol. 68, no. 2, pp. 278–290, 2009.

[9] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*. IEEE CS Press, 2010, pp. 31–40.

[10] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994. [Online]. Available: http://dx.doi.org/10.1109/32.295895

[11] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 181–190. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368114

[12] C. Couto, C. Silva, M. Valente, R. Bigonha, and N. Anquetil, "Uncovering causal relationships between software metrics and bugs," in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, Mar. 2012, pp. 223–232.

[13] C. Couto, P. Pires, M. T. Valente, R. S. Bigonha, and N. Anquetil, "Predicting software defects with causality tests," *Journal of Systems and Software*, vol. 93, pp. 24–41, 2014.

[14] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 182–191.

[4] http://p3.snf.ch/Project-162352

[15] K. Muthukumaran, N. Murthy, G. K. Reddy, and M. Aruna, “Comparative study on effectiveness of standard bug prediction approaches,” in *Proceedings of the 5th IBM Collaborative Academia Research Exchange Workshop*. ACM, 2013, p. 9.

[16] A. E. Hassan, “Predicting faults using the complexity of code changes,” in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 78–88. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070510

[17] M. Pinzger, N. Nagappan, and B. Murphy, “Can developer-module networks predict failures?” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT ’08/FSE-16. New York, NY, USA: ACM, 2008, pp. 2–12. [Online]. Available: http://doi.acm.org/10.1145/1453101.1453105

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

[19] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE ’06. New York, NY, USA: ACM, 2006, pp. 452–461. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134349

[20] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10

[21] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE ’08. New York, NY, USA: ACM, 2008, pp. 531–540. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368161

[22] A. Bacchelli, M. D’Ambros, and M. Lanza, “Are popular classes more defect prone?” in *Proceedings of FASE 2010 (13th International Conference on Fundamental Approaches to Software Engineering)*, 2010, pp. 59–73.

[23] K. Gao and T. M. Khoshgoftaar, “A comprehensive empirical study of count models for software fault prediction,” *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.

[24] S. S. Rathore and S. Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction,” *Soft Computing*, pp. 1–18, 2016.

[25] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[26] K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support vector machines,” *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[27] A. Kaur and R. Malhotra, “Application of random forest in predicting fault-prone classes,” in *2008 International Conference on Advanced Computer Theory and Engineering*. IEEE, 2008, pp. 37–43.

[28] E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models,” *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.055

[29] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, “A symbolic fault-prediction model based on multiobjective particle swarm optimization,” *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[30] R. Malhotra and Y. Singh, “On the applicability of machine learning techniques for object oriented software fault prediction,” *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 24–37, 2011.

[31] B. Twala, “Software faults prediction using multiple classifiers,” in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, vol. 4. IEEE, 2011, pp. 504–510.

[32] A. Okutan and O. T. Yıldız, “Software defect prediction using bayesian networks,” *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[33] T. M. Khoshgoftaar, N. Seliya, and N. Sundaresh, “An empirical study of predicting software faults with case-based reasoning,” *Software Quality Journal*, vol. 14, no. 2, pp. 85–111, 2006.

[34] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, “Object-oriented software fault prediction using neural networks,” *Information and software technology*, vol. 49, no. 5, pp. 483–492, 2007.

[35] Y. Singh, A. Kaur, and R. Malhotra, “Prediction of fault-prone software modules using statistical and machine learning methods,” *International Journal of Computer Applications*, vol. 1, no. 22, pp. 8–15, 2010.

[36] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: A large scale experiment on data vs. domain vs. process,” in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE ’09. New York, NY, USA: ACM, 2009, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595713

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

[38] B. Turhan and A. Bener, “A multivariate analysis of static code attributes for defect prediction,” in *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 2007, pp. 231–237.

[39] P. Singh and S. Verma, “An investigation of the effect of discretization on defect prediction using static measures,” in *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT’09. International Conference on*. IEEE, 2009, pp. 837–839.

[40] A. T. Mısırlı, A. B. Bener, and B. Turhan, “An industrial case study of classifier ensembles for locating software defects,” *Software Quality Journal*, vol. 19, no. 3, pp. 515–536, 2011.

[41] J. Cahill, J. M. Hogan, and R. Thomas, “Predicting fault-prone software modules with rank sum classification,” in *2013 22nd Australian Software Engineering Conference*. IEEE, 2013, pp. 211–219.

[42] I. Gondra, “Applying machine learning to software fault-proneness prediction,” *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008.