The Role of Meta-Learners in the Adaptive Selection of Classifiers

Dario Di Nucci[1,2], Andrea De Lucia[1]
[1]University of Salerno, Italy – [2]Vrije Universiteit Brussel, Belgium

***Abstract*—The use of machine learning techniques able to classify source code components in defective or not received a lot of attention by the research community in the last decades. Previous studies indicated that no machine learning classifier is capable of providing the best accuracy in any context, highlighting interesting complementarity among them. For these reasons ensemble methods, that combines several classifier models, have been proposed. Among these, it was proposed ASCI (Adaptive Selection of Classifiers in bug predIction), an adaptive method able to dynamically select among a set of machine learning classifiers the one that better predicts the bug proneness of a class based on its characteristics. In summary, ASCI experiments each classifier on the training set and then use a meta-learner (e.g., Random Forest) to select the most suitable classifier to use for each test set instance. In this work, we conduct an empirical investigation on 21 open source software systems with the aim of analyzing the performance of several classifiers used as meta-learner in combination with ASCI. The results show that the selection of the meta-learner has not strong influence in the results achieved by ASCI in the context of within-project bug prediction. Indeed, the use of lightweight classifiers such as NAIVE BAYES or LOGISTIC REGRESSION is suggested.**

***Index Terms*—Bug Prediction; Classifier Selection; Ensemble techniques**

I. INTRODUCTION

Limited time and manpower represent serious threats to the effective testing of a software system. Thus, the resources available should be allocated effectively upon the portions of the source code that are more likely to contain bugs. The creation of *bug prediction models* [1] which allow to predict the software components that are more likely to contain bugs and need to be tested more extensively is a powerful technique to deal with the allocation of testing resources.

Roughly speaking, a bug prediction model is a supervised method where a set of independent variables is used to predict the bug-proneness of a code components using a machine learning classifier (e.g., Logistic Regression [2]). The model can be trained using a sufficiently large amount of data available from the project under analysis, i.e., *within-project* strategy, or using data coming from other (similar) software projects, i.e., *cross-project* strategy. A factor that strongly influences the accuracy of bug prediction models is represented by the classifier used to predict buggy components. In details, the choice of the classifier can influence the accuracy of the predictions up to 30% [3]. Moreover, several study [4]–[6] demonstrated that the predictions of different classifiers are highly complementary despite the similar prediction accuracy.

Based on such findings, an emerging trend is the definition *ensemble* techniques [7] able to combine different models and their application to bug prediction [4]–[6], [8]–[12]. Among the *ensemble* techniques Di Nucci et al. [5] conjecture that a successful way to combine classifiers can be obtained by choosing the most suitable classifier based on the characteristics of classes, rather than combining the output of different classifiers. They proposed an adaptive prediction model, coined as **ASCI** (**A**daptive **S**election of **C**lass**I**fiers in bug prediction), which dynamically recommends the classifier able to better predict the bug-proneness of a class, based on the structural characteristics of the class. Specifically, given a set of classifiers the approach firstly trains these classifiers using the structural characteristics of the classes in the training set, secondly builds a meta-learner (e.g., Random Forest) able to predict which classifier should be used based on the structural characteristics of the classes.

In this paper, we investigated the role of the meta-learner on the accuracy of bug prediction models in the within-project context. In particular, we considered 6 alternative classifiers (e.g., DECISION TABLE (DT), DECISION TREE (C45), BINARY LOGISTIC REGRESSION (LOG), MULTI-LAYER PERCEPTRON (MLP), NAIVE BAYES (NB), and SUPPORT VECTOR MACHINE (SVM)) with respect to the original Random Forest (RF) as meta-learner. We experimented these variants of ASCI on the data of 21 software systems extracted from the PROMISE repository [13], comparing the accuracy achieved by the models. The results highlight that the selection of the meta-learner in ASCI has not a strong influence on accuracy of the prediction models in the context of within-project bug prediction. Thus the less complex alternative should be preferred.

Structure of the paper. Section II presents the related work. Section III reports the design of the empirical study, while Section IV presents the results. We discuss possible threats that could affect the validity of our empirical study in Section V, before concluding the paper in Section VI.

II. RELATED WORK

The selection the classifier to use represents a relevant problem for the configuration of bug prediction models [2]. In the past, most of the bug prediction models made use of Logistic Regression [14]–[18], Decision Trees [19]–[21], Radial Basis Function Network [22], [23], Support Vector Machines [24]–[26], Decision Tables [27], [28], Multi-Layer Perceptron [29], or Bayesian Network [30].

Despite this, among such classifiers, none of them is actually able to outperform the others [31]–[35] since their performance strongly depend on the specific dataset considered [31]–[35].

Table I
CHARACTERISTICS OF THE SOFTWARE SYSTEMS USED IN THE STUDY

#	Project	Release	Classes	KLOC	Buggy Classes	(%)
1	Ant	1.7	745	208	166	22%
2	ArcPlatform	1	234	31	27	12%
3	Camel	1.6	965	113	188	19%
4	E-Learning	1	64	3	5	8%
5	InterCafe	1	27	11	4	15%
6	Ivy	2.0	352	87	40	11%
7	jEdit	4.3	492	202	11	2%
8	KalkulatorDiety	1	27	4	6	22%
9	Nieruchomosci	1	27	4	10	37%
10	pBeans	2	51	15	10	20%
11	pdfTranslator	1	33	6	15	45%
12	Prop	6.0	660	97	66	10%
13	Redaktor	1.0	176	59	27	15%
14	Serapion	1	45	10	9	20%
15	Skarbonka	1	45	15	9	20%
16	Synapse	1.2	256	53	86	34%
17	SystemDataManagement	1	65	15	9	14%
18	TermoProjekt	1	42	8	13	31%
19	Tomcat	6	858	300	77	9%
20	Velocity	1.6	229	57	78	34%
21	Zuzel	1	39	14	13	45%

More importantly, Ghotra et al. [3] highlighted that the selection of an appropriate classifier might lead bug prediction models to be more or less effective by up to 30%, while Panichella et al. [4], Bowes et al. [6], and Di Nucci et al. [5] demonstrated the high complementarity of different classifiers.

Thus, the identification of the classifier to use is not a trivial task and for this reason a lot of effort has been devoted to the definition of *ensemble* techniques [7], i.e., methodologies able to combine different classifiers with the aim of improving bug prediction performances. Between all, recently Di Nucci et al. [5] proposed ASCI, an approach that dynamically recommends the classifier able to better predict the bug-proneness of a class based on its structural characteristics. The empirical study, conducted in the context of within-project bug prediction, showed that the approach is up to 5% more effective than VALIDATION AND VOTING [36]. In this paper, we built upon the findings reported above and provide an empirical investigation into the role of the meta-learner when using ASCI in bug prediction.

III. EMPIRICAL STUDY DEFINITION AND DESIGN

The *goal* of the empirical study is to evaluate the impact of the meta-learner selection on the performances of ASCI when adopted for within-project bug prediction. The *purpose* of the study is the better allocation of resources dedicated to testing activities. The *perspective* is of researchers interested in understanding how much the selection of the meta-learner has effect on the bug prediction capabilities of ASCI, as well as of practitioners who want to evaluate the usability of models based on this ensemble technique.

Specifically, the research questions formulated in the study is the following:

- **RQ1.** *To what extent does the selection of the meta-learner impact on the performances achieved by* ASCI *in the context of within-project bug prediction?*

A. Context Selection and Data Preprocessing

The *context* of the study was composed of the 21 software systems shown in Table I. Specifically, we considered projects having different scope (e.g., build or workflow management systems) and different size (e.g., from 3 to 300 KLOC). Table I reports the specific releases taken into account as well as the detailed characteristics of the projects considered in terms of (i) size, expressed as number of classes and KLOC, and (ii) number and percentage of buggy classes. It is worth noting that we considered two main factors when selecting the dataset. Firstly, we selected only publicly available datasets to guarantee a full replication of our experiments. Secondly, we selected software systems from various application domains and having different characteristics to reduce the threats to external validity of our study [3], [37]. Thus, we picked up a random sample of 21 systems available in the PROMISE dataset [13] and mined by Jureczko and Madeyski [38], after applying the guidelines proposed by Tantithamthavorn et al. [39] to ensure data robustness: specifically, we did not consider systems having more than 50% of buggy classes.

It is important to highlight that the considered dataset already contained both independent and dependent variables used to build the bug prediction models. More specifically, for each class of the considered systems the independent variables were represented by LOC and Chidamber and Kemerer metrics [40], while the dependent variable was represented by a boolean value indicating the bugginess of each class.

Once we selected the dataset, we applied some data preprocessing activities guided by the framework proposed by Song et al. [37] who suggested an ideal sequence of operations to perform before training a bug prediction model. In particular:

1) As shown by Shepperd et al. [41], the PROMISE repository might contain noise and/or erroneous entries that possibly bias the results of bug prediction models. To deal with this issue, they proposed a **data cleaning** procedure composed of 13 corrections aimed at increasing the data quality. We applied these steps to remove instances with conflicting values or presenting missing values, etc. From the initial dataset composed of 5,422 instances, we removed $\simeq$ 1% of instances. Thus the final dataset was composed of 5,361 instances.

2) Highly correlated independent variables can negatively affect the capabilities of bug prediction models [42]. To avoid this issue, we applied a **feature selection** algorithm, namely Correlation-based Feature Selection (CFS) [43]. This method uses correlation measures and a heuristic search strategy to identify a subset of actually relevant features for a model.

3) Bennin et al. [44] demonstrated that the problem of data unbalancing, i.e., datasets having a number of buggy classes much lower than non-buggy ones, can bias the performance of bug prediction models. For this reason, we applied a **data balancing** algorithm, namely *Synthetic Minority Over-sampling TEchnique*, i.e., SMOTE [45] to ensure a similar proportion of buggy and non-buggy classes in the training sets.

B. Baseline Selection

We compared the original version of ASCI that use RANDOM FOREST (RF) as meta-learner with 6 variants using a different classifier as meta-learner (e.g., DECISION TABLE (DT), DECISION TREE (C45), BINARY LOGISTIC REGRESSION (LOG), MULTI-LAYER PERCEPTRON (MLP), NAIVE BAYES (NB), and SUPPORT VECTOR MACHINE (SVM)).

We are aware of the possible impact of classifiers' configuration on the ability of finding bugs [46], however the identification of the ideal settings in the parameter space of a single classification technique would have been prohibitively expensive [47]. For this reason, we applied the classifiers using their default configuration.

C. Validation Strategies and Evaluation Metrics

As validation strategy, we adopted the *10-Fold Cross Validation* [48]. This methodology randomly partitions the data into 10 folds of equal size, applying a stratified sampling (e.g., each fold has the same proportion of bugs). A single fold is used as test set, while the remaining ones are used as training set. The process was repeated 10 times, using each time a different fold as test set. Then, the model performances were reported using the mean achieved over the ten runs. It is important to note that we repeated the 10-fold validation 100 times (each time with a different seed) to cope with the randomness arising from using different data splits [49].

As evaluation metrics, we avoid the computation of the widely used accuracy and F-Measure, as they are threshold-dependent metrics that can bias the interpretation of bug prediction capabilities [49]. Conversely, to properly evaluate the ability of our approach to predict the bug-proneness of classes we relied on the Matthew's Correlation Coefficient (MCC) and the Area Under the ROC Curve (AUC-ROC). The first measure indicates the extent to which the independent and dependent variables are well related to each other. The metric values range between -1 and 1 and values close to 1 indicate higher performances. As shown by Hall et al. [49], this is the most reliable threshold-independent metric for the evaluation of bug prediction models. The second measure, ranging between 0.5 and 1, reports the overall capabilities of a prediction model in discriminating buggy and non-buggy classes. A metric values close to 1 indicate higher performances. It is important to note that AUC-ROC and MCC are two complementary metrics: while MCC statistically measures the accuracy of the predictions obtained by the classifier, the AUC-ROC gives an indication on its robustness [50] (i.e., how well the classifier separates the binary classes).

As a final step of our analyses, we also statistically verified the validity of our findings. To this aim, we exploited the Scott-Knott ESD test [51]: this is an extension of the original Scott-Knott test [52] that (i) applies a hierarchical clustering algorithm to group together the performances of the cross-project bug prediction models experimented based on the statistically significance of the differences observed in terms of MCC and AUC-ROC, and (ii) refines the clusters by merging together groups whose differences are negligible in terms of effect size [53].

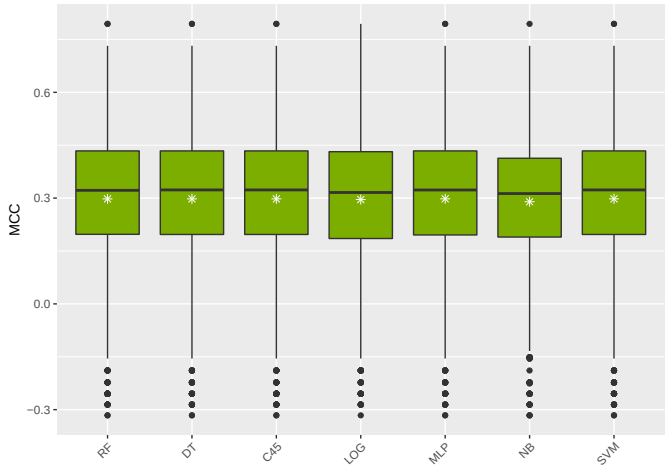

Figure 1. Boxplots of MCC achieved by the original ASCI (RF) and its variants in the within-project bug prediction context.

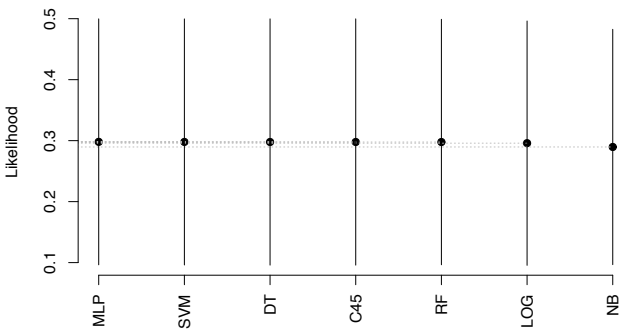

Figure 2. The likelihood of each technique in within prediction appearing in the top Scott-Knott ESD rank in terms of MCC. Circle dots indicate the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the studied datasets.

IV. ANALYSIS OF THE RESULTS

Figure 1 depicts the box plots of the MCC achieved on the 21 software systems in our dataset by the original version of ASCI (RF) and its variants in the context of within-project bug prediction (white asterisks highlight the means).

As shown, the original version of ASCI based on RANDOM FOREST (RF) has similar performances as its variants. Indeed, the median MCC achieved by this model (e.g., 32%) is the same as the ones based on DECISION TABLE, DECISION TREE, MULTI-LAYER PERCEPTRON, and SUPPORT VECTOR MACHINE. Moreover, when using simpler classifier as meta-learner such as BINARY LOGISTIC REGRESSION and NAIVE BAYES the performances drop only by 1% in term of median MCC.

Figure 2 shows the likelihood of each analyzed models to appear in the top Scott-Knott ESD rank. As expected there is no statistical significance between the considered models. Despite this, MULTI-LAYER PERCEPTRON is the classifier that acts better as meta-learner when applied to ASCI.

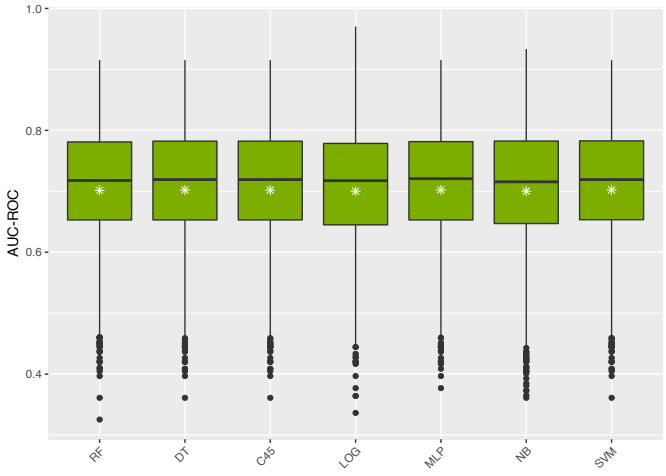

Figure 3. Boxplots of AUC-ROC achieved by the original ASCI (RF) and its variants in the within-project bug prediction contexts.

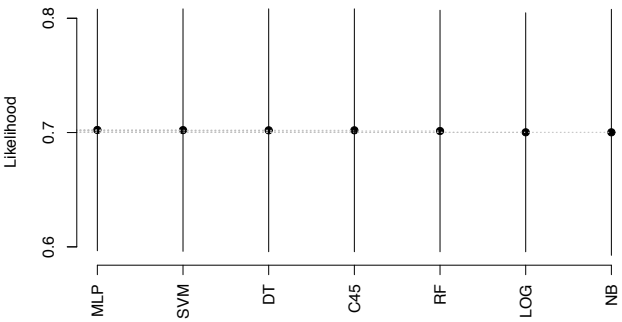

Figure 4. The likelihood of each technique in within and global/local cross prediction appearing in the top Scott-Knott ESD rank in terms of AUC-ROC. Circle dots indicate the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the studied datasets.

As an additional analysis aimed at measuring the robustness of the experimented models, we computed the Area Under the ROC Curve (AUC-ROC). Figure 3 shows the box plots representing the performances of the within-project models in terms of AUC-ROC. This analysis confirms our previous findings (achieved in terms of MCC). Indeed, the selection of meta-learner does not strongly influence the performance achieved by ASCI. Indeed, all the models achieve 70% in terms of AUC-ROC. These results are further confirmed by the Scott-Knott ESD test in Figure 4.

Summary for RQ1. In the context of within-project bug prediction, the selection of the meta-learner has not strong influence in the results achieved by an adaptive method for the dynamic selection of classifiers. The use of the MULTI-LAYER PERCEPTRON classifier as meta-learner improves the performances in a negligible way. The use of lightweight classifiers such as NAIVE BAYES or LOGISTIC REGRESSION is suggested.

V. THREATS TO VALIDITY

In this section we discuss the threats that might affect the validity of the empirical study conducted in this paper.

Threats to construct validity. Threats in this category regard the relationship between theory and observation. In our work, a threat is represented by the dataset we relied on. The dataset comes from the PROMISE repository [13], which is widely considered reliable and, indeed, has been also used in several previous work in the field of bug prediction [3], [4], [6], [8], [10], [18], [54], [55]. Although we cannot exclude possible imprecisions and/or incompleteness of the data used in the study, we applied a formal data preprocessing recommended by Shepperd et al. [41], which allowed us to reduce noise and remove erroneous entries present in the considered datasets. Moreover, it is important to note that to produce stable results we just considered software systems having less than 50% of buggy classes [39].

As for the experimented prediction models, we exploited the implementation provided by the WEKA framework [56], which is widely considered as a reliable source.

We are aware of the importance of parameter tuning for bug prediction models. To minimize this threat we used the default parameters for each classifier used in our study, since finding the best configuration for all of them would have been too expensive [47].

Threats to conclusion validity. They are related to the relation between treatment and outcome. To reduce the impact of the adopted validation methodology, we relied on the *10-Fold Cross Validation* methodology [48]. It is important to note that we repeated the 10-fold validation 100 times (each time with a different seed) to cope with the randomness arising from using different data splits [49].

To ensure that the results would have not been biased by confounding effects due to data unbalance [45] or highly correlated independent variables [57], we adopted formal procedures aimed at (i) over-sampling the training sets [45] and (ii) removing non-relevant independent variables through feature selection [43].

As for the evaluation of the performances of the experimented models, we considered AUC-ROC and MCC, which have been highly recommended by Hall et al. [49] to correctly interpret the results. We excluded, instead, other widely-used metrics such as precision, recall, and F-Measure [58] because they are threshold-dependent and possibly hide the actual performances of bug prediction models [49].

Threats to external validity. These are threats concerned with the generalizability of the findings. We analyzed 21 different software projects coming from different application domains and having different characteristics (i.e., developers, size, number of components, etc.).

Finally, it is important to note that we built models based on code metrics: as part of our future research agenda, we aim at analyzing the impact of process- (e.g., the entropy of changes proposed by [59]) and developer-related (e.g., the number of developers working on a code component [16] [29]) metrics on our findings.

VI. Conclusion

In this paper, we aimed at evaluating the role of the meta-learner selection in ASCI, an adaptive method for the dynamic selection of classifiers, in the context of within-project bug prediction. Specifically, we compared the performances of the original version of ASCI using RANDOM FOREST as meta-learner with six variants using different classifiers. The case study has been conducted on a set of 21 software projects from the PROMISE dataset.

We found that the meta-learner selected when using ASCI has not a strong impact on the prediction performance in the context of within-project bug prediction. Using MULTI-LAYER PERCEPTRON classifier as meta-learner increases the performances only in a negligible way. Hence, lightweight classifiers such as NAIVE BAYES or LOGISTIC REGRESSION should be preferred as meta-learner for ASCI with respect to RANDOM FOREST.

As future work, we plan to replicate the study in a different context such as cross-project bug prediction. Moreover, we plan to study (i) the role of the meta-learners in other ensemble techniques such as CODEP [4] and (ii) the impact of the ensemble techniques relying on a single classifier such as BOOSTING and BAGGING [7] on the its performance.

References

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

[2] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[3] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the International Conference on Software Engineering*. IEEE, 2015, pp. 789–800.

[4] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering*. IEEE, 2014, pp. 164–173.

[5] D. Di Nucci, F. Palomba, R. Oliveto, and A. De Lucia, "Dynamic selection of classifiers in bug prediction: An adaptive method," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 202–212, 2017.

[6] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?" *Software Quality Journal*, pp. 1–28, 2017.

[7] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 1–39, 2010.

[8] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 852–864, 2010.

[9] 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.

[10] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *Journal of Information & Computational Science*, vol. 8, no. 16, pp. 4241–4254, 2011.

[11] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of International Conference on Software Engineering*. IEEE, 2011, pp. 481–490.

[12] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[13] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. (2012, June) The promise repository of empirical software engineering data. [Online]. Available: http://promisedata.googlecode.com

[14] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[15] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 580–586.

[16] R. Bell, T. Ostrand, and E. Weyuker, "The limited impact of individual developer data on software defect prediction," *Empirical Software Engineering*, vol. 18, no. 3, pp. 478–505, 2013.

[17] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[18] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto, "Toward a smell-aware bug prediction model," *IEEE Transactions on Software Engineering*, 2017.

[19] R. Bell, T. Ostrand, and E. Weyuker, "Does measuring code change improve fault prediction?" in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 2011, pp. 2:1–2:8.

[20] J. S. M. Todd L. Graves, Alan F. Karr and H. P. Siy, "Predicting fault incidence using software change history," *Software Engineering, IEEE Transactions on*, vol. 26, no. 7, pp. 653–661, 2000.

[21] R. Moser, W. Pedrycz, and G. Succi, "Analysis of the reliability of a subset of change metrics for defect prediction," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2008, pp. 309–311.

[22] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *ACM Conference on Computer and Communications Security (CCS)*, ser. CCS '07, 2007, pp. 529–540.

[23] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for Eclipse," in *Proceedings of 3rd ICSE International Workshop on Predictor Models in Software Engineering*. IEEE Computer Society, 2007.

[24] A. P. Nikora and J. C. Munson, "Developing fault predictors for evolving software systems," in *Proceedings of the 9th IEEE International Symposium on Software Metrics*. IEEE CS Press, 2003, pp. 338–349.

[25] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *Proceedings of the International Conference on Software Engineering*. IEEE, 2007, pp. 489–498.

[26] T. Wolf, A. Schroter, D. Damian, and T. H. D. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 1–11.

[27] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, 2008.

[28] T. M. Khoshgoftaar, N. Goel, A. Nandi, and J. McMullan, "Detection of software modules with high debug code churn in a very large legacy system," in *Software Reliability Engineering*. IEEE, 1996, pp. 364–371.

[29] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5–24, 2017.

[30] 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*, 2008, pp. 1–12.

[31] M. E. Bezerra, A. L. Oliveira, P. J. Adeodato, and S. R. Meira, *Enhancing RBF-DDA algorithm's robustness: Neural networks applied to prediction of fault-prone software modules*. Springer, 2008, pp. 119–128.

[32] M. E. Bezerra, A. L. Oliveira, and S. R. Meira, "A constructive rbf neural network for estimating the probability of defects in software modules," in *2007 International Joint Conference on Neural Networks*. IEEE, 2007, pp. 2869–2874.

[33] M. O. Elish, "A comparative study of fault density prediction in aspect-oriented systems using mlp, rbf, knn, rt, denfis and svr models," *Artificial Intelligence Review*, vol. 42, no. 4, pp. 695–703, 2014.

[34] G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 675–686, 2007.

[35] R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with android software," *Applied Soft Computing*, vol. 49, pp. 1034–1050, 2016.

[36] A. Tosun, B. Turhan, and A. Bener, "Ensemble of software defect predictors: a case study," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 318–320.

[37] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, 2011.

[38] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868342

[39] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 321–332.

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

[41] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *Software Engineering, IEEE Transactions on*, vol. 39, no. 9, pp. 1208–1215, Sept 2013.

[42] R. M. O'brien, "A caution regarding rules of thumb for variance inflation factors," *Quality & Quantity*, vol. 41, no. 5, pp. 673–690, 2007.

[43] M. A. Hall, "Correlation-based feature selection for machine learning," Tech. Rep., 1998.

[44] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[45] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[46] S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan, "The impact of classifier configuration and classifier combination on bug localization," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1427–1443, 2013.

[47] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, pp. 281–305, 2012.

[48] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society. Series B (Methodological)*, pp. 111–147, 1974.

[49] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "Developing fault-prediction models: What the research can show industry," *IEEE Software*, vol. 28, no. 6, pp. 96–99, 2011.

[50] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[51] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transanctions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2017.

[52] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, vol. 30, pp. 507–512, 1974.

[53] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[54] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Proceedings of the IEEE Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 264–269.

[55] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[56] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations Newsletter.*, vol. 11, no. 1, 2009.

[57] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM Computing Surveys*, vol. 27, no. 3, pp. 326–327, Sep. 1995.

[58] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[59] A. E. Hassan, "Predicting faults using the complexity of code changes," in *ICSE*. Vancouver, Canada: IEEE Press, 2009, pp. 78–88.