Adaptive Selection of Classifiers for Bug Prediction: A Large-Scale Empirical Analysis of Its Performances and a Benchmark Study

Fabiano Pecorelli

SeSa Lab, University of Salerno, Italy — fpecorelli@unisa.it

Dario Di Nucci

Jheronimus Academy of Data Science, Tilburg University, The Netherlands — d.dinucci@uvt.nl

Abstract

Bug prediction aims at locating defective source code components relying on machine learning models. Although some previous work showed that selecting the machine-learning classifier is crucial, the results are contrasting. Therefore, several ensemble techniques, *i.e.,* approaches able to mix the output of different classifiers, have been proposed. In this paper, we present a benchmark study in which we compare the performance of seven ensemble techniques on 21 open-source software projects. Our aim is twofold. On the one hand, we aim at bridging the limitations of previous empirical studies that compared the accuracy of ensemble approaches in bug prediction. On the other hand, our goal is to verify how ensemble techniques perform in different settings such as cross- and local-project defect prediction. Our empirical experimentation results show that ensemble techniques are not a silver bullet for bug prediction. In within-project bug prediction, using ensemble techniques improves the prediction performance with respect to the best stand-alone classifier. We confirm that the models based on Validation and Voting achieve slightly better results. However, they are similar to those obtained by other ensemble techniques. Identifying buggy classes using external sources of information is still an open problem. In this setting, the use of ensemble techniques does not provide evident benefits with respect to stand-alone classifiers. The statistical analysis highlights that local and global models are mostly equivalent in terms of performance. Only one ensemble technique (*i.e.,* ASCI) slightly exploits local learning to improve performance.

Keywords: Bug Prediction, Within-project, Cross-project, Ensemble Classifiers.

1. Introduction

Bug prediction is the software engineering field that aims at locating potentially defective source code components in a software system [33] with the aim of prioritizing code review or testing activities [72, 73]. Researchers have been proposing various implementations of bug prediction, including the use of unsupervised approaches based on (i) topic modeling [66], (ii) self-organizing maps [20], and (iii) connectivity metrics [98]. Yet, the most widely adopted solution is represented by the adoption of supervised learning, which consists of the definition of machine learning-based models in which a set of features (also called predictors) are exploited to predict the defect-proneness of source code classes [33]. These models can be trained using three different strategies: (1) using a sufficient amount of labeled data coming from previous versions of the same system where the model should be applied to, *i.e., within-project* training [90], (2) using labeled data from similar projects, *i.e., cross-project* training [100], or (3) clustering homogeneous data from different projects to reduce variability and then building a model for each cluster, *i.e., local cross-project* training [58].

The performance of bug prediction models can be influenced by a number of factors, *e.g.,* training strategy [90], data balancing [84], or feature selection [95]. However, one of the aspects more often correlated to the ability of machine learning models to predict bugs is represented by the selection of the classifier exploited to discriminate buggy and non-buggy classes (*e.g.,* Random Forest). For instance, Ghotra et al. [28] showed that a sub-optimal selection can decrease the performance of bug prediction models up to 30%. Furthermore, a previous study [71] found that different classifiers have similar performance, yet being able to correctly predict the defect-proneness of different classes. These findings motivated the definition of alternative approaches able to exploit the complementarity of classifiers: these are known as *ensemble* techniques.

An example is given by the Validation and Voting strategy proposed by Tosun *et al* [89]: this is a two-stage approach that consists of (1) building a number of bug prediction models relying on different classifiers and (2) predicting the defectiveness of classes based on what the majority of the models built suggest. Alternative techniques exploit meta-algorithms that aim at reducing the variance of the training data to avoid model overfitting [47]

or introduce a meta-learner that can combine the outcome of individual classifiers [71].

In our recent research [24], we also contributed to the definition of ensemble techniques for bug prediction. In particular, we proposed ASCI (which is the acronym of Adaptive Selection of Classifiers in bug predIction) [24], a novel approach that is able to dynamically select, through a classification model, the machine learner to be used to predict the defect-proneness of a class based on its characteristics. The performance of the approach has been originally assessed in a within-project scenario, where we found that its usage provides better results than five individual classifiers and the VALIDATION AND VOTING technique. Later on, we also preliminarily investigated the capabilities of ASCI when applied to cross-project bug prediction [22]: also, in this case, our technique performed better than VALIDATION AND VOTING on ten software projects.

In this paper, our goal is to make a further step ahead toward understanding the value of dynamic selection of classifiers for bug prediction. We propose a large-scale empirical investigation whose aim is twofold. On the one hand, we **corroborate** the previous findings obtained by using ASCI in the context of within- and cross-project bug prediction, in an effort of verifying the generalizability of our technique when considering larger and more diverse datasets. On the other hand, we **benchmark** the performance of ASCI with a large variety of six alternative ensemble techniques. Such an analysis allows us to challenge the findings provided by Liu et al. [53] and Zhang et al. [99], who previously studied the performance of ensemble techniques for bug prediction. In particular, Liu et al. [53] showed that the usage of the VALIDATION AND VOTING technique allows bug prediction models to work better, while Zhang et al. [99] benchmarked seven ensemble techniques, belonging to four categories, confirming the findings on the superiority of the VALIDATION AND VOTING technique. While the findings by Liu et al. [53] and Zhang et al. [99] represent an important source of information for researchers and practitioners interested in the application of ensemble techniques in practice, in the context of our research we figured out six critical limitations that might have possibly threatened the conclusions provided so far. More specifically:

Data quality. Both Liu et al. [53] and Zhang et al. [99] exploited projects coming from the SEACRAFT repository [59]. Unfortunately, the findings by Shepperd et al. [81] reported that these datasets might contain noisy and/or erroneous entries that may possibly bias the interpretation of the results; as such, a correction procedure should be applied before training prediction models.

Data preprocessing. As widely demonstrated in the literature [29, 32, 45, 82], data preprocessing techniques such as (i) data normalization, (ii) feature selection, and (iii) training data balancing should always be applied before running bug prediction models to limit conclusion instability. However, Liu et al. [53] and Zhang et al. [99] did not perform a complete data preprocessing before building the experimented bug prediction models: as a consequence, their findings might have under-/over-estimated the capabilities of ensemble techniques.

Data Analysis. To evaluate the performance of bug prediction models, previous work heavily relied on metrics such as precision, recall, and F-Measure [2]. Nevertheless, these are threshold-dependent metrics that should be complemented with additional measures (*e.g.,* AUC-ROC) to have a clearer interpretation of the actual effectiveness of bug prediction models [32].

Limited size of previous analysis. The study by Liu et al. [53] has been conducted on seven systems and considering the behavior of two ensemble techniques, *i.e.,* BAGGING and VALIDATION AND VOTING, while Zhang et al. [99] took into account a dataset composed of ten systems analyzing seven ensemble approaches classified in four categories, *i.e.,* RANDOM FOREST, BAGGING, BOOSTING, and VALIDATION AND VOTING. As a consequence, a wider analysis that includes the recent ensemble techniques proposed in the literature may be beneficial.

Unclear relationship between local learning and ensemble classifiers. While the potential usefulness of *local* learning on the performance of bug prediction models has been shown [58], to the best of our knowledge there has been no attempt to investigate the extent to which such local learning strategy can benefit the usage of ensemble techniques.

Missing comparison with within-project prediction models. One of the key promises of cross-project bug prediction models is to be competitive with respect to within-project ones. As recommended in previous work [36, 92], whenever a certain technique is experimented in a cross-project setting, it should be applied in a within-project setting as well in order to fairly benchmarking its real capabilities. The studies by Liu et al. [53] and Zhang et al. [99] did not test how ensemble approaches applied in cross-project bug prediction work when compared with their adoption in a within-project scenario.

Our benchmark experiment copes with the aforementioned issues. Specifically, we consider 21 datasets coming from SEACRAFT [59] and apply a number of corrections suggested by Shepperd et al. [81] in order to make them cleaned and suitable for our purpose. Then, we take into account several different ensemble techniques, belonging to six categories, measuring their performance using the two metrics recommended by previous work, *i.e.,* AUC-ROC and Matthew's Correlation Coefficient [32, 85].

The key results of our study suggest that the problem of identifying buggy classes using external sources of information is still far from being solved. The use of ensemble

techniques does not provide evident benefits with respect to stand-alone classifiers, but in general, the VALIDATION AND VOTING [89] and ASCI [24] techniques should be preferred among the others. These observations also hold when applying ensemble methodologies to local bug prediction; moreover, we found that global and local models are mostly statistically equivalent. Finally, ensemble-based cross-project models perform worse than within-project ones in terms of prediction accuracy, yet being more robust (their performance does not vary as much as the ones of within-projects models). Furthermore, we observed that also in a within-project scenario the use of ensemble classifiers does not guarantee better prediction performances with respect to models adopting stand-alone machine learners.

Structure of the paper. Section 2 discusses the previous achievements attained by the research community. In Section 3 we describe the methodology followed to conduct the empirical study. Section 4 analyzes the obtained results, while Section 5 discusses them. Section 6 reports possible threats affecting our findings and how we mitigated them. Finally, Section 7 concludes the paper and describes our future research agenda on the topic.

2. Related Work

The empirical study we proposed is mainly concerned with (i) how to effectively train a bug prediction model, *i.e.,* how to treat training data effectively, and (ii) the machine learning technique to exploit to create a cross-project bug prediction model. The following subsections cover the literature related to these two aspects.

2.1. Training Cross-Project Bug Prediction Models

Cross-project bug prediction models are based on the usage of data coming from external (similar) projects to train a machine learner able to discriminate buggy and non-buggy instances in the project currently being analyzed [100]. While most of the research made in this area investigated *which* are the most efficient features to use in cross-project models to correctly capture the bugginess of software classes [3, 5, 12, 19, 23, 35, 54, 70], a notable effort has been also devoted to *how* to make external data suitable for the project under analysis [15, 55, 63, 92]. The latter problem aims at dealing with the fact that cross-project models suffer data heterogeneity, *i.e.,* external data might be different with respect to the one available in the project to analyze, leading to worsening the performance of bug prediction models. To this aim, the research community proposed a number of approaches, classified in (i) data normalization (ii) data filtering, and (iii) data clustering.

Data normalization approaches directly act on the metrics transferred from external projects [97]. In particular, since software metrics are usually not normally distributed, they need to be transformed before building a bug prediction model [97]. Watanabe et al. [92] proposed the use of data standardization based on the mean value of the corresponding feature in the target project, while Ma et al. [55] exploited the concept of data gravitation in order to weight external data on the basis of the similarity with the target project. Nam et al. [63] compared how the performances of bug prediction models vary when data are normalized using a min-max scaling or a Z-score approach, finding the former as the best technique for data normalization. Herbold et al. [39] performed a wide replication study, comparing 24 approaches devised for cross-project bug prediction. They recommended some methods that should be applied to have better performance such as the data normalization proposed by Camargo Cruz and Ochimizu [15]. Moreover, their results indicate that cross-project bug prediction is not yet ready for being applied in practice.

Data filtering techniques aim at selecting the training data that is more similar to the target project. Most of the approaches in this category rely on the use of clustering algorithms, and more specifically of the *k-Nearest Neighbor* one [37, 38, 90].

Finally, data clustering techniques aim at grouping together instances of the training set in order to build a separate bug prediction model for each cluster [58]: for this reason, they are called *local learning* methods. Unlike data filtering techniques, approaches in this set do not have the goal to filter training data but rather that of building a specialized bug prediction model for each cluster of the training data. Menzies et al. [58] found that the application of local learning can substantially improve the performance of bug prediction models with respect to global models. Later on, Bettenburg et al. [9] discovered that the performance of local models is strongly impacted by the tuning of the clustering algorithm exploited: as a consequence, they conclude that the usage of clustering algorithms able to automatically identify the ideal number of clusters to create is desirable. Finally, Scanniello et al. [80] tested how the performance of local bug prediction vary when grouping external classes based on dependency analysis rather than external features. While the approach proposed by Scanniello et al. [80] improved the performance of the original local bug prediction model proposed by Menzies et al. [58], the intensive analyses required to extract dependencies make it less scalable. Despite their promising performance, Herbold et al. [40] showed that local models are not significantly different from global models in terms of overall prediction performance.

The empirical study conducted in this paper can be seen as complementary to those reported above. Indeed, our main goal is to investigate whether ensemble techniques actually provide benefits in the contexts of global and local cross-project bug prediction. Moreover, we took into account previous findings concerned with the optimal setup of training data and performed a number of preprocessing steps before building bug prediction models.

2.2. Machine Learning Techniques in Bug Prediction

Selecting the classifier to use represents a relevant problem for the configuration of bug prediction models [1]. In the past, most of the bug prediction models devised made use of Logistic Regression [5, 19, 61, 62, 70], Decision Trees [4, 60, 88], Radial Basis Function Network [65, 101], Support Vector Machines [48, 67, 93], Decision Tables [46, 57], Multi-Layer Perceptron [23], or Bayesian Network [76].

Among such classifiers, none of them is actually able to outperform the others [11, 10, 26, 69, 56] since their performance strongly depends on the specific dataset considered. More importantly, Ghotra et al. [28] 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. [71] demonstrated that the predictions of different classifiers are highly complementary despite the similar prediction accuracy.

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 so-called *ensemble* techniques, *i.e.,* methodologies able to combine different classifiers with the aim of improving bug prediction performance. Tosun et al. [89] devised the VALIDATION AND VOTING technique, that is a method to combine the output of different classifiers using an aggregating function. More specifically, the technique predicts a class as buggy in case the majority of models (obtained running different classifiers on the same training set) predicts the bugginess of a class; otherwise, the class is predicted as bug-free.

Other techniques proposed in the literature are based on the BAGGING ensemble technique [78], which combines the outputs of different models trained on a sample of instances taken with a replacement from the training set. For instance, Kim et al. [47] combined multiple training data obtained applying a random sampling. More recently, some approaches inspired to the Stacking ensemble technique Rokach [78] have been proposed [71, 75]. They use a meta-learner to induce which classifiers are reliable and which are not and consider the predictions of different classifiers as input for a new classifier.

Specifically, Panichella et al. [71] devised CODEP, an approach that first applies a set of classifiers independently, and then uses the output of the first step as predictors of a new prediction model based on Logistic Regression. Zhang et al. [99] conducted a similar study like the one performed in [71], comparing different ensemble approaches. They found that there exist several ensemble techniques that improve the performance achieved by CODEP, and Validation and Voting are often one of them. Petrić et al. [75] used four families of classifiers in order to build a Stacking ensemble technique [78] based on the diversity among classifiers in the cross-project context. Their empirical study showed that their approach can perform better than other ensemble techniques and that the diversity among classifiers is an essential factor. Furthermore, Wang et al. [91] compared the performance achieved by seven ensemble techniques, each of them belonging to a different category, in the context of within-project bug prediction, showing that often VALIDATION AND VOTING stands out among them.

Finally, Di Nucci et al. [24] proposed ASCI, an approach that dynamically recommends the classifier able to better predict the bug-proneness of a class based on its structural characteristics (*i.e.,* product metrics). 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. Later on, Di Nucci et al. [22] conducted a preliminary study on the performance of ASCI when applied to cross-project bug prediction, confirming that this technique can outperform the VALIDATION AND VOTING one.

With respect to the papers discussed above, the empirical study proposed herein has the goal of further understanding the performance of ASCI as well as of ensemble techniques when applied to bug prediction. More specifically, we took into account a larger variety of software projects than previous work [22, 24, 91, 99], preprocessed data following quality standards [81, 82], and experimented with most of the state-of-the-art ensemble techniques defined so far in order to provide a clearer, updated view of the capabilities of ensemble techniques. Furthermore, we also analyzed the impact of local learning and its combination with ensemble methods on the performance of bug prediction models.

3. Empirical Study Definition and Design

The *goal* of the empirical study is twofold: in the first place, we aim at enriching the investigation of the capabilities of ASCI, an ensemble method we proposed in a previous work [24], by experimenting it with a larger set of software projects trained in both within- and cross-project scenarios. Secondly, we aim at benchmarking and comparing the capabilities of ASCI with those of six alternative approaches, hence providing a clearer overview of the performance of ensemble bug prediction. The *purpose* of the study is a better allocation of resources dedicated to testing activities. The *perspective* is of researchers interested in understanding how much the selection of an ensemble technique has an effect on bug prediction capabilities, as well as of practitioners who want to evaluate the usability of ensemble-based bug prediction models.

Specifically, our study is driven by the following research questions (**RQ**s):

RQ$_1$ *What is the performance of* ASCI *in a within-project scenario when compared to alternative state-of-the-art ensemble techniques?*

RQ$_2$ *What is the performance of* ASCI *in a cross-project scenario when compared to alternative state-of-the-art ensemble techniques?*

RQ$_3$ *What is the performance of* ASCI *in a local cross-project scenario when compared to alternative state-of-the-art ensemble techniques?*

The first research question (**RQ$_1$**) has the goal of replicating our previous study [24] by considering a wider set of both projects and baselines in the context of within-project bug prediction, while **RQ$_2$** aims at providing a clearer view of how our technique works when applied in a cross-project scenario, also when compared to alternative ensemble methods. Finally, **RQ$_3$** focuses on the role of local learning, namely on the combination between local bug prediction and the use of ensemble methodologies.

3.1. Context of the Study

The *context* of the study was composed of 21 software systems, collected by Jureczko and Madeyski [44] and available in the SEACRAFT repository [59], whose details are shown in Table 1. 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 1 reports details about the specific releases taken into account, the size (in terms of number of classes and KLOC), and the number and percentage of buggy classes. To properly select the dataset we considered two main factors. Firstly, we considered 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 the external validity of our study [28, 82]. Thus, we picked up 21 systems collected by Jureczko and Madeyski [44] and available in the SEACRAFT repository [59], after applying the guidelines proposed by Tantithamthavorn et al. [85] to ensure data robustness: specifically, we did not consider systems having more than 50% of buggy classes.

It is important to highlight that the dependent, as well as the independent variables used in our study, were already contained in the dataset. In particular, we exploited structural metrics, represented by LOC and Chidamber and Kemerer metrics [18], to predict the bugginess of each class, represented by a boolean value.

After we selected the dataset, we performed some data preprocessing steps as reported below:

1. **Data Cleaning**. To remove possible noise or erroneous entries, we applied the data cleaning procedure proposed by Shepperd et al. [81], which is composed of 13 steps needed to fix the NASA dataset for software defect prediction. However, not all the steps apply to our datasets where we had only to remove (i) constant features and (ii) features with missing values. The cleaning involved 61 instances (*i.e.,* $\simeq$ 1%) of the 5,422 in the initial dataset.

2. **Data Normalization**. The performance of prediction models can also be affected by the different levels of design-complexity metrics [15, 63]. Based on the results provided by Nam et al. [63] and Herbold et al. [40], we applied a linear normalization in the [0,1] interval, relying on the normalization filter implemented in WEKA [30].

3. **Feature Selection**. Another aspect that can negatively affect the performance of bug prediction models is the high correlation between independent variables [68]. To deal with this issue, we relied on the Correlation-based Feature Selection (CFS) approach [31], which allows identifying a subset of actually relevant features for a model. It is worth noting that we applied CFS only after we combined the instances belonging to different software systems, as recommended by Hall et al. [33].

4. **Data Balancing**. Usually, bug prediction is an unbalanced problem, *i.e.,* the number of buggy classes in a system is much lower than non-buggy ones. Since this aspect can bias the performance of our prediction model [6], we applied the *Synthetic Minority Over-sampling TEchnique*, *i.e.,* SMOTE [17] to ensure training sets having a similar proportion of buggy and non-buggy classes.

It is important to note that the order of the preprocessing steps has been guided by the framework proposed by Song et al. [82], who suggested an ideal sequence of operations to perform before training a bug prediction model. The final preprocessed datasets are available in our online appendix [74].

3.2. Baseline Selection

The performances achieved by our approach were firstly compared with those obtained by the model relying on the NAÏVE BAYES classifier, which was found to be the best stand-alone machine learner over our dataset. More specifically, we ran seven stand-alone classifiers, *i.e.,* MULTI-LAYER PERCEPTRON, NAÏVE BAYES, LOGISTIC REGRESSION, RADIAL BASIS FUNCTION, C4.5, DECISION TABLE, and SUPPORT VECTOR MACHINE on the same set of systems considered in the study and using the same validation methodology. As a result, we found that the use of NAÏVE BAYES led to the best results in terms of MCC. For this reason, we considered such a classifier as our baseline. A complete overview of the results achieved by the stand-alone classifiers is available in our online appendix [74].

In the second place, we benchmarked ASCI with a set of ensemble techniques that were previously experimented for bug prediction. Specifically:

- **Boosting**. The BOOSTING technique iteratively uses a set of models built in previous iterations to manipulate the training set [78]. At the next iteration, the model focuses on those instances more difficult to predict. ADAPTIVE BOOSTING (ADABOOST) [27] is a well-known BOOSTING technique. During

Table 1: 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%

the training phase, AdaBoost repetitively trains a *weak* classifier on subsequent training data. At each iteration, a weight is assigned to each instance of the training set, with the purpose of assigning higher weights to misclassified instances that should have more chances to be correctly predicted by the new models. At the end of the training phase, a weight is assigned to each model in order to reward models having higher overall accuracy. During the test phase, the prediction of a new instance is performed by voting of all models. The results are thus combined using the weights of the models, in the case of binary classification a threshold of 0.5 is applied. We used the default configuration provided by the Weka toolkit [30], implementing NBBoosting, a model relying on Adaptive Boosting and using Naïve Bayes (NB) as the *weak* learner. We used NB because, in the considered context, this classifier achieved better performance with respect to other classifiers.

- **Bootstrap Aggregating (Bagging)**. Bagging [14] combines the output of various models in a single prediction. During the training phase, m datasets with the same size as the original one are generated by performing sampling with replacement (Bootstrap) from the training set. Hence for each dataset, a model is trained using a *weak* classifier. During the test phase, for each instance, the composite classifier uses a majority voting rule to combine the output of the models into a single prediction. We used the default configuration provided by the Weka toolkit [30], implementing NBBagging, a model that use the same *weak* learner used for Boosting (*i.e.,* NB).

- **Validation and Voting**. Validation and Voting [49] (also called Voting) is a weighting method. It combines the confidence scores obtained by the underlying classifiers. For each instance to predict, each classifier returns a confidence score ranging between 0 and 1. The scores are combined by an operator (*e.g.,* Average in case of AvgVoting and Maximum in case of MaxVoting). A class is marked as buggy if the combination of the confidence scores is higher than 0.5, while it is predicted as clean otherwise. We configured Voting as already done in previous work [71] using Logistic Regression, Radial Basis Function, C4.5, Decision Table, Multi-Layer Perceptron, Naïve Bayes, and Support Vector Machine.

- **Adaptive Selection of ClassIfiers** [24]. This algorithm has been developed specifically for bug prediction purposes. The basic idea is to dynamically choose the classifier for each test set instance relying on its structural characteristics. During the training phase, ASCI trains each of the base classifiers and collects the evaluations on the training set. Lately, a decision tree is built: the internal nodes represent the structural characteristics of the classes contained in the training set, while the leaves represent the classifiers able to correctly classify the bug-proneness of instances having such structural characteristics. During the test phase, ASCI firstly predicts for each instance the most suitable model and then uses it to predict the bugginess of this instance. In this study, we configured the model by using the same set of

classifiers used for the VALIDATION AND VOTING ensemble methods.

- **CODEP.** CODEP [71] is a technique based on STACKING [94] that uses a meta-classifier (*e.g.,* LOGISTIC REGRESSION) to infer the bugginess of classes [78]. During the training phase, CODEP trains each of the base classifiers and creates a new dataset by collecting the confidence scores assigned by the classifiers on each training instance. Finally, with the aim of combining the outputs of the base classifiers, a meta-classifier is built on such a new dataset. In our study, we configured the model by adding SUPPORT VECTOR MACHINE, a popular machine learner, to the set of base classifiers previously used by Panichella et al. [71] and used also for the VALIDATION AND VOTING ensemble methods. As done in [71], we used LOGISTIC REGRESSION as meta-classifier.

- **Random Forest**. RANDOM FOREST [42] is an ensemble of pruned decision trees. As showed for BAGGING, each decision tree is built by using BOOTSTRAP. The combination of the prediction of the decision trees is performed by using majority voting. In our experiments, we used the default configuration provided by the WEKA toolkit [30].

We are aware of the possible impact of classifiers' configuration on the ability of finding bugs [87], however, the identification of the ideal settings in the parameter space of a single classification technique would have been prohibitively expensive [7]. For this reason, we applied the classifiers using their default configuration.

3.3. Validation Strategies and Evaluation Metrics

A key decision in this context was the selection of an appropriate validation strategy [86].

In the context of *within-project* bug prediction ($\mathbf{RQ}_1$), we adopted the *10-Fold Cross Validation* [83]. 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 performance was 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 [32].

In the contexts of *cross-project* bug prediction ($\mathbf{RQ}_2$), we could not adopt the *10-Fold Cross Validation* as validation strategy, as we could not use as test set data coming from the same system as the training set. Thus, we opted for the *Leave-One-Out Cross-Validation* [79]. In this strategy, the model is trained using the data of all the systems but one, which is retained as *test set*. The cross-validation has been then repeated 21 times, allowing each of the 21 systems to be the test set exactly once [79]. We used this validation strategy since it is among the least biased and most stable validation approaches, according to the findings reported by Tantithamthavorn et al. [86]. Furthermore, it is important to note that this strategy (i) allows all systems to be used for both training and test purpose, and (ii) has been widely used in the context of bug prediction [16, 71, 90, 99].

In the context of $\mathbf{RQ}_3$, we had to build local bug prediction models. To this aim, we exploited the EXPECTATION MAXIMIZATION (EM) clustering algorithm proposed by Dempster et al. [21]. The choice of this algorithm was driven by multiple factors. Firstly, it can automatically determine the number of clusters through an internal cross-validation process. Secondly, it is similar to the MCLUST algorithm used by Bettenburg et al. [8] and Menzies et al. [58]. Lastly, previous work [40] showed that the performance achieved by EM are close to those obtained by the algorithm originally proposed by Menzies et al. [58]. In the context of this study, we relied on the implementation of the algorithm available in the WEKA toolkit [30]. Given a project P_i, the input of the clustering algorithm was represented by the data coming from all the systems but P_i, *i.e.,* we still worked in a cross-project setting by means of the *Leave-One-Out Cross-Validation* [79] where the test sets were represented by the data of P_i. We created a bug prediction model relying on ASCI for each of the clusters.

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 [32]. Conversely, to properly evaluate the ability of our approach to predict the bug-proneness of classes we relied on Matthew's Correlation Coefficient (MCC), which is a measure indicating the extent to which the independent and dependent variables are well related to each other. Metric values close to 1 indicate higher performance. As shown by Hall et al. [32], this is the most reliable threshold-independent metric for the evaluation of bug prediction models.

As a final step of our analyses, we also statistically verified the validity of our findings. To this aim, we exploited the Nemenyi test [64] for statistical significance and report its results by mean of MCB (Multiple comparisons with the best) plots [50]. As a significance level, we used 0.05. The elements plotted above the gray band in the figures are statistically larger than the others.

4. Analysis of the Results

In this section, we present the results of our study, by discussing each research question independently.

4.1. $\mathbf{RQ}_1$ *- Evaluation of Ensemble Techniques when Adopted for Within-Project Bug Prediction*

Figure 1 depicts the box plots of the MCC achieved on the 21 software systems in our dataset by ASCI and the

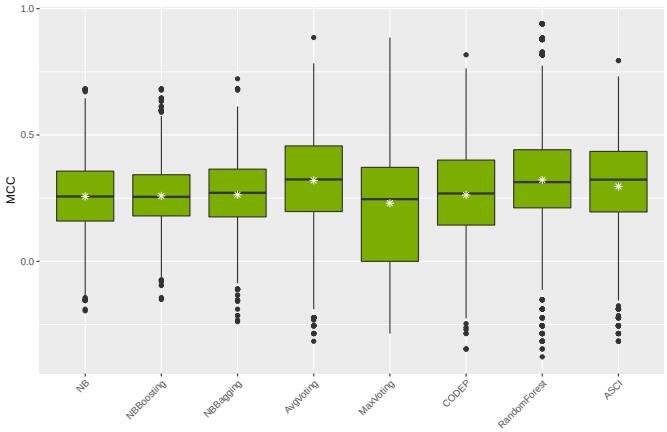

Figure 1: Boxplots of MCC achieved by the ensemble methods under study and NB in the within-project bug prediction context.

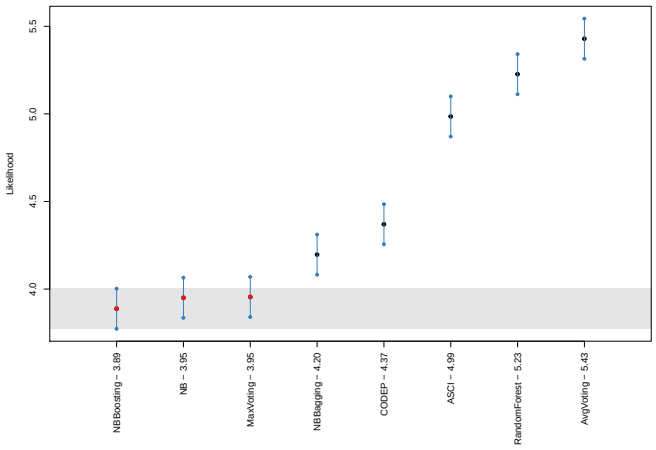

Figure 2: The likelihood of each technique in within prediction appearing in the top Nemenyi rank in terms of MCC. Circle dots are the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50%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

baselines experimented in within-project bug prediction models (white asterisks highlight the means).

As shown, ASCI performance is comparable with the other ensemble techniques: it has a median MCC slightly higher than all the baseline approaches but AvgVoting. However, it does not provide major improvements in the identification of buggy classes. Thus, our study confirms previous findings on the capabilities of ASCI [24].

As expected the performance of the model trained using Naïve Bayes (NB) are worse than most of the ensemble techniques. However, we can observe that most of the ensemble techniques perform in a similar manner. Furthermore, on the one hand, we can conclude that the use of ensemble techniques improves the performance of machine-learning models for defect prediction. On the other hand, most of the ensemble techniques are not better than the others from a statistically significant perspective. Indeed, looking at the other models, we found that AvgVoting on average obtains the best results (*e.g.,* +7% with respect to NB) but the improvement *is quite limited or negligible* with respect to what previous work [24, 91] depicted. It is worth noticing that AvgVoting performs better than MaxVoting, the same ensemble technique using the maximum operator.

More in general, we noticed that *within-project models suffer from high-performance variability* across different runs, meaning that variations in the training set lead to very different results. Thus, practitioners interested in setting up within-project models need to be careful when selecting the training set of machine learning techniques.

Figure 2 shows the likelihood of each analyzed ensemble technique along with Naïve Bayes to appear in the top Nemenyi rank. Interestingly, the statistical test showed that the differences observed among AvgVoting, RandomForest, and ASCI are negligible, meaning that they are statistically equivalent in terms of prediction capabilities. From a practitioner's perspective, this result suggests the selection of one of these three techniques while setting up a prediction model based on ensemble classifiers.

> ***Finding 1***. *In within-project bug prediction, using ensemble techniques improves the prediction performance with respect to the best stand-alone classifier (i.e.,* Naïve Bayes*). We confirm that the models based on* Validation and Voting *achieve slightly better results. However, they are similar to those obtained by other ensemble techniques.*

4.2. $\mathbf{RQ_2}$ - *Evaluation of Ensemble Techniques when Adopted for Cross-Project Bug Prediction*

Figure 3 depicts the box plots of the AUC-ROC and MCC achieved on the 21 software systems in our dataset by the experimented cross-project bug prediction models (white asterisks highlight the means).

Also in this context, we found Naïve Bayes to have a performance similar to that achieved by ensemble methods. Thus, we confirm again that the models based on ensemble classifiers do not necessarily provide improvements with respect to a well-selected stand-alone model. As for ASCI, our results clearly show that its performance is lower than Naïve Bayes, NBBoosting and NBBagging. At the same time, AvgVoting shows a slightly better accuracy with respect to ASCI in terms of MCC (+2% on average). Thus, the application of our technique in a cross-project setting does not provide improvements in the prediction of bugs: possibly, this means that ASCI suffers from the well-known problem of data heterogeneity.

In the second place, we noticed that the performance of the Validation and Voting approach depends on the operator used to combine the outputs of different stand-alone classifiers: specifically, AvgVoting tends to perform slightly better than MaxVoting in terms of MCC

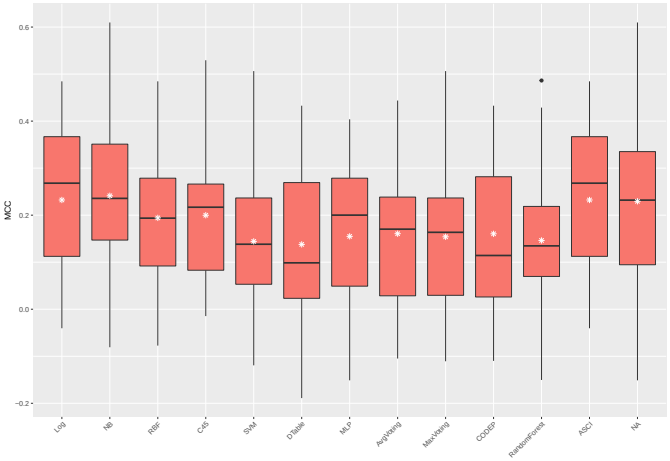

Figure 3: Boxplots of MCC achieved by the ensemble methods under study and NB in the cross-project bug prediction context.

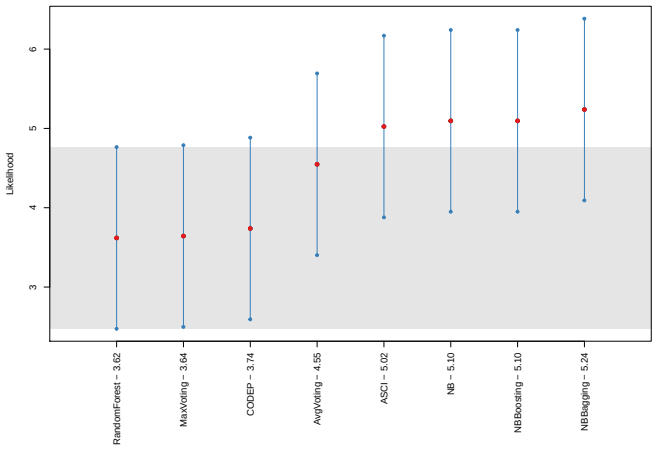

Figure 4: The likelihood of each technique in cross prediction appearing in the top Nemenyi rank in terms of MCC. Circle dots are the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the studied datasets.

(21% vs 18%). Thus, the setting of the technique has an impact on its performance by up to 3% in terms of MCC. Furthermore, unlike previous work [53, 99] we found that in some cases AvgVoting performs worse than other ensemble techniques. For example, the average MCC achieved by NBBoosting and NBBagging is respectively 4% and 3% higher (25% and 24% vs 21%).

A third interesting finding regards the performance of CODEP. Our results confirm those reported by Zhang et al. [99]: in particular, we found that AvgVoting outperforms CODEP (+2% in terms of MCC). Hence, *we can confirm that the* AvgVoting *approach tends to be more stable and accurate than* CODEP.

It is also interesting to discuss the results of the Random Forest technique. Previous work by Robnik-Šikonja [77] and Jiang et al. [43] observed that it is one of the most reliable and accurate machine learners, however also in this case we discovered that it is not able to provide improved performance with respect to other ensemble techniques. In particular, its performance is worse than AvgVoting in terms of MCC (*e.g.,* -2%).

As a more general observation, it is important to note that the performance of all the cross-project models experimented is quite low—on average they do not exceed 25% in terms of MCC. On the one hand, all the experimented models solely relied on code metrics as independent variables. As suggested by literature [19, 23, 60] a combination of predictors of different natures (*e.g.,* process metrics) can affect the overall performance of bug prediction models. On the other hand, our results still suggest that cross-project bug prediction is still far from being actually usable in practice. For this reason, the research community needs to investigate more the problem, trying to identify useful tools to make cross-project bug prediction actually effective.

The results discussed so far are also statistically significant: Figure 4 shows the likelihood of each analyzed ensemble techniques along with Naïve Bayes to appear in the top Nemenyi rank. Naïve Bayes and the ensemble using it as weak learner (*e.g.,* NBBoosting and NBBagging), together with ASCI, are able to achieve the best performance in terms of MCC. All the other experimented ensemble methods are statistically worst.

> ***Finding 2.*** *None of the cross-project experimented models exceeds 25% in terms of MCC on average. Identifying buggy classes using external sources of information is still an open problem. Furthermore, the use of ensemble techniques does not provide evident benefits with respect to stand-alone classifiers: the models based on* Naïve Bayes *or using it as a weak learner (e.g.,* NBBoosting *and* NBBagging*) achieve the best performance.*

4.3. $\mathbf{RQ_3}$: *Evaluation of Ensemble Techniques when Adopted for Local Cross-Project Bug Prediction*

On the basis of the results achieved in $\mathbf{RQ_2}$, we verified whether the application of local learning—that was suggested as a promising way to reduce data heterogeneity—could improve the performance of ASCI. Figure 5 depicts the box plots reporting MCC achieved on the 21 subject systems when combining local learning and the ensemble techniques considered in our study, along with those achieved by the standard local bug prediction model that relies on Naïve Bayes. To ease the comparison with the results of $\mathbf{RQ_2}$, we also report box plots for the global models built using the same set of classifiers.

As a first observation, *local models do not always achieve better performance with respect to global models.* While we could confirm previous findings [40], we also observed that *the use of ensemble techniques for local bug*

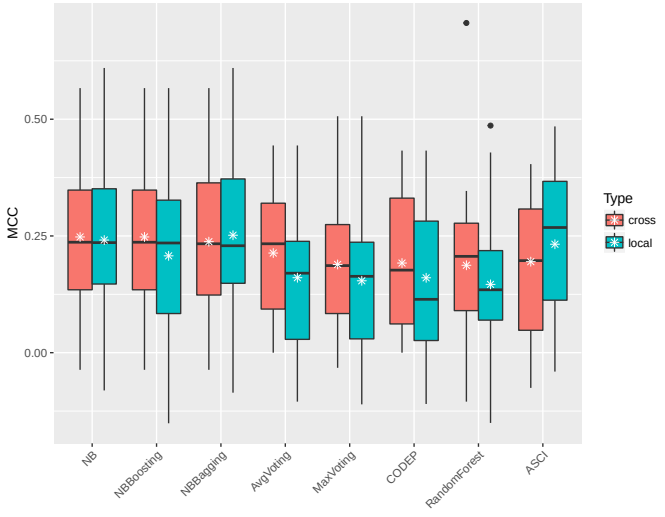

Figure 5: Boxplots of MCC achieved by the ensemble methods under study and NB in the global/local cross-project bug prediction context.

prediction can sometimes badly affect the capabilities of the resulting models. For example, NBBoosting showed an average decrease in terms of MCC (*i.e.,* -6%). Furthermore, in the comparison with Naïve Bayes, techniques like NBBoosting and NBBagging showed similar performance as the one of stand-alone classifiers. Another interesting observation concerns the results of ensemble techniques combining the output of different classifiers, *i.e.,* Validation and Voting and CODEP. In these cases, local models tend to provide worse performance than global ones. As for the two Validation and Voting techniques experimented, a likely motivation for such result comes from the characteristics of the algorithms: as shown in previous research [24, 75], this technique fails in case of high variability among the predictions provided by different classifiers because the majority of the base classifiers might wrongly classify the bug-proneness of a class, thus negatively influencing the performance of techniques which combine the output of different classifiers. A similar conclusion can be drawn when applying Validation and Voting. From a quantitative point of view, the local scenario reduces the average MCC achieved by AvgVoting and MaxVoting by up to 5% and 3%, respectively, when compared to global models.

Looking at Random Forest, we found that this technique was badly affected when applying the clustering technique. Specifically, local models exploiting such classifiers obtained a median MCC 7% lower with respect to global cross-project models built using the same classifiers.

Besides the results discussed so far, we found ASCI to be the only classifier actually able to exploit the lower heterogeneity of data provided by local models. Indeed, it was able to boost its MCC of 7%, becoming much more effective than the baselines, both considering their global and local versions. In other words, *the local version of* ASCI *provides better performance with respect to all the other global and local models experimented.*

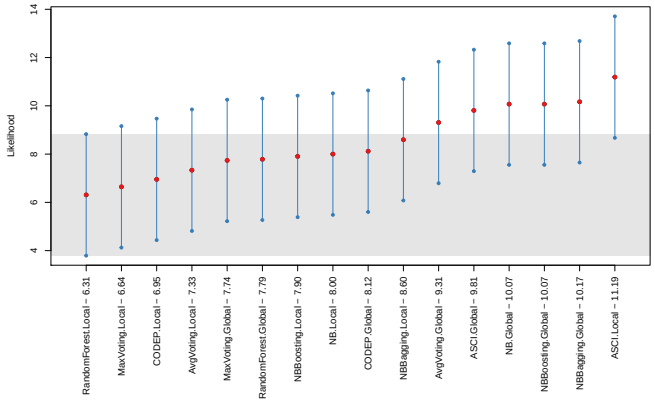

Figure 6: The likelihood of each technique in global/local cross prediction appearing in the top Nemenyi rank in terms of MCC. Circle dots are the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the datasets.

As previously done, we performed the Nemenyi test. Figure 6 shows the likelihood of each of the analyzed ensemble techniques along with Naïve Bayes to appear in the top Nemenyi rank. Also in this case, we report both *local* and *global* cross-project bug prediction models for the sake of understandability. We could notice that three global models, *i.e.,* NBBagging.Global, NBBoosting.Global, and NB.Global, have a similar average likelihood as the best local model, *i.e.,* ASCI.Local. Thus, we can conclude that *local and global models are mostly equivalent from a statistical point of view.* Moreover, local bug prediction should be considered a valuable option when applying ASCI in the context of cross-project bug prediction.

> ***Finding 3***. *Local learning often does not improve the performance of bug prediction models. The only exception is represented by* ASCI*, in which the local-project setting slightly improves the performance. However, the statistical analysis highlighted that local and global models are mostly equivalent in terms of performance.*

5. Discussion and Further Insights

In this section, we provide a deeper discussion of our findings.

Figure 7 shows the box plots reporting MCC values achieved by global and local cross-project models as well

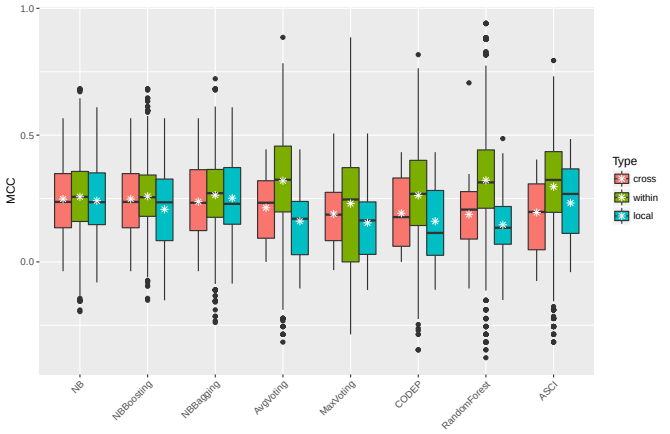

Figure 7: Boxplots of MCC achieved by the ensemble methods under study and NB in the global/local cross-project and within-project bug prediction contexts.

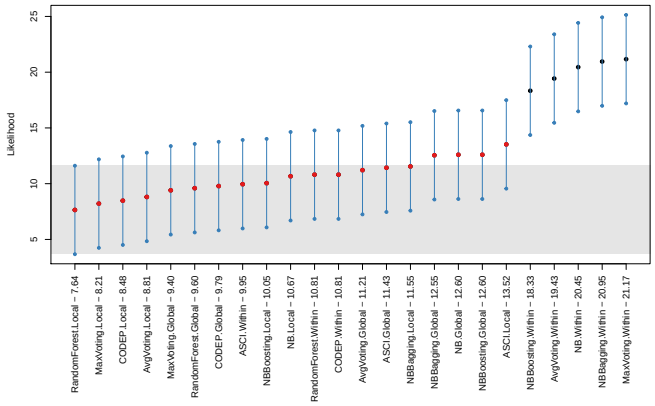

Figure 8: The likelihood of each technique in within and global/local cross-project prediction appearing in the top Nemenyi rank in terms of MCC. Circle dots indicate the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the studied datasets.

as by within-project models built using the ensemble techniques considered in our empirical study.

Looking at it, we can conclude that *within-project models generally exhibit better performance with respect to cross-project ones*, thus confirming previous findings by Turhan et al. [90]. The result holds for all the experimented ensemble techniques. In general, we noticed that also in the within-project scenario the *use of ensemble classifiers does not guarantee better prediction performance with respect to stand-alone models*, *e.g.,* MaxVoting performs 1% worse than Naive Bayes.

Looking more in-depth, we noticed that the model relying on Naive Bayes slightly improve its capabilities when trained using a within-project strategy (+1% and +2% with respect to global and local cross-project models, respectively). This result is confirmed also when applying Boosting (+1% and +5%, respectively) and Bagging (+2% and +1%, respectively).

Interesting is the case of models based on Validation and Voting. We found a significant performance decay when changing the training strategy, independently from the combination operation (*i.e.,* Average or Maximum): the MCC of AvgVoting and MaxVoting in the cross-project evaluation context were 11% and 4%, respectively, lower than those achieved in the within-project context and even lower when looking at the local models (-15% and -8%, respectively). Also in the case of classifiers based on the combination of multiple learners (*i.e.,* CODEP, ASCI, and RandomForest) we observed important differences when considering a within- or cross-project training. In particular, the average MCC reported by Random Forest dropped by 13%, while CODEP and ASCI dropped by 7% and 10%, respectively.

The Nemenyi test in Figure 8 shows the likelihood of each of the analyzed ensemble techniques in the contexts of local, within-project, and cross-project bug prediction. The statistical analyses confirmed the results discussed so far: indeed, within-project models are able to achieve better performance in terms of MCC, thus being statistically more accurate than the others.

As an additional analysis aimed at measuring the robustness of the experimented models, we computed the Area Under the ROC Curve (AUC-ROC). This metric, ranging between 0.5 and 1, reports the overall capabilities of a model in discriminating buggy and non-buggy classes. Values close to 1 indicate higher performance. 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 of its robustness [34] (*i.e.,* how well the classifier separates the binary classes).

Figure 9 shows boxplots representing the performance of global cross-, local cross-, and within-project models in terms of AUC-ROC. As shown, we observed that *cross-project models generally perform better than within-project ones*—this result shows that an interpretation solely based on F-Measure does not provide a comprehensive picture of the performance of bug prediction models. Looking deeper into the results, we found that cross-project models behave similarly, if not better than within-project ones when considering the AUC-ROC. From a practical point of view, this means that *cross-project models are more robust than within-project ones* (*i.e.,* their performance does not vary as much as one of cross-projects models when run on different data), and at the same time *within-project models are more precise than cross-project ones* (*i.e.,* their accuracy is higher than the one of cross-project models). This result seems to suggest that within- and cross-project strategies have different pros and cons, being to some extent complementary: as part of our future agenda, we plan to further

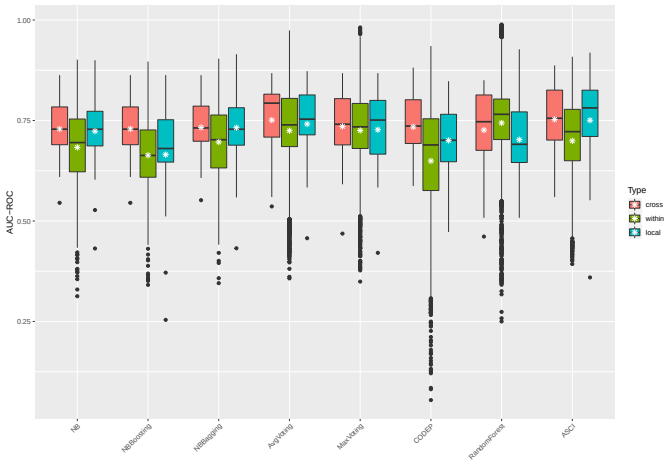

Figure 9: Boxplots of AUC-ROC achieved by the ensemble methods under study and NB in the global/local cross-project and within-project bug prediction contexts.

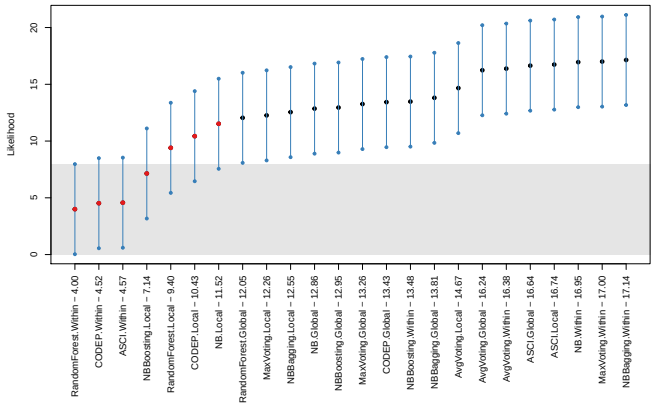

Figure 10: The likelihood of each technique in within and global/local cross prediction appearing in the top Nemenyi rank in terms of AUC-ROC. Circle dots indicate the median likelihood, while the error bars indicate the 95% confidence interval. 50% of likelihood means that a classification technique appears at the top-rank for 50% of the studied datasets.

investigate the extent to which a smart mixture of both the strategies can lead to better bug prediction performance.

As a final discussion point, it is worth noting that the local version of ASCI is the technique that performs better than all the others. This result confirms that such a technique should be preferred in case robustness is the main objective that a practitioner wants to achieve when running a bug prediction model.

The Nemenyi test in Figure 10 shows the likelihood of each of the analyzed ensemble techniques along with Naive Bayes in the contexts of within and cross-project bug prediction. The statistical analyses confirmed the results discussed so far: indeed, there is no significant statistical difference between within-project bug prediction models and cross-project ones when considering the AUC-ROC.

6. 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 exploited. We relied on several datasets available in the Seacraft repository [59], which is widely considered reliable and, indeed, has been also used in several previous works in the field of bug prediction [13, 99, 91, 71, 28, 53, 52, 70]. Although we cannot exclude possible imprecision and/or incompleteness of the data used in the study, we applied a formal data preprocessing recommended by Shepperd et al. [81], 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 [85].

As for the experimented prediction models, we exploited the implementation provided by the Weka framework [30], 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 [7]. As a future goal, we plan to further analyze the impact of parameters' configuration on our findings.

Threats to conclusion validity. These are related to the relation between treatment and outcome. To reduce the impact of the adopted validation methodology, we relied on the *Leave-One-Out Cross-Validation* methodology [79]. This choice was driven by results recently reported that showed that such a validation technique is among the ones that are more stable and reliable [86].

To ensure that the results would have not been biased by confounding effects due to data unbalance [17] or highly correlated independent variables [25], we adopted formal procedures aimed at (i) over-sampling the training sets [17] and (ii) removing non-relevant independent variables through feature selection [31].

As for the evaluation of the performance of the experimented models, we considered AUC-ROC and MCC, which have been highly recommended [32, 96] to correctly interpret the results. Finally, we also statistically verified the validity of our findings by exploiting the Nemenyi test [64] for statistical significance.

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.). Of course, we cannot claim the generalizability with respect to industrial environments, however, a replication of our study in different settings, including the industrial one, is part of our future research agenda.

Regarding the selected ensemble techniques, we considered those representing the state of the art [13, 71]. Finally, it is important to note that the features in our models are 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 Hassan [35]) and developer-related (*e.g.,* the number of developers working on a code component [5]) metrics on our findings.

7. Conclusion

In this paper, we aimed at corroborating the results achieved when evaluating ASCI in within- and cross-project scenarios and benchmarking it with respect to a variety of alternative ensemble techniques on a set of 21 software projects from the PROMISE dataset. In so doing, we mitigated possible threats to validity affecting previous benchmarking studies through the application of some precautions concerned with the quality of data used.

We found that the problem of cross-project bug prediction is still far from being solved. The use of ensemble techniques does not provide evident benefits with respect to stand-alone classifiers, but in general, the VALIDATION AND VOTING and ASCI techniques should be preferred among other ensemble methods.

When turning our attention to the combination of local learning and ensemble classifiers, we did not observe major differences; indeed, the statistical analyses revealed that local and global models are mostly equivalent. Nevertheless, we found that ASCI is the only technique that is effective in exploiting local learning to reduce data heterogeneity and improve its prediction capabilities.

Finally, we provide insights into the relation between cross- and within-project models. In the first place, the latter are more precise than cross-project models, independently from the training strategy (global vs local). Nevertheless, the use of ensemble classifiers in the context of within-project models does not guarantee better prediction performance with respect to models relying on stand-alone classifiers. On the other hand, we also observed that cross-project models are more robust, meaning that the two strategies might be complemented in order to take advantage of the pros of each strategy.

Our findings provide some key implications for both researchers and practitioners. For researchers, our results confirm that the problem of cross-project bug prediction still needs noticeable attention in order to devise methodologies to properly transfer external information into a new context, especially because existing local learning methods and ensemble techniques do not represent yet a suitable solution; moreover, within-project models are not bullet-proof and their robustness cannot be improved by means of ensemble techniques: thus, more research is needed to overcome such limitation. For practitioners, our findings suggest that the use of more sophisticated techniques to mix different classifiers does not provide immediate advantages: thus, they need to carefully evaluate the suitability of ensemble methods before using them. Similarly, collecting bug-related data from the project under development still represents the most effective way to apply bug prediction in practice, even because the use of cross-project models might actually lead to higher inspection costs.

The observations and implications discussed above represent the main starting point for our future research agenda. Furthermore, we plan to replicate the study in industrial and larger contexts, using a richer set of independent variables, and investigating the impact of classifiers configuration on our findings. In particular, we aim at building a larger dataset based on the bug fixes that Herbold et al. [41] are mining and validating. More importantly, following the suggestions by Lanza et al. [51] we plan to perform a user study with developers aimed at evaluating the real usefulness of the suggestions provided by the different bug prediction models experimented. Finally, we plan to investigate how to combine cross-project and within-project strategies to achieve more accurate and robust models.

Acknowledgment

Dario is supported by the European Commission grants no. # 825040 (RADON H2020).

References

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

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

[3] Basili, V., Briand, L., Melo, W., Oct 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 22 (10), 751–761.

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

[5] Bell, R., Ostrand, T., Weyuker, E., 2013. The limited impact of individual developer data on software defect prediction. Empirical Software Engineering 18 (3), 478–505.

[6] Bennin, K. E., Keung, J., Phannachitta, P., Monden, A., Mensah, S., 2017. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Transactions on Software Engineering PP (99), 1–1.

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

[8] Bettenburg, N., Nagappan, M., Hassan, A. E., 2012. Think locally, act globally: Improving defect and effort prediction

models. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on. IEEE, pp. 60–69.
[9] Bettenburg, N., Nagappan, M., Hassan, A. E., 2015. Towards improving statistical modeling of software engineering data: think locally, act globally! Empirical software engineering 20 (2), 294–335.
[10] Bezerra, M. E., Oliveira, A. L., Adeodato, P. J., Meira, S. R., 2008. Enhancing RBF-DDA algorithm's robustness: Neural networks applied to prediction of fault-prone software modules. Springer, pp. 119–128.
[11] Bezerra, M. E., Oliveira, A. L., Meira, S. R., 2007. A constructive rbf neural network for estimating the probability of defects in software modules. In: 2007 International Joint Conference on Neural Networks. IEEE, pp. 2869–2874.
[12] Bowes, D., Hall, T., Harman, M., Jia, Y., Sarro, F., Wu, F., 2016. Mutation-aware fault prediction. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, pp. 330–341.
[13] Bowes, D., Hall, T., Petrić, J., 2017. Software defect prediction: do different classifiers find the same defects? Software Quality Journal, 1–28.
[14] Breiman, L., 1996. Bagging predictors. Machine learning 24 (2), 123–140.
[15] Camargo Cruz, A. E., Ochimizu, K., 2009. Towards logistic regression models for predicting fault-prone code across software projects. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, pp. 460–463.
[16] Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S., 2013. Multi-objective cross-project defect prediction. In: Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE Computer Society, pp. 252–261.
[17] Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., 2002. Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research 16 (1), 321–357.
[18] Chidamber, S., Kemerer, C., Jun 1994. A metrics suite for object oriented design. Software Engineering, IEEE Transactions on 20 (6), 476–493.
[19] D'Ambros, M., Lanza, M., Robbes, R., 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering 17 (4-5), 531–577.
[20] Dean, D. J., Nguyen, H., Gu, X., 2012. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In: Proceedings of the 9th International Conference on Autonomic Computing. ACM, pp. 191–200.
[21] Dempster, A. P., Laird, N. M., Rubin, D. B., 1977. Maximum likelihood from incomplete data via the em algorithm. Journal of the royal statistical society. Series B (methodological), 1–38.
[22] Di Nucci, D., Palomba, F., De Lucia, A., 2018. Evaluating the adaptive selection of classifiers for cross-project bug prediction. In: 2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). IEEE, pp. 48–54.
[23] Di Nucci, D., Palomba, F., De Rosa, G., Bavota, G., Oliveto, R., De Lucia, A., 2017. A developer centered bug prediction model. IEEE Transactions on Software Engineering PP (99), 1–1.
[24] Di Nucci, D., Palomba, F., Oliveto, R., De Lucia, A., 2017. Dynamic selection of classifiers in bug prediction: An adaptive method. IEEE Transactions on Emerging Topics in Computational Intelligence 1 (3), 202–212.
[25] Dietterich, T., Sep. 1995. Overfitting and undercomputing in machine learning. ACM Computing Surveys 27 (3), 326–327.
[26] Elish, M. O., 2014. A comparative study of fault density prediction in aspect-oriented systems using mlp, rbf, knn, rt, denfis and svr models. Artificial Intelligence Review 42 (4), 695–703.
[27] Freund, Y., Schapire, R. E., 1996. Experiments with a new boosting algorithm. In: Icml. Vol. 96. pp. 148–156.
[28] Ghotra, B., McIntosh, S., Hassan, A. E., 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the International Conference on Software Engineering. IEEE, pp. 789–800.
[29] Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B., 2011. The misuse of the nasa metrics data program data sets for automated software defect prediction. In: Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on. IET, pp. 96–103.
[30] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H., 2009. The weka data mining software: An update. SIGKDD Explorations Newsletter. 11 (1), 10–18.
[31] Hall, M. A., 1998. Correlation-based feature selection for machine learning. Tech. rep.
[32] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2011. Developing fault-prediction models: What the research can show industry. IEEE Software 28 (6), 96–99.
[33] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2012. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering 38 (6), 1276–1304.
[34] Hanley, J. A., McNeil, B. J., 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology 143 (1), 29–36.
[35] Hassan, A. E., 2009. Predicting faults using the complexity of code changes. In: ICSE. IEEE Press, Vancouver, Canada, pp. 78–88.
[36] He, P., Li, B., Liu, X., Chen, J., Ma, Y., 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology 59 (C), 170–190.
[37] He, Z., Peters, F., , T., Yang, Y., 2013. Learning from open-source projects: An empirical study on defect prediction. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 45–54.
[38] Herbold, S., 2013. Training data selection for cross-project defect prediction. In: Proceedings of the 9th International Conference on Predictive Models in Software Engineering. ACM, p. 6.
[39] Herbold, S., Trautsch, A., Grabowski, J., 2017. A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering PP (99), 1–1.
[40] Herbold, S., Trautsch, A., Grabowski, J., 2017. Global vs. local models for cross-project defect prediction. Empirical Software Engineering 22 (4), 1866–1902.
[41] Herbold, S., Trautsch, A., Ledel, B., 2020. Large-scale manual validation of bugfixing changes. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 611–614.
[42] Ho, T. K., 1995. Random decision forests. In: Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on. Vol. 1. IEEE, pp. 278–282.
[43] Jiang, Y., Cukic, B., Menzies, T., 2008. Can data transformation help in the detection of fault-prone modules? In: Proceedings of the 2008 Workshop on Defects in Large Software Systems. ACM, pp. 16–20.
[44] Jureczko, M., Madeyski, L., 2010. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th international conference on predictive models in software engineering. pp. 1–10.
[45] Khoshgoftaar, T. M., Gao, K., Seliya, N., 2010. Attribute selection and imbalanced data: Problems in software defect prediction. In: 2010 22nd IEEE International Conference on Tools with Artificial Intelligence. Vol. 1. pp. 137–144.
[46] Khoshgoftaar, T. M., Goel, N., Nandi, A., McMullan, J., 1996. Detection of software modules with high debug code churn in a very large legacy system. In: Software Reliability Engineering. IEEE, pp. 364–371.
[47] Kim, S., Zhang, H., Wu, R., Gong, L., 2011. Dealing with noise in defect prediction. In: Proceedings of International Conference on Software Engineering. IEEE, pp. 481–490.

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

[49] Kittler, J., Hatef, M., Duin, R. P., Matas, J., 1998. On combining classifiers. IEEE transactions on pattern analysis and machine intelligence 20 (3), 226–239.

[50] Koning, A. J., Franses, P. H., Hibon, M., Stekler, H. O., 2005. The m3 competition: Statistical tests of the results. International Journal of Forecasting 21 (3), 397–409.

[51] Lanza, M., Mocci, A., Ponzanelli, L., 2016. The tragedy of defect prediction, prince of empirical software engineering research. IEEE Software 33 (6), 102–105.

[52] Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 34 (4), 485–496.

[53] Liu, Y., Khoshgoftaar, T. M., Seliya, N., 2010. Evolutionary optimization of software quality modeling with multiple repositories. IEEE Transactions on Software Engineering 36 (6), 852–864.

[54] Lumpe, M., Vasa, R., Menzies, T., Rush, R., Turhan, B., 2012. Learning better inspection optimization policies. International Journal of Software Engineering and Knowledge Engineering 22 (05), 621–644.

[55] Ma, Y., Luo, G., Zeng, X., Chen, A., 2012. Transfer learning for cross-company software defect prediction. Information and Software Technology 54 (3), 248–256.

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

[57] Marcus, A., Poshyvanyk, D., Ferenc, R., 2008. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Transactions on Software Engineering 34 (2), 287–300.

[58] Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., Zimmermann, T., 2013. Local versus global lessons for defect prediction and effort estimation. IEEE Transactions on software engineering 39 (6), 822–834.

[59] Menzies, T., Krishna, R., Pryor, D., 2017. The seacraft repository of empirical software engineering data.
URL `https://zenodo.org/communities/seacraft`

[60] Moser, R., Pedrycz, W., Succi, G., 2008. 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, pp. 309–311.

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

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

[63] Nam, J., Pan, S. J., Kim, S., 2013. Transfer defect learning. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp. 382–391.

[64] Nemenyi, P., 1962. Distribution-free multiple comparisons. In: Biometrics. Vol. 18. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, p. 263.

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

[66] Nguyen, T. T., Nguyen, T. N., Phuong, T. M., 2011. Topic-based defect prediction (nier track). In: Proceedings of the 33rd International Conference on Software Engineering. ACM, pp. 932–935.

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

[68] O'brien, R. M., 2007. A caution regarding rules of thumb for variance inflation factors. Quality & Quantity 41 (5), 673–690.

[69] Pai, G. J., Dugan, J. B., 2007. Empirical analysis of software fault content and fault proneness using bayesian methods. IEEE Transactions on Software Engineering 33 (10), 675–686.

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

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

[72] Pascarella, L., Palomba, F., Bacchelli, A., 2019. Fine-grained just-in-time defect prediction. Journal of Systems and Software 150, 22–36.

[73] Pascarella, L., Spadini, D., Palomba, F., Bruntink, M., Bacchelli, A., 2018. Information needs in contemporary code review. Proceedings of the ACM on Human-Computer Interaction 2 (CSCW), 1–27.

[74] Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A., 2020. Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performance and a benchmark study - replication package.
URL `https://bit.ly/36ECrTE`

[75] Petrić, J., Bowes, D., Hall, T., Christianson, B., Baddoo, N., 2016. Building an ensemble for software defect prediction based on diversity selection. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, p. 46.

[76] Pinzger, M., Nagappan, N., Murphy, B., 2008. Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 1–12.

[77] Robnik-Šikonja, M., 2004. Improving random forests. In: European conference on machine learning. Springer, pp. 359–370.

[78] Rokach, L., 2010. Ensemble-based classifiers. Artificial Intelligence Review 33 (1), 1–39.

[79] Sammut, C. (Ed.), 2010. Leave-One-Out Cross-Validation. Springer US, Boston, MA, pp. 600–601.

[80] Scanniello, G., Gravino, C., Marcus, A., Menzies, T., 2013. Class level fault prediction using software clustering. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, pp. 640–645.

[81] Shepperd, M., Song, Q., Sun, Z., Mair, C., Sept 2013. Data quality: Some comments on the nasa software defect datasets. Software Engineering, IEEE Transactions on 39 (9), 1208–1215.

[82] Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J., 2011. A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering 37 (3), 356–370.

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

[84] Tantithamthavorn, C., Hassan, A. E., Matsumoto, K., 2018. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Transactions on Software Engineering.

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

[86] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Transanctions on Software Engineering 43 (1), 1–18.

[87] Thomas, S. W., Nagappan, M., Blostein, D., Hassan, A. E., 2013. The impact of classifier configuration and classifier combination on bug localization. IEEE Transactions on Software Engineering 39 (10), 1427–1443.

[88] Todd L. Graves, Alan F. Karr, J. S. M., Siy, H. P., 2000. Predicting fault incidence using software change history. Software Engineering, IEEE Transactions on 26 (7), 653–661.

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

[90] Turhan, B., Menzies, T., Bener, A. B., Di Stefano, J., 2009. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 14 (5), 540–578.

[91] Wang, T., Li, W., Shi, H., Liu, Z., 2011. Software defect prediction based on classifiers ensemble. Journal of Information & Computational Science 8 (16), 4241–4254.

[92] Watanabe, S., Kaiya, H., Kaijiri, K., 2008. Adapting a fault prediction model to allow inter languagereuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering. ACM, pp. 19–24.

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

[94] Wolpert, D. H., 1992. Stacked generalization. Neural networks 5 (2), 241–259.

[95] Xu, Z., Liu, J., Yang, Z., An, G., Jia, X., 2016. The impact of feature selection on defect prediction performance: An empirical comparison. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). IEEE, pp. 309–320.

[96] Yao, J., Shepperd, M., 2020. Assessing software defection prediction performance: why using the matthews correlation coefficient matters. In: Proceedings of the Evaluation and Assessment in Software Engineering. pp. 120–129.

[97] Zhang, F., Keivanloo, I., Zou, Y., 2017. Data transformation in cross-project defect prediction. Empirical Software Engineering, 1–33.

[98] Zhang, F., Zheng, Q., Zou, Y., Hassan, A. E., 2016. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th International Conference on Software Engineering. ACM, pp. 309–320.

[99] Zhang, Y., Lo, D., Xia, X., Sun, J., 2015. 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, pp. 264–269.

[100] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B., 2009. 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. ACM, pp. 91–100.

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