Evaluating the Adaptive Selection of Classifiers for Cross-Project Bug Prediction

Dario Di Nucci
Vrije Universiteit Brussel, Belgium
ddinucci@vub.ac.be

Fabio Palomba
University of Zurich, Switzerland
palomba@ifi.uzh.ch

Andrea De Lucia
University of Salerno, Italy
adelucia@unisa.it

ABSTRACT

Bug prediction models are used to locate source code elements more likely to be defective. One of the key factors influencing their performances is related to the selection of a machine learning method (a.k.a., classifier) to use when discriminating buggy and non-buggy classes. Given the high complementarity of stand-alone classifiers, a recent trend is the definition of *ensemble* techniques, which try to effectively combine the predictions of different stand-alone machine learners. In a recent work we proposed ASCI, a technique that dynamically selects the right classifier to use based on the characteristics of the class on which the prediction has to be done. We tested it in a within-project scenario, showing its higher accuracy with respect to the Validation and Voting strategy. In this paper, we continue on the line of research, by (i) evaluating ASCI in a *global* and *local* cross-project setting and (ii) comparing its performances with those achieved by a stand-alone and an ensemble baselines, namely Naive Bayes and Validation and Voting, respectively. A key finding of our study shows that ASCI is able to perform better than the other techniques in the context of cross-project bug prediction. Moreover, despite local learning is not able to improve the performances of the corresponding models in most cases, it is able to improve the robustness of the models relying on ASCI.

CCS CONCEPTS

• **Software and its engineering → Software maintenance**;

KEYWORDS

Bug Prediction; Cross-project; Ensemble Classifiers

ACM Reference Format:
Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. 2018. Evaluating the Adaptive Selection of Classifiers for Cross-Project Bug Prediction. In *RAISE'18 : RAISE'18 :IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194104.3194112

1 INTRODUCTION

Bug prediction is the branch of software engineering that aims at discovering which are the source code elements more prone to be affected by faults so that developers may carry out focused testing activities [19]. Besides the adoption of unsupervised techniques prioritizing inspection and testing tasks based on (i) self-organizing maps [10], (ii) topic modeling [31], or (iii) connectivity metrics [46], the research community has been widely studying the usage of supervised techniques, which involve the definition of bug prediction models relating a set of independent variables (a.k.a., predictors) to the bug-proneness of code artifacts using machine learning methods [19]. Such models can be trained using a sufficient amount of labeled data coming from (i) the previous history of the same project where the model is applied to, i.e. using a *within-project* strategy, or (ii) other similar projects, i.e. using a *cross-project* strategy.

Previous studies showed that within-project bug prediction models have higher capabilities than cross-project ones since they rely on data that represents better the characteristics of the source code elements of the project where the models have to be applied [44]. As a drawback, a within-project training strategy cannot often be adopted in practice since new projects might not have enough data to setup a bug prediction model [48]. As a consequence, the research community has started investigating ways to make cross-project bug prediction models more effective with the aim of allowing a wider adoption of bug prediction models [21, 27]. For instance, Menzies et al. [25] recently proposed the concept of *local* bug prediction introducing a technique that (i) firstly clusters homogeneous data coming from different projects with the aim of reducing its heterogeneity and (ii) then builds for each cluster a different model using the Naive Bayes classifier [30]. The empirical analysis showed that such a technique can significantly improve the performances of *global* bug prediction models, thus paving the way for a re-visitation of cross-project techniques.

Looking at the literature on cross-project bug prediction, it is clear that one of the factors that impacts more the performances of such techniques is represented by the choice of the machine learning method used to discriminate buggy and non-buggy instances: as a matter of fact, Ghotra et al. [16] demonstrated that a wrong selection of the classifier might impact the model capabilities up to 30%; perhaps more importantly, Panichella et al. [34] found that different classifiers perform similarly, being however highly complementary. As a direct consequence of these findings, researchers have tried to exploit the complementarity among different classifiers by means of *ensemble techniques*, that is, methodologies able to combine together different classifiers to improve bug prediction performances [5, 13, 24, 34, 36, 45]. For instance, Tosun et al. [43] experimented with the Validation and Voting strategy, a two-step method that firstly builds a set of bug prediction models relying on different classifiers, and then predicts a class as buggy in case the majority of models predict the bugginess of the class.

More recently, we proposed a novel ensemble technique, coined ASCI (Adaptive Selection of Classifiers in bug predIction) [13], able

RAISE'18 , May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5723-4/18/05…$15.00
https://doi.org/10.1145/3194104.3194112

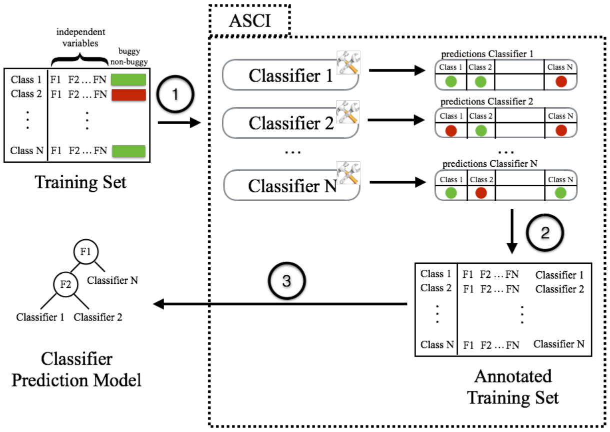

Figure 1: ASCI - Workflow.

to dynamically select among a set of machine learning classifiers the one which better predicts the bug proneness of a class based on its characteristics. We tested its performances in an empirical study involving 30 software systems, where we trained ASCI in a within-project setting. As a result we found that ASCI was able to achieve higher performances than (i) the base classifiers used to build it and (ii) the Validation and Voting ensemble classifier.

In this paper, we aim at making a further step ahead by evaluating ASCI in a cross-project setting, considering both *global* and *local* training strategies. The study was conducted on a Promise dataset composed of 10 software systems, where we applied a number of corrections suggested by Shepperd et al. [38] in order to make it cleaned and suitable for our purpose. We compared our technique with Validation and Voting and Naive Bayes, that we found to be the best stand-alone classifier in the context of our study.

The results suggest that the problem of predicting bugs using cross-project information is still far to be solved, since none of the experimented models exceed 40% in terms of F-Measure. The use of ensemble techniques improves the performances of prediction models, and indeed ASCI is the classifier performing better than Naive Bayes and Validation and Voting when considering the F-Measure and AUC-ROC. At the same time, we found that the local learning in most cases is not able to effectively reduce data heterogeneity and improve the performances obtained by the models. Despite this, we found that combining ASCI with local learning leads to more robust models.

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 results, as well as possible threats to validity. Finally, Section 5 concludes the paper and describes our future research agenda on the topic.

2 BACKGROUND AND RELATED WORK

This section summarizes the related literature and reports an overview of ASCI.

2.1 Related Literature

For sake of space limitations, we only briefly summarize the papers on cross-project bug prediction and ensemble techniques on which this study builds upon.

Cross-project bug prediction. 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 [48]. 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 [19], a notable effort has been also devoted to *how* to make external data suitable for the project under analysis [25]. 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, thus worsening the performances of bug prediction models. One of most recent advances in this area is represented by the so-called *local learning* methods. These approaches 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. [25] studied for the first time the effectiveness of local learning, finding that it can substantially improve the performances of bug prediction models with respect to global models.

Ensemble techniques. The need of using ensemble techniques comes from previous studies that showed that there is no classifier able to clearly outperform the others [4], since their performance strongly depend on the specific dataset considered [15]. More importantly, Ghotra et al. [16] 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. [34] 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 performances. Tosun et al. [43] experimented with 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 literature are based on the Bagging ensemble technique [36], 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. [23] combined multiple training data obtained applying a random sampling. More recently, some approaches inspired to the Stacking ensemble technique [36] have been proposed [34, 35]. 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. [34] devised CODEP, an approach that firstly 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. [47] conducted a similar study as the one performed by [34] comparing different ensemble approaches. They found that there exist several ensemble techniques that improve the performances achieved by CODEP, and Validation and Voting is often one of them. Petric et al. [35] used 4 families of classifiers in order to build a Stacking ensemble technique [36] 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, it is worth mentioning the work by Wang et al. [45], who compared the performances 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.

2.2 Adaptive Selection of Classifiers

Figure 1 reports the main steps adopted by ASCI to recommend which classifier should be used for evaluating the bugginess of a class. More specifically:

(1) Let $C = \{c_1, ..., c_n\}$ be a set of n different stand-alone machine learners, and let $T = \{e_1, ..., e_m\}$ be the set of classes composing the training set. Each $c_i \in C$ is run against the set T, so that the predictions of each classifier for each $e_j \in T$ are collected.

(2) After the first step, each $e_j \in T$ is labeled with the information regarding the classifier $c_i \in C$ which correctly identified its bugginess. Two possible scenarios can arise. If only one classifier c_i is able to predict the bug-proneness of e_j, then e_j will be associated with c_i. Conversely, if more classifiers or none of them correctly identified the bugginess of e_j, the classifier c_i having the highest F-Measure on the whole training set is assigned to e_j. As output of this step, an annotated training set T' is created.

(3) In the last step, based on T', ASCI builds a *classifier prediction model* using a decision tree DT as classifier. In other words, given the code metrics of the classes in the annotated training set as independent variables, it aims at predicting which classifier $c_i \in C$ should be used for that class. Thus, a prediction consists of a nominal value indicating the name of the classifier $c_i \in C$ that is considered as most indicated to properly classify the bug-proneness of $e_j \in T$.

It is worth noting that in our previous work [13] we evaluated ASCI in the within-project context, comparing it with (i) the base classifiers used to build it, i.e. Naive Bayes, Logistic Regression, Multi Layer Perceptron, J48, Radial Basis Function Network, and (ii) the Validation and Voting ensemble classifier.

3 EMPIRICAL STUDY DESIGN

The *goal* of the study is to evaluate how ASCI works when adopted for cross-project bug prediction and what is the effect of local learning on its performances. The specific research questions formulated are the following:

- **RQ1.** *How does ASCI work in the context of cross-project bug prediction when compared to existing ensemble techniques?*
- **RQ2.** *To what extent can local learning improve the performances of ASCI?*

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	InterCafe	1	27	11	4	15%
4	Ivy	2.0	352	87	40	11%
5	pBeans	2	51	15	10	20%
6	Serapion	1	45	10	9	20%
7	Synapse	1.2	256	53	86	34%
8	SystemDataManagement	1	65	15	9	14%
9	TermoProjekt	1	42	8	13	31%
10	Tomcat	6	858	300	77	9%

The first research question (**RQ1**) aims at assessing the performances of our approach in the context of cross-project bug prediction, while **RQ2** is focused on the combination between local learning and the adaptive selection of classifiers.

3.1 Context Selection and Data Preprocessing

The *context* of the study was composed of the 10 software systems shown in Table 1, which 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. Starting from the Promise dataset [26], we (i) filtered out systems having more than 50% of buggy classes as recommended by Tantithamthavorn et al. [40] to ensure data robustness, and (ii) randomly selected 10 projects from the remaining ones. It is important to highlight that the considered dataset already contained both independent and dependent variables used to build the bug prediction models. Indeed, for each class of the considered systems the independent variables were represented by LOC and Chidamber and Kemerer metrics [8], while the dependent variable was represented by a boolean value indicating the bugginess of each class.

Once we selected the dataset, we applied the following data preprocessing activities:

(1) **Data Cleaning**. As shown by Shepperd et al. [38], the Promise repository might contain noise and/or erroneous entries that possibly bias the results of bug prediction models. To deal with this issue, we applied the 13 corrections they proposed to remove instances with conflicting values or presenting missing values. From the initial dataset composed of 5,422 instances, we removed ≃1% of them. Thus the final dataset was composed of 5,361 instances.

(2) **Data Normalization**. A second element that could badly affect the performance of the prediction models is related to the different levels of design-complexity metrics [6]. To overcome this issue, we applied the normalization filter implemented in Weka [17] which linearly normalizes the data in the [0,1] interval. It is important to note that the choice of this normalization technique came from the results provided by Nam et al. [29] and Herbold et al. [22], who showed that such a technique represents the best one for this task.

(3) **Feature Selection**. Highly correlated independent variables can negatively affect the capabilities of bug prediction models [32]. To avoid this issue, we applied the Correlation-based Feature Selection (CFS) approach [18]. This method uses correlation measures and a heuristic search strategy to identify a subset of actually relevant features for a model. It is worth noting that we applied CFS for each training set obtained from the *Leave-One-Out* cross validation process, described later in Section 3.2. Thus, we firstly combined the training instances belonging to different software systems and then we applied the feature selection algorithm, as recommended by Hall et al. [19].

(4) **Data Balancing**. Bennin et al. [1] 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 the *Synthetic Minority Over-sampling TEchnique*, i.e. SMOTE [7] to ensure a similar proportion of buggy and non-buggy classes in the training sets.

It is worth noting that the order of the preprocessing steps have been guided by the framework proposed by Song et al. [39], who suggested an ideal sequence of operations to perform before training a bug prediction model.

3.2 RQ1 - Evaluating ASCI in Cross-Project Bug Prediction

To answer **RQ1** we ran ASCI over the 10 preprocessed datasets selected. As done in our previous work [13], the selected base learners used to configure ASCI were Naive Bayes, Logistic Regression, Multi Layer Perceptron, J48, Radial Basis Function Network.

A key decision in this context was the selection of an appropriate validation strategy [41]. We opted for the *Leave-One-Out Cross-Validation* [37]. 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 10 times, allowing each of the 10 systems to be the test set exactly once [37]. 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. [41].

For the evaluation of the performances obtained by ASCI, we employed widely-adopted metrics such as (i) accuracy, (ii) F-measure, i.e. the harmonic mean of precision and recall, and (iii) the Area Under the Curve (AUC), which quantifies the overall ability of a model to discriminate between buggy and non-buggy classes.

The performances achieved by our approach were firstly compared with those obtained by the model relying on the Naive 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, Naive 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 Naive Bayes led to the best results in terms of F-Measure. For this reason, we considered such a classifier as our baseline. In the second place, we benchmarked ASCI with the Validation and Voting (VV) ensemble classifier [43], which predicts the bug-proneness of a class based on the majority of "votes" of the base classifiers. Also in this case, the base learners were Naive Bayes, Logistic Regression, Multi Layer Perceptron, J48, and Radial Basis Function Network.

The choice of using Validation and Voting as baseline was driven by the findings provided by Zhang *et al.* [47], which demonstrated that the VV method is able to outperform other ensemble classifiers in the context of cross-project [47] bug prediction.

We are aware of the possible impact of classifiers' configuration on the ability of finding bugs [42], however the identification of the ideal settings in the parameter space of a single classification technique would have been prohibitively expensive [2]. For this reason, we applied the classifiers using their default configuration.

Due to space limitations, we report and discuss the boxplots of the distributions of the accuracy, the F-Measure, and the AUC achieved by the single classifiers independently on the 21 considered systems. A complete report of the results is available in our online appendix[1].

3.3 RQ2 - Combination of Local-Project Strategy and Ensemble Techniques

In the context of this research question, we had to build local bug prediction models. To this aim, we exploited the Expectation Maximization (EM) clustering algorithm proposed by Dempster et al. [11]. Its choice 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. [3] and Menzies et al. [25]. Lastly, previous work [22] showed that the performance achieved by EM are close to those obtained by the algorithm originally proposed by Menzies et al. [25]. We relied on the implementation of the algorithm available in the Weka toolkit [17].

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* [37] where the test sets was represented by the data of P_i. Unlike traditional leave-one-out cross validation, we created a bug prediction model for each of the clusters. During the testing phase, for each class in P_i we used the model trained on the cluster the class is closed to.

As done for **RQ1** we we compared ASCI with Naive Bayes and Validation and Voting using accuracy, F-Measure, and AUC to evaluate the performances of our technique. The fine-grained results are available in our online appendix[1]. To measure the extent to which local learning has an effect on cross-project performances, we also compared local models with global models defined in the context of **RQ1**.

4 ANALYSIS OF THE RESULTS

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

[1] https://figshare.com/articles/Ensemble_of_Classifiers_for_Cross-Project_Bug_Prediction_A_Replicated_Study/5555002

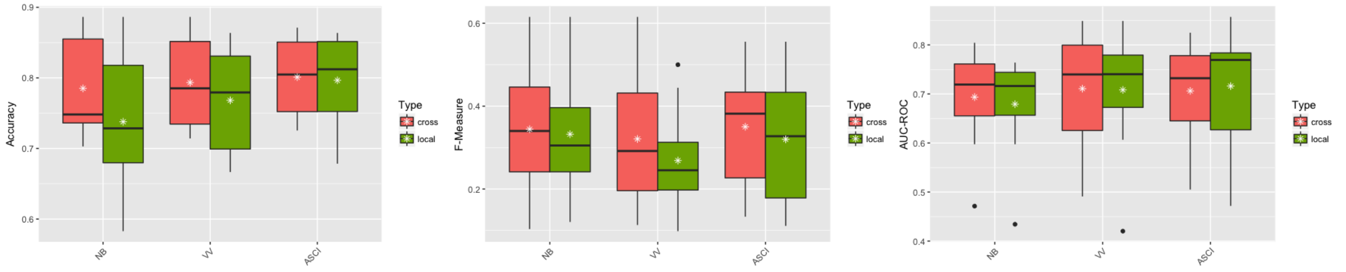

Figure 2: Boxplots of the Accuracy, F-Measure, and AUC-ROC achieved by NB, VV, and ASCI.

4.1 RQ1 - Evaluation of Ensemble Techniques when Adopted for Cross-Project Bug Prediction

Figure 2 depicts the box plots of the Accuracy, F-Measure and AUC-ROC achieved on the 10 software systems in our dataset by the experimented cross-project bug prediction models (white asterisks highlight the means). In red we report the performances in the global setting, while in green the ones of local learning.

First of all, we found that in some cases the models based on Naive Bayes exhibit better performances than those based on the ensemble classifiers. Thus, we can claim that the models based on ensemble classifiers do not always provide improvements with respect to a well selected stand-alone model. Indeed, as for ASCI, our results show how its performances are better than Naive Bayes (e.g. median Accuracy +6%, median F-Measure +4%, and median AUC-ROC +2%). At the same time, it shows better performance with respect to Validation and Voting in terms of Accuracy, F-Measure (e.g. median Accuracy +2%, median F-Measure +8%, ≊ median AUC-ROC). Thus, the application of our technique in a cross-project setting is able to provide improvements in the prediction of bugs with respect to the models based on the well-known **Naive Bayes** and Validation and Voting ensemble technique.

As a more general observation, it is important to note that the performances of all the experimented cross-project models are quite low—on average they do not exceed 40% in terms of F-Measure. On the one hand, all the experimented models solely relied on code metrics as independent variables. As widely shown in literature [9, 12, 28] a combination of predictors of different natures (e.g. process metrics) has an important effect on the overall performances 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.

Summary for RQ1. None of the experimented cross-project models is able to exceed 40% of F-Measure (on average), meaning that the problem of identifying buggy classes using external sources of information is still far from being solved. However, the models relying on ASCI are able to provide benefits with respect to stand-alone classifiers and the Validation and Voting ensemble technique.

4.2 RQ2: Evaluation of Ensemble Techniques when Adopted for Local Cross-Project Bug Prediction

On the basis of the results achieved in **RQ1**, we verified whether the application of local learning—that was suggested as a promising way to reduce data heterogeneity—could improve the performances of ASCI. Figures 2 depicts the box plots reporting Accuracy, F-Measure, and AUC-ROC achieved on the 10 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 Naive Bayes. To ease the comparison with the results of **RQ1**, 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 performances with respect to global models*. In particular, in case of Naive Bayes, we can notice a decrease of the performances of 4% in terms of median F-Measure. It is worth noting that combining local learning with Naive Bayes does not achieve better performances than those obtained using ensemble techniques without local learning.

Analyzing the differences between the global and local versions of the experimented ensemble methods, we observe that *researchers should be careful when applying local bug prediction*. Indeed, looking at the median F-Measure, we can observe that the Validation and Voting model is not able to take full advantage of the local learning, achieving a decrease of 5% of in terms of median F-Measure. A likely motivation for such result comes from the characteristics of the algorithm: as shown in previous research [13, 35], 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 performances of techniques which combine the output of different classifiers. When applying Validation and Voting locally, we observed that specific classifiers act much better on some specific clusters than other machine learners, meaning that very few of them can correctly classify the bug-proneness of code entities. As a consequence, the voting is often not useful, leading to a decreasing of the overall performances of local bug prediction for some systems. This result confirm the findings of Herbold et al. [22] that find that local models have a small influence on the results of cross-project bug prediction models.

Looking at ASCI, we found that this classifier is able to exploit some of the advantages provided by the lower heterogeneity of data provided by local models. Indeed, despite there is a decrease in terms of median F-Measure of 5%, there is an increase of 2% in terms of Accuracy and 4% in terms of AUC-ROC. It is worth noting that the AUC-ROC measures gives an indication on the robustness of bug prediction models [20] (i.e. how well the classifier separates the binary classes), while the F-Measure indicates the accuracy with which the predictions are done. In other words, they are complementary metrics that capture robustness and accuracy, respectively. Thus, we can affirm that local learning is able to improve the robustness of a technique for the adaptive selection of classifiers in the context of cross-project defect prediction.

Summary for RQ2. Researchers should be careful when applying local learning. Indeed, in most cases local learning is not able to improve the performances of bug prediction models. Despite this, the combination of local learning and ASCI is able to improve the models performances in terms of robustness.

4.3 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 come from the Promise repository [26], which is widely considered reliable and, indeed, has been also used in several previous work in the field of bug prediction [16, 24, 33, 34, 47]. 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. [38], 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 [40].

As for the experimented prediction models, we exploited the implementation provided by the Weka framework [17], 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 [2]. As future goal, we plan to further analyze the impact of parameters' configuration to our findings.

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 *Leave-One-Out Cross-Validation* methodology [37]. This choice was driven by results recently reported that showed that such validation technique is among the ones that are more stable and reliable [41].

To ensure that the results would have not been biased by confounding effects due to data unbalance [7] or highly correlated independent variables [14], we adopted formal procedures aimed at (i) over-sampling the training sets [7] and (ii) removing non-relevant independent variables through feature selection [18].

As for the evaluation of the performances of the experimented models, we used a widely used set of metrics (i.e. accuracy, F-measure, and AUC-ROC) to evaluate the performances of bug prediction classifiers [5, 16, 24, 34].

Threats to external validity. These are threats concerned with the generalizability of the findings. We analyzed 10 different software projects coming from different application domains and having different characteristics (i.e. developers, size, number of components, etc.). We are aware that this is a small set of systems (especially as training set) and that this could have bad influenced the results achieved in terms of F-Measure. Of course, we cannot claim the generalizability with respect to industrial environments, however the replication of the study on industrial projects is part of our future research agenda.

5 CONCLUSION

In this paper, we aimed at evaluating the usage of ASCI for *global* and *local* cross-project bug prediction. Specifically, we compared its performances with those achieved by the best stand-alone classifier, i.e. Naive Bayes, and the Validation and Voting ensemble technique, which has been shown to be the most reliable in previous work [47]. The study was conducted on a set of 10 software projects from the Promise dataset, where we mitigated possible threats to validity applying some precautions with respect to the quality of data used.

We found that our approach improves the performances of bug prediction models when adopted in a global scenario: in this case, ASCI tends to work better than the other considered models, thus highlighting how a combination of stand-alone classifiers generally improve bug prediction performances. When turning our attention on the local learning scenario, our results show that the use of this technique is not able in most cases to improve the performances achieved by the models. Despite this, the combination of ASCI with local learning is able to produce better models in terms of robustness (e.g. AUC-ROC), while there are conflicting results in terms of accuracy.

The observations above represent the main starting point for our future research agenda. We firstly plan to replicate the study on a larger set of systems, using a richer set of independent variables, using a large number of ensemble classifiers, and investigating the impact of classifiers configuration on our findings. In the second place, we aim at studying what are the characteristics that make the use of ensemble techniques useful or not useful, so that we can build recommenders able to automatically suggest to developers in which contexts it is really worth to use ensemble methods.

ACKNOWLEDGMENTS

Di Nucci is funded by the Belgian Innoviris TeamUp project INTiMALS. Palomba acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

REFERENCES

[1] 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*, PP(99):1–1, 2017.

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

[3] N. Bettenburg, M. Nagappan, and A. E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 60–69. IEEE, 2012.

[4] 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*, pages 119–128. Springer, 2008.

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

[6] A. E. Camargo Cruz and K. Ochimizu. 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*, pages 460–463. IEEE Computer Society, 2009.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[8] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, Jun 1994.

[9] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.

[10] D. J. Dean, H. Nguyen, and X. Gu. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th International Conference on Autonomic Computing*, pages 191–200. ACM, 2012.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[12] 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*, PP(99):1–1, 2017.

[13] 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*, 1(3):202–212, 2017.

[14] T. Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, 27(3):326–327, Sept. 1995.

[15] 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*, 42(4):695–703, 2014.

[16] 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*, pages 789–800. IEEE, 2015.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations Newsletter.*, 11(1), 2009.

[18] M. A. Hall. Correlation-based feature selection for machine learning. Technical report, 1998.

[19] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[20] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[21] S. Herbold, A. Trautsch, and J. Grabowski. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2017.

[22] S. Herbold, A. Trautsch, and J. Grabowski. Global vs. local models for cross-project defect prediction. *Empirical Software Engineering*, 22(4):1866–1902, 2017.

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

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

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

[26] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

[27] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[28] 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*, pages 309–311. ACM, 2008.

[29] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391. IEEE Press, 2013.

[30] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

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

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

[33] 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.

[34] 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*, pages 164–173. IEEE, 2014.

[35] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo. 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*, page 46. ACM, 2016.

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

[37] C. Sammut, editor. *Leave-One-Out Cross-Validation*, pages 600–601. Springer US, Boston, MA, 2010.

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

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

[40] 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*, pages 321–332, 2016.

[41] 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*, 43(1):1–18, 2017.

[42] 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*, 39(10):1427–1443, 2013.

[43] 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*, pages 318–320. ACM, 2008.

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

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

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

[47] 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*, volume 2, pages 264–269. IEEE, 2015.

[48] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 91–100. ACM, 2009.