Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Global vs. Local Models for Cross-project Defect Prediction

A Replication Study

Steffen Herbold · Alexander Trautsch · Jens Grabowski

Received: date / Accepted: date

Abstract Although researchers invested significant effort, the performance of defect prediction in a cross-project setting, i.e., with data that does not come from the same project, is still unsatisfactory. A recent proposal for the improvement of defect prediction is using local models. With local models, the available data is first clustered into homogeneous regions and afterwards separate classifiers are trained for each homogeneous region. Since the main problem of cross-project defect prediction is data heterogeneity, the idea of local models is promising. Therefore, we perform a conceptual replication of the previous studies on local models with a focus on cross-project defect prediction. In a large case study, we evaluate the performance of local models and investigate their advantages and drawbacks for cross-project predictions. To this aim, we also compare the performance with a global model and a transfer learning technique designed for cross-project defect predictions. Our findings show that local models make only a minor difference in comparison to global models and transfer learning for cross-project defect prediction. While these results are negative, they provide valuable knowledge about the limitations of local models and increase the validity of previously gained research results.

Keywords defect prediction · cross-project · local models

1 Introduction

The prediction of software defects with data that is not from the past of the target project is an ongoing research issue in recent years. Within this article, we differentiate between *within-project defect prediction*, where data from

Steffen Herbold, Alexander Trautsch, Jens Grabowski
Institute of Computer Science
University of Göttingen, Germany
E-mail: {herbold,graboswki}@cs.uni-goettingen.de, alexander.trautsch@stud.uni-goettingen.de

the same project is used for prediction, *mixed cross-project defect prediction*, where data from old revisions of the same project is used together with data from other projects, and *strict cross-project defect prediction*, where only data from other projects is used. Since the first large study on this cross-project defect prediction by Zimmermann et al (2009) showed that the expected success, if you just select another project at random to predict the defects, is very low[1], a lot of research went into the improvement of this, e.g., through data selection (Turhan et al, 2009; He et al, 2012, 2013; Herbold, 2013), data transformation (Watanabe et al, 2008; Camargo Cruz and Ochimizu, 2009), and weighting of the training data (Ma et al, 2012; Zhang et al, 2014). While all of the above approaches have shown promise and improved the performance of cross-project defect prediction, the achieved performance is still not on a level that is sufficient for practice, especially the precision is still problematic. As Tan et al (2015) point out, for defect predictions models to be accepted, the developers need to be convinced of the results, partially by presenting the precision of predictions. However, Rahman et al (2012) point out that the overall performance might not be that bad when using cost saving instead of traditional machine learning metrics for the evaluation of the performance.

Within this paper, we take a detailed look at local models for cross-project defect prediction. Local models use all of the available data for training, but internally separate the data into local groups and train a different model for each group. This is in contrast to global models, that train a single classifier based on all training data.

Local models were introduced by Menzies et al (2011, 2013) for effort and defect prediction and are based on the idea that there might be homogeneous regions in otherwise heterogeneous data. If this is true, it would be better to first detect the homogeneous regions, and then train a local classifier for each of these regions separately in a second step. In comparison, the traditional approach is to train a global classifier on all training data. In their study, Menzies et al showed that the *precision* of local models is higher than that of global models. In a replication study (Bettenburg et al, 2012) and its extension (Bettenburg et al, 2014), Bettenburg et al observed similar advantages for the *precision* when it comes to gaining insights in the data used for training. However, they also observed that these advantages may be due to overfitting on the local region which could mean that global models perform better for predictions. Their results are, therefore, partially contradicting the first study by Menzies et al. In a third study by Scanniello et al (2013), the effect of local models not on the *precision*, but rather on the *error* of the models is investigated. In their study, they determine that the error of local models is indeed lower than that of global models.

Due to the findings by the above studies, we think that local models should be investigated in detail for cross-project defect prediction, i.e., defect predictions where only data from a foreign project context is used for the prediction.

[1] With the data used in the study and the success criterion of having both *recall* and *precision* of at least 0.75 they achieved a success rate of about 3%.

One of the major problems of cross-project defect prediction is the heterogeneity of the data which leads to a rather poor *precision* if it is not treated (Turhan et al, 2009). Since local models are designed to deal with heterogeneity, they can potentially mitigate this problem and, thereby, increase prediction performances of cross-project defect prediction.

However, when it comes to the application of local models to cross-project defect prediction, we identified multiple aspects that have not yet been sufficiently explored by the previous studies on local models. None of the previous studies focuses on local models for strict cross-project defect prediction. Only the studies by Menzies et al (2011, 2013) consider a setting with predictions from multiple projects, but it is not a strict cross-project setting, as we discuss in the related work (see Section 2). The other studies do not evaluate cross-project effects. Moreover, none of the previous studies compare local models with other transfer learning techniques for cross-project defect prediction. Transfer learning is a learning task where knowledge about the target domain is used "to improve the learning of the target predictive function" Pan and Yang (2010). To this aim, the model may transform the data (Watanabe et al, 2008; Nam et al, 2013), work on a subset of the training data (Turhan et al, 2009; He et al, 2013; Herbold, 2013), weight training instances differently (Ma et al, 2012) or otherwise use knowledge about the target project to improve the prediction result. These issues are a threat to the validity in the context of cross-project predictions.

Additionally, there is a general lack of external validity regarding the studies on local models. Two of the previous studies only look at rather small data sets, with seven (Menzies et al, 2011, 2013), three (Bettenburg et al, 2012, 2014) projects for defect prediction, respectively. Only the study by Scanniello et al (2013) uses a larger number with 29 projects. However, the latter study focuses rather on the impact of using dependencies and differs also significantly in the creation of the local models.

Siegmund et al (2015) recently determined that there seems to be a general lack of replication studies that enhance the internal and external validity of previous reseach results, even though the value of replication studies has already been pointed out years ago (e.g., Shull et al, 2008; Kitchenham, 2008). Therefore, the aim of this article is to improve the body of knowledge on local models in the context of cross-project defect prediction and, moreover, increase the general external validity of the results on local models through a replication study. Our replication study is an independent conceptual replication, according to the classification of replication study types by Shull et al (2008). We used the guidelines for replication studies proposed by Carver (2010) as foundation for the reporting of our study.

Through our study, we try to answer the following four research questions:

RQ1: Is there a significant advantage in using local models instead of global models for cross-project defect prediction?

RQ2: Is there a significant advantage of local models over transfer learning techniques for cross-project defect prediction?

RQ3: Is there a significant advantage of normalizing data before the application of local models to reduce scale effects?

RQ4: Is there a significant impact on the peformance of using strict cross-project data in comparison to mixed conditions?

To answer our research questions, we analyze the effects of local models on three different defect prediction data sets with a total of 79 distinct software versions. Within our study, we apply two local models and compare the performance with a global classifier that simply uses all available training data and a transfer learning approach proposed by Turhan et al (2009) as that is often used for baseline comparisons in cross-project defect prediction (e.g., He et al, 2013; Peters et al, 2015). Moreover, we evaluate the effects of normalization (Han et al, 2011) on the local models to see if it helps to first remove scale effects from the metrics before applying the clustering of the local models. Finally, we want to evaluate the difference between a strict cross-project setting and a mixed cross-project setting.

The remainder of this paper is structured as follows. In Section 2, we discuss related work on the usage of local models for defect prediction. Afterwards, in Section 3, we take a general look at local models and their implementation. Then, we proceed to the main contribution of our work, a case study on local models in cross-project defect prediction in Section 4. In Section 5, we discuss the results of our replication study and compare them with the previous results. Then, we summarize the lessons we learned from our case study and answer our research questions based on our findings in Section 6. Afterwards, we discuss the threats to the validity of our results in Section 7. Finally, we conclude the paper and give an outlook on future work in Section 8.

2 Related Work

Within this section, we discuss related work. First, we discuss previous studies on local models in detail. Then, we discuss related work on cross-project defect prediction in general.

2.1 Local Models

The usage of local models in software defect prediction is a rather new approach. To the best of our knowledge, there are only three studies[2] up to date, performed by Menzies et al (2011, 2013), Bettenburg et al (2012, 2014), and Scanniello et al (2013). In the following, we discuss the three studies and emphasize the differences to our work. Moreover, we summarize the case studies performed in the related work, including the key differences to our study in Table 1. For the structure of this table, we took pattern from the guidelines

[2] The studies by Menzies et al and Bettenburg et al were first published in an initial version at a conference and then in greater detail in a journal publication, leading to five publications for the three studies.

for replication studies by Carver (2010), which defines the parameters that should be compared.

The first study was performed by Menzies et al (2011) and extended in Menzies et al (2013). In their paper, they first proposed the idea of using local models. To this aim, they propose to apply the WHERE algorithm to the data in order to create clusters. The WHERE algorithm first applies the Fastmap algorithm (Faloutsos and Lin, 1995), to map the input data to two dimensions. Afterwards, they apply QuadTree clustering (Schikuta and Schikuta, 1993) with an additional post-procedure to join small clusters to determine local regions where the data is homogeneous. They then analyze the local clusters with the WHICH algorithm (Huang et al, 2010) in order to extract information. The authors noted an increase in *precision* of the local models in comparison to the global models.

In their study, they considered seven data sets with defect data, and, additionally, two data sets with effort data. They combine all defect data into a single set which is then used as input for the WHERE algorithm. As the authors state, "each cluster may now contain examples from multiple sources" (Menzies et al, 2013). Due to the combination of data from multiple sources, cross-project effects should be present in their evaluations. However, since data from all projects may be within the clusters used for training data, this should be seen as a mixture of cross-project and within-project prediction and not strict cross-project predictions. In comparison, in our study we look at the effect of local models in a strict cross-project defect prediction setting, where the data from the target project cannot be part of the training of a predictor. Additionally, we investigate a mixed cross-project setting, in which we allow data from old versions of the same projects. Note, that this mixed setting is different from the mixture between cross-project and within-project prediction by Menzies at al., where even data from the same version could be used for predictions. Moreover, our study uses data of 79 software versions. Furthermore, we do not only use the WHERE algorithm for clustering, but also the EM clustering algorithm (Dempster et al, 1977) and investigate the effects of normalization on the local models.

Bettenburg et al (2012) performed detailed studies regarding the internal validity of local models that was further extended in Bettenburg et al (2014). In comparison to Menzies et al., they did not use the WHERE algorithm for clustering and WHICH algorithm for predictions. Instead, they used the MCLUST (Fraley and Raftery, 1999) algorithm for clustering and linear regression models as well as Multivariate Adaptive Regression Splines (MARS) models for predictions. Moreover, they applied the k-means clustering algorithm with different values for k and compared the results to those achieved with MCLUST.

In their study, they determined that local models are better than global models when used for gaining insights about the data under consideration as they better fit the training data. They observe that this may lead to problems when it comes to predictions. This effect is demonstrated by applying the k-means algorithm with different values of k. They show that the prediction

Study	Menzies et al (2011, 2013)
Research questions	How good are global/local models suited to generate lessons to minimize efforts and defects?
Data	Seven defect prediction data sets and two effort prediction data sets.
Design	All data of the same type joined into one set, comparison of the results between global and local models. Usage of the WHICH algorithm to infer rules.
Result summary	Local models better suited for learning lessons and inferred rules are more precise.
Differences in our study	Focus on prediction models instead of rule inference; strict focus on the cross-project setting; different data; *precision*, *recall*, *F-measure* and *error* instead of fit of rules.
Study	Bettenburg et al (2012, 2014)
Research questions	Is there an advantage of using local models over global models, with respect to goodness of fit? Is there an advantage of using local models over global models, with respect to prediction performance? What is the role of clustering of datasets when building local models, as compared to randomly partitioning the data into smaller chunks? What is the impact of choice of clustering algorithm and parameters on the performance of the resulting local models? For the same clustering method, modeling techniques, and predicted outcome, how do different software engineering metrics respond to local modeling? What are the considerations in the use of local models over global models for practitioners?
Data	Three defect prediction data sets and two effort prediction data sets.
Design	10x10 cross-validation for each data set; comparison of the results for different local models (impact of clustering) and between the local and global model.
Result summary	Local models have a better fit which leads to a better precision, but may not generalize well for predictions. The clustering has a big impact on the outcomes.
Differences in our study	Different focus (cross-project defect prediction); different data; *precision*, *recall*, *F-measure* and *error*, instead of model fitting metrics.
Study	Scanniello et al (2013)
Research Questions	Does the new clustering-based approach improve fault prediction results compared with the baseline? (By new clustering approach the authors refer to the approach based on the BorderFlow clustering algorithm.)
Data	29 defect prediction data sets extendend with information about the architecture of the software.
Design	Leave-one-out cross-validation for each of the 29 data sets to evaluate the difference of the *error* between the local model and the global model.
Result summary	The *error* of the local model is lower than of the global model.
Differences in our study	Different focus (cross-project defect prediction); different data; our study does not take architecture into account; evaluation of *precision*, *recall*, and *F-measure* in addition to the *error*.

Table 1: Summary of the case studies performed in the related work including the key differences to our study.

performance decreases with increasing k. From this, they conclude that careful tuning of the local model is required for predictions and suggest the usage of a clustering algorithm that does this automatically.

The study performed by Bettenburg et al (2012, 2014) was rather small in terms of the data considered. They only used five software versions for their evaluations: three with defect data and two with effort data. They note that this is a major threat to the external validity of their results. Moreover, they also only considered the within project-context and it remains unclear how their findings translate to the cross-project context. As we stated above, we use data from 79 software versions, explore the cross-project application and apply with the WHERE and EM clustering two clustering algorithms that both tune themselves automatically. The EM clustering algorithm is somewhat similar to both MCLUST and k-means and could be seen as a middleground between the two clustering algorithms. Additionally, we investigate the impact of normalization on the local models.

Scanniello et al (2013) performed a third study using local models. Within their work, they use a graph representation of the class dependencies in combination with the BorderFlow clustering algorithm (Ngomo, 2009) to determine local regions. Due to the detailed knowledge about class dependencies they also require access to the source code in order to create the clustering. For predictions, they apply step-wise linear regression, a model similar to Bettenburg et al (2012). Within their study, they observe that the *error* of the local models is lower than that of global models. A strong point of their work is the usage of data from 29 software versions in their study, which leads to a higher external validaty of the results.

In comparison to our study, Scanniello et al. only evaluate the *error* of the predictions. The *error* alone is problematic as a measure for defect prediction, because the data sets are usually imbalanced and only optimizing for the error often favors trivial classification models (Rahman et al, 2012). Therefore, we take additional performance metrics into account, i.e., *recall*, *precision*, and *F-measure*. Furthermore, the clustering approach applied by Scanniello et al. requires access to the source code in order to create the dependencies between classes. In comparison, the clustering approaches we apply can work with any numerical data. While the amount of data used is already quite large with 29 data sets, they all come from the same source, whereas we use three different data sources in our case study.

2.2 Cross-project Defect Prediction

Because the replication study targets local models in the context of cross-project defect prediction, we also briefly discuss the related work from that area. The body of work is mostly focused on transfer learning approaches, i.e., approaches that try to manipulate the training based on the target product in order to improve the result of the defect prediction. Watanabe et al (2008) proposed to use a standardization technique based on the mean value

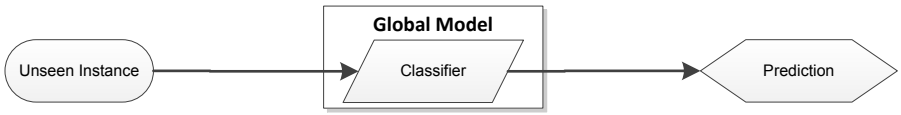
of the target product. A similar approach was proposed by Camargo Cruz and Ochimizu (2009), but based on the median and power transformations. Ma et al (2012) use the idea of data gravitation to weigth the training data based on the similarity to the target product. Nam et al (2013) investitagated min-max standardization as well as Z-score standardization. Zhang et al (2014, 2015) propose a transformation of the data based on clusters created using the project context. A different venue of investigations is to only use a subset of the training data. Turhan et al (2009) propose to use the k-nearest neighbor algorithm to select software entities similar to the entities in the target product. Jureczko and Madeyski (2010) proposed an approach based on self-organizig maps to select appropriate software products that can be used as training data. Similarly, Herbold (2013) proposed the k-nearest neighbor algorithm and the EM clustering algorithm as candidates for the selection of appropriate software products based on distributional characteristics of software metrics. He et al (2013) propose to select appropriate software products based on the separability in comparison to the target product. Moreover, He et al (2013) propose to select a subset of the attributes used for classification based on the same strategy, as well as to use product wise bagging. Another approach for relevancy filtering based on the DBSCAN algorithm was proposed by Kawata et al (2015) suggest to use the DBSCAN algorithm for relevancy filtering. Amasaki et al (2015) also suggest to select a subset of the attributes based on synonym pruning (Kocaguneli et al, 2013). For all of the above approaches, the result of the training of a defect prediction model, depends on the target product. This is a contrast to local models, which depend only on the training data and are not adopted depending on the target product.

There are also proposals in the literature regarding the treatment of data in general. Peters et al investigate the how data for cross-project predictions can be shared privatly, in order to allow companies to share the proprietary data. To this aim, they proposed CLIFF+MORPH (Peters et al, 2013) and LACE2 (Peters et al, 2015). He et al (2014) and Nam and Kim (2015) started a new venue of investagation where they try to combine data from different sources with different metrics for the usage of cross-project predictions.

There are also some general studies on cross-project defect prediction. Zimmermann et al (2009) determined in a study with over 622 pair-wise cross-project predictions, that only 21 results achieved their desired quality[3]. From their results, they built a decision tree, that demonstrate that the results can be greatly improved, if the correct product is selected for training in comparison to picking training data at random. A similar study was performed by He et al (2012), wo evaluated not pair-wise predictions, but combinations of one, two, or three projects as training data. Through their study, they demonstrated that distributional characteristics of metrics can be used as indicator for the selection of training data. Premraj and Herzig (2011) evaluated the usage of network metrics for cross-project defect prediction and found that the results are not much improved.

[3] *recall*, *precision*, and *accuracy* all at least 0.75.

(a) Training of a global model. The data is directly used to create a classifier.



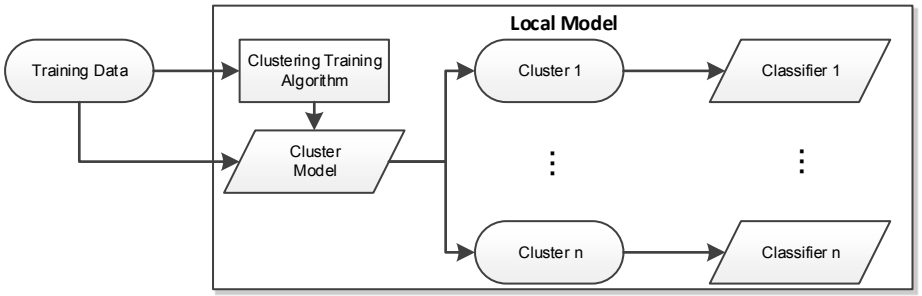
(b) Prediction with a global model. The single trained clusterer is directly used to make the prediction.

Fig. 1: Training and prediction with global models.

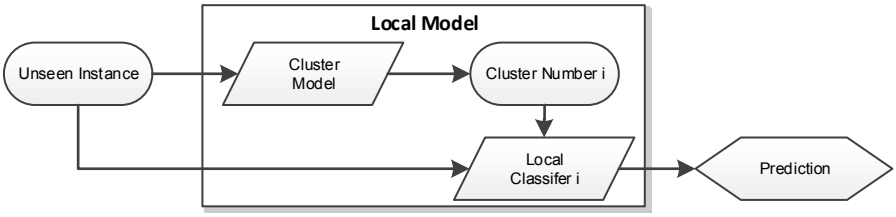
3 Local Models

The general idea of local models is quite simple: find homogeneous regions and train a classifier for each of these regions. Figure 1 visualizes how the training (Figure 1(a)) and prediction (Figure 1(b)) with a global model looks like. Figure 2 shows how the training (Figure 2(a)) and prediction (Figure 2(b)) with a local model look like in comparison. For the global model, the training data is directly used to create a classifier, e.g., a Support Vector Machine (SVM), a linear regression model, or a decision tree. This classifier is then used to predict unseen data. In comparison, the first step for the training of a local model is the creation of a cluster model with a clustering training algorithm. The clustering training algorithm gets the training data without the classification as input. From this, it creates a model that describes the clusters within the data. The cluster model takes as input also the training data and separates it into clusters according to the model. Then, for each of the clusters, a local classifier is trained. For prediction, this means that first the cluster model is used to determine to which cluster an unseen instance belongs. Then, the local classifier belonging to the identified cluster is used for the prediction on this instance. In this section, we summarize the steps for training and prediction with local classifiers and note for each step possible problems that need to be taken care of.

The main design decision of a local model is the method for determining the homogeneous regions within the data. For this task, researchers have used unsupervised clustering algorithms. In the past, the WHERE, MCLUST, k-means, and BorderFlow algorithms were used (see Section 2). However, one could potentially use also one of the many other clustering methods. In their survey on clustering algorithms, Xu and Wunsch (2005) list over 30 different methods for clustering. Bettenburg et al (2014) observed that the clustering algorithm has an impact on the results achieved. The results from Bettenburg et al suggest that a clustering algorithm that automatically determines the clustering parameters, most importantly the number of clusters, should be



(a) Training of a local model. The clusterer and the prediction models per cluster make up the local model that can then be used for predictions.



(b) Prediction with a local model. First, the clusterer is used to determine which concrete of the n local classifiers is used by determining to which cluster $i \in 1 \ldots n$ the unseen instance belongs. Then, the prediction is made with the local classifier i.

Fig. 2: Training and prediction with local models.

used. In our study, we decided to use the WHERE algorithm and the previously not considered EM clustering algorithm (Dempster et al, 1977).

We choose the WHERE algorithm due to its efficiency and because it has shown strong results in a previous study (Menzies et al, 2011, 2013). However, we could not simply implement the algorithm as it was described by Menzies et al, because we consider the cross-project setting. In this setting, it must be possible to classify data from unknown sources, which may cause problems, e.g., due to different value ranges. The QuadTree algorithm, which is internally used by the WHERE algorithm, can only deal with data in the range that was used during the training of the local model. Hence, we adapted the QuadTree algorithm such that values outside the previously observed range are interpreted as if they were at the closest boundary of the data seen during the training. For example, if an instance in the test data has a value for some metric $m_{test} = 1000$, but the highest observed value of of the metric m in the training data is $\max_{trainingdata}(m) = 100$, the original QuadTree algorithm cannot classify the instance. To enable the QuadTree algorithm to cluster such data, we treat the values out of range as if they have the maximal, respectively minimal value, i.e., in our example we treat m_{test} as if it would have the value 100, i.e., the highest value within the range of the training data.

As second algorithm, we choose the EM clustering algorithm. We choose this algorithm for multiple reasons. First of all, it can determine the number

of clusters through an internal cross-validation loop automatically. Moreover, it is a good complement to the WHERE algorithm. The WHERE algorithm simplifies the data structurally by reducing it to two dimensions, whereas the EM algorithm clusters on the original dimensions. Furthermore, the FastMap routine of the WHERE algorithm introduces a random component into the clustering, whereas the EM algorithm is deterministic. However, these advantages of the EM algorithm come at the cost of a much higher runtime. Finally, the EM algorithm is similar to both the MCLUST algorithm and the k-means algorithm used by Bettenburg et al (2012, 2014) and can be seen as the middleground between MCLUST and k-means.

Due to the automatic determination of the number of clusters by the EM algorithm, we encountered problems due to very small clusters. Outliers in the data were often assigned to their own cluster, which led to clusters with single instances. The WHERE algorithm handles this problem naturally by merging small clusters. We cannot do this with the EM algorithm. Instead, we have to repeat the clustering, with fewer clusters allowed until all clusters have a sufficient size. We defined two criteria for sufficient size:

- at least five instances within each cluster; and
- at least one instance of each class.

These requirements are choosen such that they enable the training of internal classification models for the cluster. For example, the random forest implementation of Weka Hall et al (2009), requires at least five training instances and breaks otherwise. Similarly, SVMs cannot work if they do not get at least one instance of each class, because they find nothing to separate. Hence, if one cluster does not meet the above criteria, we re-execute the clustering with exactly one cluster less than currently computed allowed and thereby force the algorithm to merge clusters. For example, if the EM algorithm calculates a result with six clusters, where one cluster has only one instance, we re-execute the clustering with a maximum number of five clusters allowed and check again if the clusters meet our criteria.

4 Case Study

The main contribution of this paper is a case study that evaluates the performance of local models for cross-project defect prediction. Our study follows the guidelines for conducting case studies proposed by Runeson and Höst (2009).

In order to answer our research questions, we formulate hypotheses, which we evaluate according to the results of our case study. Since we are performing a replication study, we have expectations on our results based on the previous findings. For **RQ1**, we have to determine if there is a difference between local and global models for cross-project defect prediction. To this aim, we analyze the effect of local models on the metrics *recall*, *precision*, *F-measure*, *error*, and *Area Under the ROC Curve (AUC)*. From the findings of the previous studies (Menzies et al, 2011, 2013; Bettenburg et al, 2012, 2014; Scanniello

et al, 2013), we believe that the precision of local models is higher than that of global models, but at the cost of a lower *recall*. Bettenburg et al (2014) observed that the gain in *precision* is for the most part offset by the loss in recall. Therefore, we believe that the overall performance, if these two metrics are used together, is similar for global and local models, which should be reflected by similar values in the *F-measure*. Moreover, due to the findings by Scanniello et al (2013), we believe that the *error* of local models is lower than that of global models. Due the tradeoff between *recall* and *precision*, we believe that local and global models perform similar in terms of AUC, same as *F-measure*. From these conjectures, we formulate five hypotheses:

H1: Local models have a lower *recall* than global models.
H2: Local models have a higher *precision* than global models.
H3: Local models and global models perform similar in terms of the *F-measure*.
H4: Local models have a lower *error* than global models.
H5: Local models and global models perform similar in terms of AUC.

Regarding **RQ2**, we have to evaluate the effects of local models in comparison to transfer learning techniques. We find no reason to believe that there will not also be a positive effect through local models. However, we have no evidence that they will actually perform better than transfer learning techniques when applied for cross-project defect prediction. To this aim, we compare the local models against the k-nearest neighbor approach for selecting training data by Turhan et al (2009). The approach works by selecting a subset of the available training data as follows: for each instance of the target product, only the k closest entities in the training data are kept, all other entities are removed. This approach is a transfer learner because it select the training data based on the target data, i.e., the knowledge about the target domain. We choose the k-nearest neighbor approach because it is often used for comparisons in cross-project defect prediction studies (e.g., He et al, 2013; Peters et al, 2015). Therefore, we selected this approach as baseline comparison for transfer learning techniques. The local models must be able to at least outperform this technique, in order to be able to compete with the more recently proposed state of the art in cross-project defect prediction. A complete comparison to all proposed transfer learning techniques is out of scope of this paper. Our aim is only to evaluate if local models can compete, in general, or if they are already worse than this baseline. To this aim, we formulate the following hypothesis.

H6: Local models perform better than the k-nearest neighbor model by Turhan et al (2009).

Regarding **RQ3**, we assume that normalizing the data, i.e., scaling the metric values to the interval [0,1] will help with the local models. In other experiments (e.g., Nam et al, 2013), positive effects due to normalization were observed, however, not in conjunction with local models. Moreover, in a previous work, we observed that normalization can help with the reduction of scale effects on distance-based clustering (Herbold, 2013). Therefore, we assume

that this may also be the case for local models and formulate the following hypothesis.

H7: Local models perform better if the data is normalized prior to the clustering.

Regarding **RQ4**, we consider different data configurations for training: strict cross-project data from the same project and mixed cross-project data where we allow at least older version from the same project. Then, we compare the performance of the classication models on the two data sets to evalute the difference between the strict and the mixed setting. Because data from previous versions should be similar to the target data, we believe that the overall performance of the classification models is better with the mixed data. To this aim, we formulate the following hypothesis.

H8: Mixed cross-project data yields better results than strict cross-project data.

In the following, we will describe the data we used, the experimental setup of our case study, and our evaluation criteria before we summarize our results.

4.1 Data

We apply our experiments to three data sets from different sources. The first data set we used was obtained from the tera-PROMISE[4] repository (Menzies et al, 2014) and contains defect data about 62 versions of 32 different open-source projects donated by Jureczko and Madeyski (2010). The data contains for each class of a version 20 static source code metrics, including the popular Chidamber and Kemerer (CK) metrics for object-oriented software (Chidamber and Kemerer, 1994), but also additional metrics, such as Lines Of Code (LOC). The complete list of metrics can be found in the original publication of the data set (Jureczko and Madeyski, 2010). Table 2 gives information about the number of classes of each project, including the number of defect-prone classes and their percentage. For simplicity, we will refer to this data set as JSTAT data from now on.

Table 2: Software products from the JSTAT data that are part of the study, including their total number of classes and the number of defect-prone classes.

Product	**#Classes**	**#Defect-prone**	**% Defect-prone**
ant 1.3	125	20	16%
ant 1.4	178	40	22%
ant 1.5	293	32	11%
ant 1.6	351	92	26%

Continued on next page

[4] The tera-PROMISE repository is the successor of the PROMISE repository, which was previously located at http://promisedata.googlecode.com.

Product	#Classes	#Defect-prone	% Defect-prone
ant 1.7	745	166	22%
arc	234	27	12%
berek	43	16	37%
camel 1.0	339	11	4%
camel 1.2	608	216	36%
camel 1.4	872	145	17%
camel 1.6	965	188	19%
ckjm	10	5	50%
e-learning	64	5	8%
forrest 0.7	29	5	17%
ivy 1.1	111	63	57%
ivy 1.4	241	16	7%
ivy 2.0	352	40	11%
jedit 3.2	272	90	33%
jedit 4.0	306	75	25%
jedit 4.1	312	79	25%
jedit 4.2	367	48	13%
jedit 4.3	492	11	2%
kalkulator	27	6	22%
log4j 1.0	135	34	25%
log4j 1.1	109	37	34%
log4j 1.2	205	189	92%
lucene 2.0	195	91	47%
lucene 2.2	247	144	58%
lucene 2.4	340	203	60%
nieruchomosci	27	10	37%
pbeans 1	26	20	77%
pbeans 2	51	10	20%
pdftranslator	33	15	45%
poi 1.5	237	141	59%
poi 2.0	314	37	12%
poi 2.5	385	248	64%
poi 3.0	442	281	64%
redaktor	176	27	15%
serapion	45	9	20%
skarbonka	45	9	20%
sklebagd	20	12	60%
synapse 1.0	157	16	10%
synapse 1.1	222	60	27%
synapse 1.2	256	86	34%
systemdata	65	9	14%
szybkafucha	25	14	56%
termoproject	42	13	31%
tomcat	858	77	9%

Continued on next page

Product	#Classes	#Defect-prone	% Defect-prone
velocity 1.4	196	147	75%
velocity 1.5	214	142	66%
velocity 1.6	220	78	35%
workflow	39	20	51%
wspomaganiepi	18	12	67%
xalan 2.4	723	110	15%
xalan 2.5	803	387	48%
xalan 2.6	885	411	46%
xalan 2.7	909	898	99%
xerces initial	162	77	48%
xerces 1.2	440	71	16%
xerces 1.3	453	69	15%
xerces 1.4	588	437	74%
zuzel	29	13	45%
Total	17681	6062	34%

The second data set we used was also obtained from the tera-PROMISE repository and is a the preprocessed version of the NASA MDP data set provided by Shepperd et al (2013) and contains data about 12 different projects. The reason why we use the preprocessed version by Shepperd et al is that Gray et al (2011) noted problems with the consistency of the originally published data, which Shepperd et al tried to resolve with their further processing. The projects in the data sets share 17 static source code metrics, mostly Halstead metrics (Halstead, 1977), but also additional metrics, such as LOC and Cyclomatic Complexity (VG) (McCabe, 1976). Table 3 gives information about the number of modules of each project, including the number of defect-prone modules and their percentage. For simplicity, we will refer to this data set as MDP data from now on.

The third data set is also publicly available online[5] and was donated by D'Ambros et al (2010). It contains data about 5 different Java projects. The data contains 31 metrics for each class of a project. The major difference between this data set and the other two data sets is, that this data does not only contain 17 static source code metrics, but also 14 process metrics that measure changes in the classes, e.g., number of lines that have been changed in a release. Table 4 gives information about the number of classes of each project, including the number of defect-prone classes and their percentage. For simplicity, we will refer to this data set as JPROC data from now on.

We do not merge the data from different data sets for three reasons. First, we get three independent results for the comparisons between the classification models which increases the validity of our results. Second, the common subset of metrics between the data sets is very small. The only metric that is used

[5] `http://bug.inf.usi.ch/`

in all three data sets is the cyclomatic complexity (McCabe, 1976). Third, we think that the combination of data from different sources is a research topic on its own (e.g. He et al, 2014; Nam and Kim, 2015), that we do not want to mix with our investigation of local models for cross-project defect predictions.

In the following, we will refer to each version of a project in a data set as a software product. For our experiments, we need to select target products and appropriate available candidates for training data. As target product, we select each software product once, i.e., we apply each experiment configuration to the 62 products in the JSTAT data, the 12 products in the MDP data, and the 5 products in the JPROC data. For the candidate data for the training we differentiate between *strict* cross-project data and *mixed* cross-project data. Strict cross-project data does not contain any previous data about the project, i.e., no old revisions of the same project. With mixed cross-project data, we allow old revisions of the same project.

To setup the strict cross-project data, we follow the methodology proposed by Herbold (2013). The candidates for the training data are based on the target product. Only products from the same data set as the target product are used as candidates, i.e., products from the JSTAT are not used to predict products from the JPROC data and vice versa. This is due to the differences in the metrics, as we discussed above. Moreover, the data often contains multiple versions of the same product, e.g., the versions 1.3–1.7 of ant. For the strict cross-project data, no other versions of the target product are allowed. Hence, when we train a classification model for ant 1.5, the versions ant 1.3–1.4 and 1.6–1.7 are not part of the candidate training data.

For the mixed cross-project data, we only consider the JSTAT data, because most products in that data set have multiple revisions. We only select those revisions as target product, for which a previous revision exists, i.e., for ant we select the revisions 1.4–1.7 as target product. This leaves us with 31 out of the 62 products as target product. The candidates for the training data are all other products from the JSTAT data, with the exception the the versions of the same project that are newer than the target product. This means that for ant 1.5, the versions 1.3–1.4 of ant are part of the training data, but not versions 1.6–1.7. This selection for the training data is also the reasons we restrict the target products to the 31 products as described above: for all other, the results would be equal to the strict setting because the training data would be exactly the same.

We took all data sets as is and did not perform any pre-processing or outlier treatment.

Table 3: Software products from the MDP data that are part of the study, including their total number of modules and the number of defect-prone modules.

Product	**#Modules**	**#Defect-prone**	**% Defect-prone**

Continued on next page

Product	#Modules	#Defect-prone	% Defect-prone
CM1	344	42	12%
JM1	9593	1759	18%
KC1	2096	325	16%
KC3	200	36	18%
MC1	9277	68	1%
MC2	127	44	35%
MW1	264	27	10%
PC1	759	61	8%
PC2	1585	16	1%
PC3	1125	140	12%
PC4	1399	178	13%
PC5	17001	503	3%
Total	43770	3199	7%

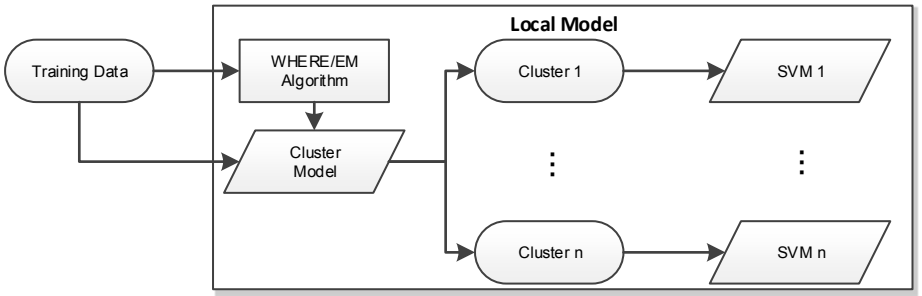
Table 4: Software products from the JPROC data that are part of the study, including their total number of classes and the number of defect-prone classes.

Product	#Classes	#Defect-prone	% Defect-prone
lucene	691	64	9%
pde	1497	209	14%
mylyn	1862	245	13%
eclipse	997	206	21%
equinox	324	129	40%
Total	5371	893	16%

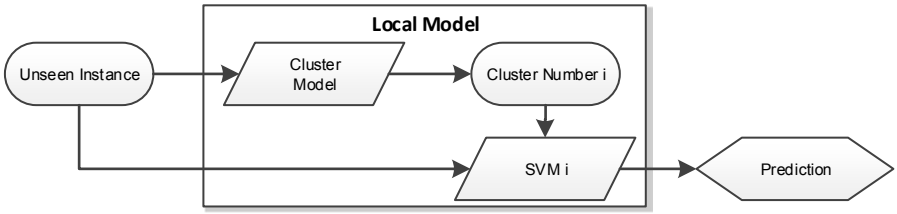
4.2 Classification Models

Our experiments are designed to compare local models with global models as well as transfer learning. Moreover, we want to evaluate if normalization has an impact on the local models. Through normalization all metric data is scaled to the interval $[0, 1]$, which reduces the impact of differenct metric scales and also harmonizes data between projects. In total, we consider six different classification models. A classification model is the combination of training data treatment (e.g., clustering for local models, data selection for transfer learning, or simply using all data as is) with a classifier that is trained following the data treatment (e.g., a SVM, C4.5 decision tree, or random forest).

1. N-WHERE: a local classifier that uses the WHERE algorithm for clustering and normlizes all metric data, i.e., scales it to the interval $[0, 1]$ before clustering;



(a) Instantiation of the training of a local model.



(b) Instantiation of the prediction with a local model.

Fig. 3: Instantiation of the training and prediction local model with the WHERE/EM algorithm for the training of the clustering model and a SVMs as internally used classifier.

2. WHERE: a local classifier that uses the WHERE algorithm for clustering;
3. N-EM: a local classifier that uses the EM algorithm for clustering and normlizes all metric data, i.e., scales it to the interval $[0, 1]$ before clustering;
4. EM: a local classifier that uses the EM algorithm for clustering;
5. GLOBAL: a normal classifier that builds a model on all training data;
6. KNN: a classifier trained with data selected with the k-nearest neighbor approach proposed by Turhan et al (2009). We use $k = 10$, same as Turhan et al (2009) in their study.

For all six models, we internally train a SVM with a Radial Basis Function (RBF) kernel for classification (Schölkopf and Smola, 2002). SVMs were found to be a very generic classifier that performs well in almost any setting (van Gestel et al, 2004). Moreover, we found that in case bias treatment is performed (see below), SVMs perform very well for defect prediction (Herbold, 2013). Figure 3 shows the instantiation of the local models with the WHERE/EM algorithm for clustering and the SVM for classification.

Since most of the data contains more non-defect-prone entities than defect-prone entities, there might be a classification bias towards non-defect-prone classifications. To handle this potential bias, we use undersampling (Drummond and Holte, 2003). In case there are more non-defect-prone entities in a data set, undersampling randomly draws from the non-defect-prone entites to create a data set with the same number defect-prone and non-defect-prone

entities. As a result of undersampling, all defect-prone and a subset of the non-defect prone entities is contained in the data. We repeated the experiments 10 times and report only on the average values. The dependent variable of the models is the indicator variable which indicates if a class/module contained a defect, the independent variables are the software metrics.

4.3 Analysis Procedure

To evaluate our hypotheses, we train each of the six classification models (N-WHERE, WHERE, N-EM, EM, GLOBAL, KNN) on each of the three data sets (JSTAT, MDP, JPROC). This gives us a total of twelve defect classification models.

To evaluate the classification models we use five performance metrics. The first four metrics are *recall*[6], *precision*, *F-measure*, and *error*. The metrics are defined as

$$recall = \frac{tp}{tp + fn}$$
$$precision = \frac{tp}{tp + fp}$$
$$F\text{-}measure = 2 \cdot \frac{recall \cdot precision}{recall + precision}$$
$$error = \frac{fp + fn}{tp + fn + tn + fp}$$

where tp, respectively, tn is the number of true positive, resp., negative predictions, fp, resp., fn is the number of false positive, resp., negative predictions. The *recall* measures how many of the existing defects are found. The *precision* measures how many of the found results are actually defects. The *F-measure* is the harmonic mean between *precision* and *recall*. The *error* measures the overall rate of misclassifications. Additionally, we consider the *Area Under the ROC Curve (AUC)*. The ROC stands for receiver operating characteristic and is the plot of the false positive rate ($\frac{fp}{fp+tn}$) against the true positive rate (i.e., recall). AUC is distributed between zero and one. The higher the value, the better the performance of a classifier. A value of value of 0.5 indicates that the performance is roughly equal to random guessing.

We choose these measures because they are widely used in defect prediction studies. We are aware that other metrics might also be well suited, e.g., *G-measure*, *AIC*, and more. However, we want to note that there is no agreement within the community on the best metrics for the evaluation of defect prediction results (Jiang et al, 2008).[7]

[6] instead of *recall*, sometimes *PD* or *tpr* are used in the literature. *PD* stands for probability of defect and *tpr* for true positive rate.

[7] This problem is still very relevant. For example, during the 37th International Conference on Software Engineering held in May 2015, there were five papers on defect prediction

To evaluate hypotheses **H1**–**H7**, we rank the classification models using the Scott-Knott test (Scott and Knott, 1974; Jelihovschi et al, 2014). With this test, we created ranked groups based on the significant differences between the mean values of the respective performance metric achieved with the various classification models. As threshold value for the significance of results, we use $p < 0.05$, i.e., 95% confidence in the result. We then evaluate our hypotheses **H1**, **H2**, **H3**, respectively, **H4** by evaluating the rankings according to the performance measures *recall*, *precision*, *F-measure*, respectively, *error*. For hypothess **H5**–**H7** we consider the rankings according to all five performance measures. Moreover, the consider the improvement ratio of AUC in order to get a better view of the overall difference between local models, the GLOBAL model, and the KNN model. The improvement ratio is defined as the

$$ir = \frac{alternative\ value}{original\ value}. \tag{1}$$

For example, if the GLOBAL model achieves an AUC of 0.6 and the WHERE model achieves an AUC of 0.7 the improvement ratio of WHERE in comparison to GLOBAL is

$$ir = \frac{0.7}{0.6} = 1.167. \tag{2}$$

To evaluate hypothesis **H8**, we consider the mean values for all five performance measures and perform a Mann-Whitney-U test Mann and Whitney (1947) to evaluate if the differences between the strict and the mixed cross-project setting we observe are statistically significant. Same as for the Scott-Knott test, we use a threshold of $p < 0.05$ for the significance. In case the difference is statisically significant, we say that the data with the better mean value wins. Otherwise we declare a draw between the strict and the mixed setting.

The Scott-Knott test, respectively the Mann-Whitney-U test is applied to rank, respectively compare all results achieved with a classification model on a specific data set, e.g., all versions of JSTAT.

4.4 Results

The presentation of the results is split in two parts: first, we present the results for our comparison of local model versus global models within a strict cross-project defect prediction setting. Then, we compare the strict cross-project setting to the mixed setting. We only report the most important numbers for the evaluation of our results. The complete results are published online[8]. The published results also contain information on how to replicate our case study

(Caglayan et al, 2015; Ghotra et al, 2015; Peters et al, 2015; Tan et al, 2015; Tantithamthavorn et al, 2015). None of them used exactly the same performance measures.

[8] Github: `https://github.com/sherbold/replication-kit-emse-2016-local-models/tree/master/replication-kit`
Zipped Archive: `http://hdl.handle.net/21.11101/0000-0001-3C55-D`

with the CrossPare tool (Herbold, 2015) as well as additional performance metrics we collected during the experiments.

4.4.1 Local vs. Global Models

The results of the comparison of the local and global models are summarized in Table 5, Figure 4, and Figure 5. The subtables of Table 5 shows the mean values achieved with the metric and the ranking we determined with the Scott-Knott test. The subfigures of Figure 4 show box plots of the results. The subfigures of Figure 5 show box plots of the improvement ratio of AUC of the models in comparison to the GLOBAL and the KNN model.

For the *recall*, we observe that the GLOBAL model and the EM model share the best average rankings with 1.66. EM ranks twice first and once third, whereas global ranks only once first, and twice second. The other four models share the same average ranking of 2.

For the *precision*, we observe that N-WHERE model has the best average ranking with 1.33, ranking first twice and second once. The KNN model and the EM model follow with an average ranking of 1.66, both having one first and two second places.

For the *F-measure*, we observe that all models perform nearly the same, i.e., there are no statistically significant different ranks. Only the N-WHERE model is ranked second behind the others once for the MDP data set, due to the extremly poor recall on that data. This leads to an average rank of 1 for all models except N-WHERE and an average rank of 1.33 for the N-WHERE model.

For the *error*, we observe that the N-WHERE and the EM model have best average rank with 2. The N-WHERE model ranks fourth and last for JSTAT data, on which the EM model ranks first. On the other hand, the N-WHERE model ranks first on the MDP and JPROC data, where the EM model is only ranked third, respectively second. The GLOBAL and KNN models follow with an average rank of 2.33, the WHERE and N-EM model are last with an average rank of 2.66.

For the AUC, we observe that the EM model is notably the best and always ranks first. The GLOBAL models follows with an average rank of 1.33 and is only beaten by the EM model on the JSTAT data. Then WHERE, N-EM, and KNN model follow with an average rank of 1.66. The N-WHERE is on the (shared) last rank for all three data sets and has an average rank of 2.33. Moreover, we should note that the all classification models are on the same rank for the JPROC data set, i.e., there are no statistically significant differences accourding to the Scott-Knott test. While none of the mean values of AUC are very high, some are very close to 0.5, i.e., the performance of a completely random model. The lowest mean is achieved by the N-WHERE model on the JSTAT data with 0.511.

For all measures we observe that the mean values are well correlated by the rankings according to the Scott-Knott test, that the mean values for clas-

Table 5: Results for the performance metrics. The tables contain the rankings determined with the Scott-Knott test and the mean values of the performance metrics.

(a) Results *recall*

	N-WHERE rnk/*rec.*	WHERE rnk/*rec.*	N-EM rnk/*rec.*	EM rnk/*rec.*	GLOBAL rnk/*rec.*	KNN rnk/*rec.*
JSTAT	1 / 0.969	3 / 0.546	2 / 0.883	3 / 0.530	2 / 0.868	2 / 0.817
MDP	3 / 0.034	1 / 0.930	2 / 0.706	1 / 0.870	2 / 0.721	2 / 0.668
JPROC	2 / 0.309	2 / 0.456	2 / 0.416	1 / 0.658	1 / 0.680	2 / 0.231
avg. rnk	2	2	2	1.66	1.66	2

(b) Results *precision*

	N-WHERE rnk/*prc.*	WHERE rnk/*prc.*	N-EM rnk/*prc.*	EM rnk/*prc.*	GLOBAL rnk/*prc.*	KNN rnk/*prc.*
JSTAT	2 / 0.355	2 / 0.407	2 / 0.371	1 / 0.492	2 / 0.390	2 / 0.386
MDP	1 / 0.377	2 / 0.143	2 / 0.177	2 / 0.195	2 / 0.221	2 / 0.199
JPROC	1 / 0.479	2 / 0.301	2 / 0.281	2 / 0.318	2 / 0.325	1 / 0.642
avg. rnk	1.33	2	2	1.66	2	1.66

(c) Results *F-measure*

	N-WHERE rnk/*F-m.*	WHERE rnk/*F-m.*	N-EM rnk/*F-m.*	EM rnk/*F-m.*	GLOBAL rnk/*F-m.*	KNN rnk/*F-m.*
JSTAT	1 / 0.475	1 / 0.401	1 / 0.473	1 / 0.442	1 / 0.486	1 / 0.460
MDP	2 / 0.059	1 / 0.231	1 / 0.273	1 / 0.299	1 / 0.314	1 / 0.230
JPROC	1 / 0.297	1 / 0.347	1 / 0.319	1 / 0.408	1 / 0.409	1 / 0.290
avg. rnk	1.33	1	1	1	1	1

(d) Results *error*

	N-WHERE rnk/*err.*	WHERE rnk/*err.*	N-EM rnk/*err.*	EM rnk/*err.*	GLOBAL rnk/*err.*	KNN rnk/*err.*
JSTAT	4 / 0.641	2 / 0.453	3 / 0.581	1 / 0.360	3 / 0.535	3 / 0.543
MDP	1 / 0.121	4 / 0.713	3 / 0.412	3 / 0.454	2 / 0.340	3 / 0.512
JPROC	1 / 0.211	2 / 0.312	2 / 0.315	2 / 0.336	2 / 0.343	1 / 0.178
avg. rnk	2	2.66	2.66	2	2.33	2.33

(e) Results *AUC*

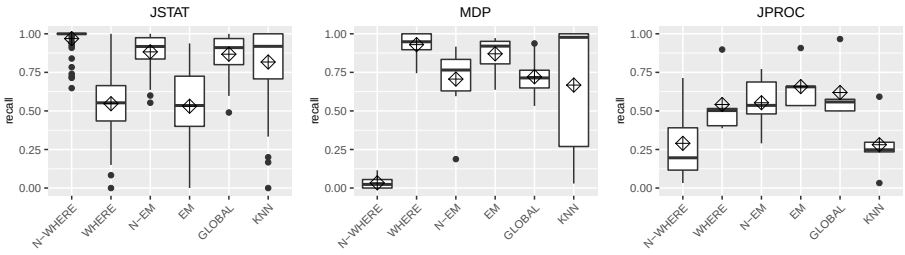
	N-WHERE rnk/*AUC*	WHERE rnk/*AUC*	N-EM rnk/*AUC*	EM rnk/*AUC*	GLOBAL rnk/*AUC*	KNN rnk/*AUC*
JSTAT	4 / 0.511	2 / 0.565	3 / 0.538	1 / 0.635	2 / 0.576	2 / 0.550
MDP	2 / 0.516	2 / 0.566	1 / 0.626	1 / 0.678	1 / 0.680	2 / 0.557
JPROC	1 / 0.610	1 / 0.597	1 / 0.579	1 / 0.658	1 / 0.667	1 / 0.596
avg. rnk	2.33	1.66	1.66	1	1.33	1.66

sification models on the same rank are similar to each other, and that there is a gap between the mean values of the classification models on the next rank.

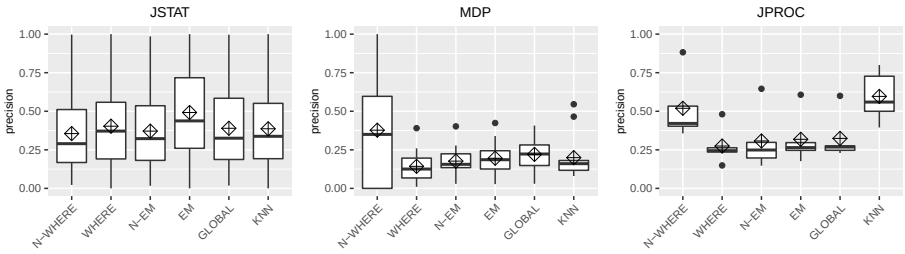
4.4.2 Strict vs. Mixed Data

The results for the comparison between strict and mixed cross-project data on the 31 versions of the JSTAT data for which we also have older versions in the data are summarized in Table 6. The subtables show the mean values on

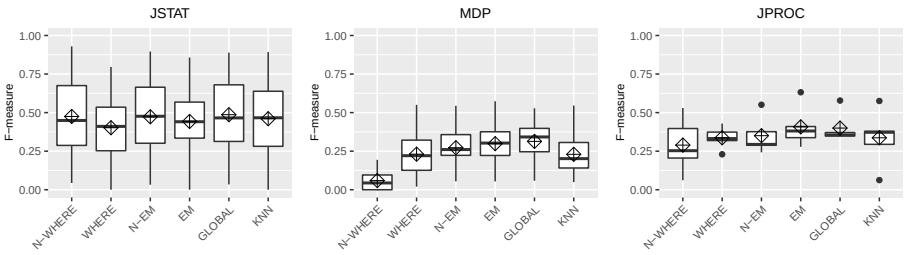
Fig. 4: Boxplots of the performance metrics for each data set. The diamonds mark the mean values.



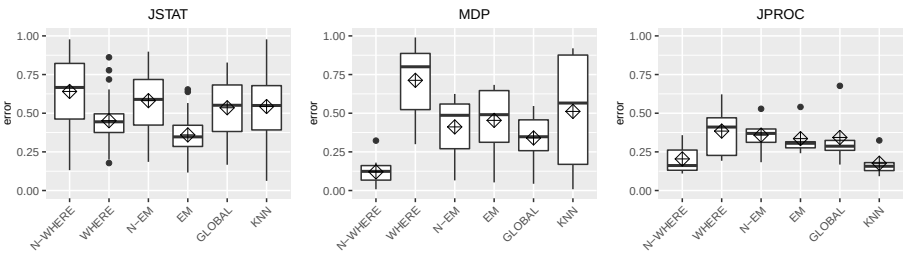
(a) Results *recall*



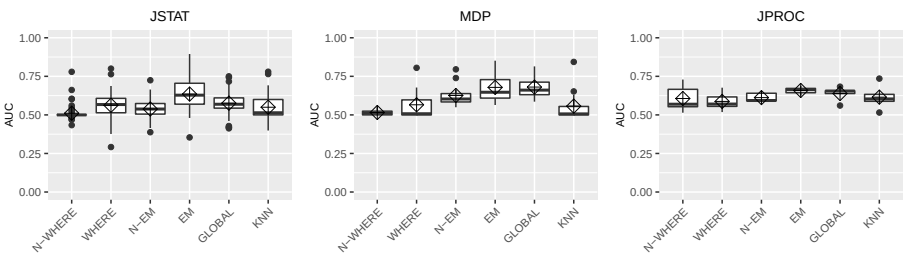
(b) Results *precision*



(c) Results *F-measure*

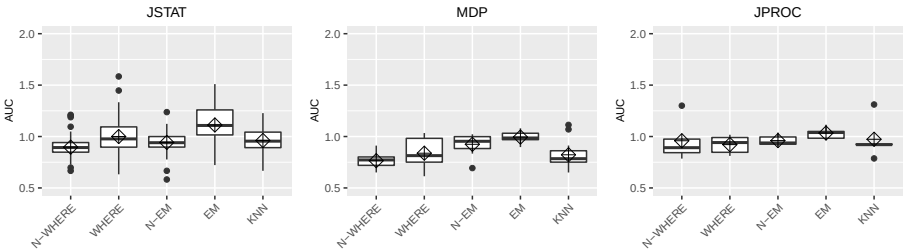


(d) Results *error*

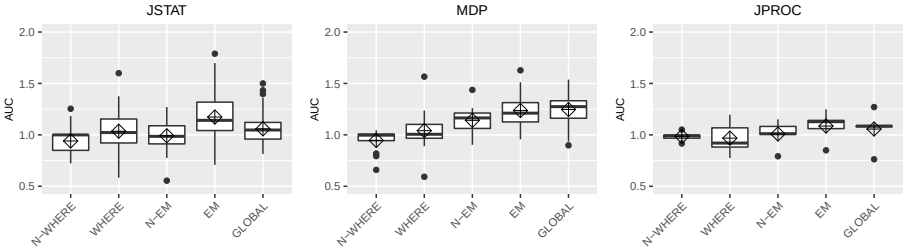


(e) Results *AUC*

Fig. 5: Boxplots of the improvement ratio of the results in comparison to the KNN and the GLOBAL model for AUC and for each data set. The diamonds mark the mean values.



(a) Improvement ratio of AUC in comparison to the GLOBAL model



(b) Improvement ratio of AUC in comparison to the KNN model

the 31 products for both the strict and mixed data, as well as the *p-value* of the Mann-Whitney-U test and the winner between the two data sets.

The results show that the *recall* of the strict setting is better than that of the mixed setting. Only for the EM and the WHERE model, the results are a draw, for the other four models the strict setting produces significantly better results with a quite large gap of at least 0.215. For the *precision*, we observe no significant difference between the strict and mixed data, although the mean values for the mixed data are slightly higher than for the strict data. For the *F-measure*, we also observe no significant differences, which is surprising due to the quite large difference in the *recall* and the unsignificant differences in the *precision*. However, the small and stastically insignifant higher mean *precision* of the mixed data seems to be enough to offset the statistically significantly higher *recall*, which leads to draws for the *F-measure*.

The *error*, we observe a completely different effect as for the recall. The EM and WHERE model have draws again, but for the other four models the mixed data is significantly better than the strict data. The same holds for the *AUC*, except that the N-EM model also has a draw and only three models, i.e., N-WHERE, GLOBAL, and KNN are significantly better on the mixed data.

Table 6: Results for the comparison between strict and mixed cross-project predictions on the JSTAT data.

(a) Results *recall*

	strict	mixed	p-value	win
N-WHERE	0.951	0.702	$< 10^{-8}$	strict
WHERE	0.589	0.604	0.910	draw
N-EM	0.875	0.580	$< 10^{-8}$	strict
EM	0.550	0.541	0.768	draw
GLOBAL	0.854	0.545	$< 10^{-7}$	strict
KNN	0.857	0.642	$< 10^{-4}$	strict

(b) Results *precision*

	strict	mixed	win	
N-WHERE	0.363	0.424	0.190	draw
WHERE	0.409	0.424	0.878	draw
N-EM	0.378	0.411	0.605	draw
EM	0.463	0.471	1	draw
GLOBAL	0.396	0.489	0.160	draw
KNN	0.397	0.463	0.245	draw

(c) Results *F-measure*

	strict	mixed	win	
N-WHERE	0.472	0.489	0.625	draw
WHERE	0.425	0.438	0.746	draw
N-EM	0.472	0.413	0.363	draw
EM	0.450	0.440	0.652	draw
GLOBAL	0.484	0.453	0.675	draw
KNN	0.483	0.471	0.889	draw

(d) Results *error*

	strict	mixed	win	
N-WHERE	0.636	0.413	$< 10^{-4}$	mixed
WHERE	0.452	0.440	0.706	draw
N-EM	0.579	0.475	0.011	mixed
EM	0.376	0.374	0.838	draw
GLOBAL	0.537	0.389	$<^1$ 0−3	mixed
KNN	0.539	0.411	0.012	mixed

(e) Results *AUC*

	strict	mixed	win	
N-WHERE	0.518	0.620	$< 10^{-5}$	mixed
WHERE	0.577	0.587	0.476	draw
N-EM	0.539	0.566	0.109	draw
EM	0.623	0.628	0.822	draw
GLOBAL	0.574	0.633	0.001	mixed
KNN	0.575	0.623	0.011	mixed

5 Discussion

In the following, we evaluate our research hypotheses with respect to the results of the case study. Afterwards, we compare our findings with the findings

of previous studies. Finally, we comment on the overall performance of all classification models.

5.1 Hypotheses Evaluation

H1: Local models have a lower recall than global models.

Our results regarding the *recall* are inconclusive. The EM ranks first twice for the MDP and JPROC data, but fails on the JSTAT data with the last place. The GLOBAL model has the same shared rank of 1.66 and is outperformed by the EM model on the MDP data, but clearly beats it on the JSTAT data. Depending on the point of view, one might say that either the EM model is better (higher ceiling) or the GLOBAL model (more consistency). When we compare the local models to each other, the EM model is the clear winner, followed by the N-EM model due to its consistency. The WHERE and N-WHERE model behave erratically and should, therefore, not be selected if the recall is important.

H2: Local models have a higher precision than global models.

Our results show only small differences in the *precision*. Especially the N-WHERE model yields a consitently good precision. It is the only model that achieves an average precision of at least 0.35 on all three data sets. The other models are very similar to each other, with nearly no significant differences. Hence, we conclude that the N-WHERE model is a good choice if a robust but on average still rather low *precision* is important and that local models thus can indeed have higher precision than global models, but if and only if the correct clustering strategy is used.

H3: Local models and global models perform similar in terms of the F-measure

Our results show that there is almost no difference in the *F-measure* between the global and the local models. The only outlier is the N-WHERE model for the MDP data, which performs significantly worse on this data set due to an extremely low recall. Due to these findings, we find strong support for this hypothesis and conclude that the differences between local and global models offset when *recall* and *precision* are both considered.

H4: Local models have a lower error than global models.

Our results regarding the *recall* are inconclusive. The GLOBAL model is outperformed by one local model on each data set (EM and WHERE on JSTAT, N-WHERE on MDP and JPROC), but ranks very consistently between the local models. The average rankings of the N-WHERE and EM model are slightly better than that of the GLOBAL model, but both are also clearly

outperformed by the GLOBAL model once. Hence, our findings here are similar to the findings for the *recall*: the local models seem to have a higher ceiling if they are used in the right circumstances, but the GLOBAL model is more consistent.

H5: Local models and global models perform similar in in terms of AUC.

Our results regarding AUC clearly distintuish the EM model as the best overall model, through a consistent first rank. However, the EM model is followed very closely by the GLOBAL model, which is only slightly worse on the JSTAT data. The worst model is clearly the N-WHERE model. Not only does it have the worst average rank, but it also has mean values very close to randomness for the JSTAT and the MDP data. For *AUC*, we also considered the improvement ratio of the local models in comparison to the GLOBAL model (see Figure 5(a).) The improvement ratio supports the conclusions drawn based on the mean values and Scott-Knott rankings. The EM model outperforms the GLOBAL model on the JSTAT data, on the other data, the results are quite similar. Hence, we cannot conclude that local models outperform global models or vice versa. Instead, that the right local model (i.e., the EM model) can beat the GLOBAL model, but more often than not the GLOBAL model wins.

H6: Local models perform better than the k-nearest neighbor model by Turhan et al (2009).

Our results regarding this research question mimic the results for the comparison between the GLOBAL and the local models model, because KNN and GLOBAL perform rather similar for most metrics. Hence, the local models sometimes have a higher ceiling than the KNN model, but the KNN model is more robust. In total, at least the EM model seems to be consistently on par or better than the KNN model, the difference between KNN and the other local models seem to be completely dependent on the data the performance metric that is used. For *AUC*, we also considered the improvement ratio of the local models in comparison to the KNN model (see Figure 5(b)). The improvement ratios support the conclusions drawn based on the mean values and Scott-Knott rankings. The EM model outperforms the KNN model on all three data set. The N-EM model also offers a slight advantage over the KNN model on the MDP data, but not on the other data sets. Hence, we conclude that we found one local model that outperforms the KNN model and for the others it depends on the circumstances.

H7: Local models perform better if the data is normalized prior to the clustering.

Our results show that this depends on the clustering algorithm that is used and the data under consideration. The N-WHERE model performs better than

the WHERE model on the MDP and JPROC data, but worse on the JSTAT data. On the JSTAT data, it seems as if normalization actually leads to trivial N-WHERE models, where almost everything is predicted defective, hence the very high recall.

The EM model consistently outperforms the N-EM model. Here, normalization seems to hinder the clustering. This effect can be explained because EM internally works with probability distributions and the variance plays an important role. By changing the metrics scales to the interval [0,1] through a linear transformation, we change the relative variance in the data. The EM clustering apparently works better, if no such changes are performed. Hence, if normalization or some other form of data transformation is to be applied, it should not change the relative variance within the scale.

Thus, our conclusion regarding this hypothesis that normalization can help if applied judiciously, but one should check the potential consequences normalization may have on the clustering first.

H8: Mixed cross-project data yields better results than strict cross-project data.

The results show that this depends on the perfomance metric. On the one hand, if one is interested in a high *recall*, strict cross-project data performs better than mixed data. Previous research already indicated that cross-project data allows a higher *recall* than within-project data (Herbold, 2013). On the other hand, if the overall error or misclassification rate independent of the class is important (i.e., independent of whether an entity is defect prone or not), the mixed data seems to perform better as is shown by the better results in terms of *error* and AUC. In terms of *precision*, and *F-measure*, the differences between the strict and mixed data are not statistically significant.

5.2 Comparison with Previous Studies

In comparison to the study by Menzies et al (2011, 2013), our results show no big gain by using local models. This is likely caused by the different research focus: we focus on prediction in the very difficult cross-project setting, whereas Menzies at al focus on finding rules for known data to infer lessons about how to prevent defect and decrease effort.

Considering the study of Bettenburg et al (2012, 2014), our results are mostly in line with their findings or conjectures. From their results, they concluded that “global models produce general trends” whereas “local models produce too much insight”. Since the predictions are made on data from a different context than the training data, the general trends from the global models are almost as accurate as the better insights into local regions gained through the local models. However, Bettenburg et al also note that “without careful calibration of parameters, the resulting local models can be considerably worse than their global model counterparts”. Since we did not apply any

manual tuning, but only used clustering algorithms with automated parameter selection, local models might still outperform global models. However, to our mind, tuning on the local models would likely lead to overfitting, which is especially problematic for cross-project defect prediction.

In comparison to Scanniello et al (2013), we did not observe a major advantage in the *error* of the local models over other models. However, this does not mean that our results directly contradict those of by Scanniello et al First of all, we did not take architectural factors into account, when we created the local models. Hence, we cannot rule out that if we would have used architectural features during the creation of our local model, that we would not have seen the same reduction in *error*. However, the clustering approach based on architectural dependencies cannot be applied in a cross project setting: there are no architectural dependencies between the different contexts. A second explanation for the difference in the results is that we used the cross-project setting instead of cross-validation with data from the same project.

5.3 Insights Into the Local Models

While the main motivation for our work was the possible advancement of the state of the art of cross-project defect prediction through local models, we also gained some insights into the inner workings on local models which can be useful for future applications.

The first interesesting aspect we observed is that local models can be combined with other techniques, but it should be done judiciously as the combination of the local model with something else might actually be detrimental. This is shown by the effect of normalization: it works quite well the the WHERE model, but decreases the performance of the EM algorithm. For WHERE, only the distances matter, which only change lineary with normalization, hence, the relative distances are preserved. For EM on the other hand, the variance in the data matters, which is changed by the normalization. Such effects are easy to overlook, which is why we propose to always compare a local model without modifications to a local model that is modified in some way with another technique to determine if there is actually an advantage.

The second aspect is that we believe that neither of the EM nor the WHERE model are close to where local models could be, if a good clusterer can be found. We gained this intuition by the sometime huge performance differences between the local models. If a clusterer could be found, that would combine the best aspects of EM and WHERE, the performance would already be much better. However, we believe that this is not simply done with the choice of a correct clustering algorithm. Instead, the correct meta-data about the entities must be determined in order to really create homogenuous clusters. This problem is closely related to the problem of selecting training data for cross-project defect predictions. We believe that a combination of the following aspects is required to create a good local model.

- Static metrics: factors like the size and complexity of an entity are used in all three data sets used in this study. Our results, as well as related work on defect prediction (Menzies et al, 2008) indicate that these metrics are useful, but not sufficient for good defect prediction models.
- Process metrics: the number of changes to an entity as it is part of the JPROC data set. A recent study by Madeyski and Jureczko (2015) on which metrics could be useful listed these metrics as likely candidates for improvement, which is also supported by related work on within-project defect prediction, e.g., Hassan (2009).
- Social metrics: factors like the number of different commiters metrics gained through the analysis of developer social networks (Meneely et al, 2008). The study by Madeyski and Jureczko (2015) also listed social metrics among the likely candidates for model improvement.
- Entity context: a currently missed attribute in defect prediction is the type of entity considered, e.g., if the entity is a data class, a user interface class or a network connection handler. The structure and fault types associated with these different contexts differ, hence, we believe that if it were possible to include this information defect prediction models could be improved.

If all the information were available, it would be possible to create clusters that really contain software entities from the same context, of the similar size, developed by people with similar skills, and so on. Moreover, the additional information could also be harnessed by the defect prediction model, which may further improve the performance. However, for such local models clustering might not even be the best approach. Instead, the idea by Zimmermann et al (2009) to create an approprate decision tree that selects similar data could be revisited, instead of for complete projects. However, due to the current lack of a benchmark data set that contains all of the information, we could not analyze if these aspects really effect defect prediction within this study.

5.4 Comments on the Overall Performance

Our biggest concern with the results of our case study is not regarding the rather negative results regarding the advantages of local models, but the overall performance of local models and the transfer learning. In Table 7 we compare the results of the simple global model with the best result achieved with an advanced technique, i.e., a local model or the KNN transfer learning algorithm. We boldfaced the cases where the global model beats all competitors. Additionally, Figure 5(a) shows the improvement ratio of AUC for all models in comparison to the global model.

The comparison shows that the advanced techniques do not really offer a major advantage over a simple global model. This is also already indicated by the analysis of the research hypothesis. In the evaluation of the hypotheses we basically determined that there is no major difference between global and local models in terms of performance. There may be a small advantage in *precision* for the local models, but other than that, the global models are quite robust.

Table 7: Comparison between the results of the GLOBAL model with the best other model.

	JSTAT				
	recall	*precision*	*F-measure*	*error*	*AUC*
GLOBAL	0.868	0.390	**0.486**	0.535	0.576
Best other	0.969	0.492	0.475	0.360	0.635
	(N-WHERE)	(EM)	(N-WHERE)	(EM)	(EM)
	MDP				
	recall	*precision*	*F-measure*	*error*	*AUC*
GLOBAL	0.721	0.221	**0.314**	0.340	**0.680**
Best other	0.930	0.377	0.299	0.121	0.678
	(WHERE)	(N-WHERE)	(EM)	(N-WHERE)	(EM)
	JPROC				
	recall	*precision*	*F-measure*	*error*	*AUC*
Global	0.680	0.325	**0.409**	0.343	**0.667**
Best other	0.658	0.642	0.408	0.211	0.658
	(EM)	(KNN)	(EM)	(N-WHERE)	(EM)

Only **H5** shows some advantage of in terms of AUC, but only for the EM model and only on one data set. In terms of *F-measure*, the global model is not beaten once. Through this analysis, we do not want to state that global models are always better or equal to advanced techniques for cross-project defect prediction. As we already said, we choose the KNN model, because it is used as baseline comparison for cross-project defect prediction. Hence, there are approaches that have beaten the KNN model in previous experiments.

However, we would like to make researchers aware that we could not find statistically significant major differences that indicate that advanced techniques are better within our study. Therefore, we would urge other researchers to always use a global model as comparison in cross-project defect prediction experiments in order to demonstrate the gain against this simple technique.

5.5 Other classifiers

While we only report on the results achieved with the SVM, we found similar findings for other classifiers. We internally performed experiments with logistic regression, random forests, Naïve Bayes, and C4.5 decision trees. For all of these classification models, we did not find an advantage for using local models. Moreover, the performance in terms of the reported on performance metrics was slightly worse than for the SVM for those models, including the low *AUC*. Only the random forest achieved a similar performance. We included the results for the random forest in the replication kit for comparison. Further details on the experiments are not reported here, because the study is on local vs. global and not on the performance of different classifiers.

6 Lessons Learned

When we look at the results of our case study, our findings are quite straightforward: while the local models are different from the global models, the evidence suggests that they can perform similar to global models, but not actually beat them consistently, even though they sometimes show better results for single metrics. The EM model performs overall quite similar to the GLOBAL model, with some differences where it is open to interpretation which one performs better. The other three configurations of the local models perform for the most part worse than the GLOBAL model. Moreover, we observe that local models do not solve the problems with *precision*. The N-WHERE model sometimes improves the *precision*, but has such big problems with the *recall* in those instances, that this does not matter. Therefore, we came to the following conclusion regarding **RQ1**:

> Local models are not significantly different from global models in terms of overall prediction performance for cross-project defect prediction. Hence, we suggest carefull evaluation if the local models actually bring a benefit in comparison to a global view in terms of prediction performance.

Regarding our second research question, we did not observe a big difference between the KNN approach by Turhan et al (2009) and the local models either. This means that local models might be able to compete with transfer learning, but problably only if they are further adapted to the cross-project setting. Therefore, we came to the following conclusion regarding **RQ2:**

> Local models are comparable with the KNN transfer learner and may be able to compete if they are tailored to the problem or combined with transfer learning.

For our third research question, i.e., if normalization is helpful, our results show that it can help, but one must be carefull with the effects it might have on the clustering. Therefore, we came to the following conclusion regarding **RQ3:**

> Normalization should be applied judiciously and only after checking if it might have adverse effects on the clustering algorithm. If there are no adverse effects, normalization can improve the results, but this should always be checked through a comparison with the results without normalization.

While we only investigated normalization in our replication study, we believe that this lesson learned can be transfered to other data transformation techniques, e.g., Z-score normalization (Kotsiantis et al, 2006).

Regarding our fourth research question our findings are clear and it depends on the performance metrics. Therefore, we came to the following conclusion regarding **RQ4:**

> The strict setting is better if *recall* is the most important performance metrics, if the overall performance matters the mixed setting yields better results.

However, our most important lesson learned from our study is that the perfomance of nearly all models we used was quite similar. As we discussed in Section 5.4, the global model is not outperformed. From this, we conclude the following:

> Global classification models based on all available data models are not consistently beaten by advanced techniques and should, therefore, always be used for baseline comparisons.

7 Threats to Validity

There are several threats to the validity of our case study results and our drawn conclusions. In the following, we will list both the internal and external factors we identified.

7.1 Internal Threats

The major source for threats for the internal validity of our case study are the choice of the clustering algorithms. We did not perform any tuning of the clustering algorithms or test other clustering algortihms. The WHERE and EM clustering algorithms may not be very good choices for local models and the detection of homogeneous regions. Our modification of the QuadTree algorithm to work with data outside the observed range during training may also lead to bad results.

We did not sample k used for the KNN model, but reused the value $k = 10$ that was used by Turhan et al (2009). A bad choice of k could lead to unnecessarily bad results for the KNN model. However, Turhan et al (2009) use the same MDP data we use in our study, therefore, we believe that at least for this data, the choice of k is good, even though we did not sample this. Furthermore, He et al (2013) and Peters et al (2015) both use subsets of the JSTAT data we use, also with $k = 10$ and did not note any problems specific to this choice of k. Only for the JPROC data, the value of $k = 10$ was not evaluated, yet.

Moreover, we only used the KNN model as an example for transfer learning in order to evaluate **H6** and **RQ2**. This means we only compared the

local model to a rather simple transfer learning approach that is often used as baseline. Other transfer learning approaches that have shown promise in the cross-project defect prediction literature (e.g., Watanabe et al, 2008; Camargo Cruz and Ochimizu, 2009; Ma et al, 2012; Herbold, 2013; He et al, 2013; Zhang et al, 2014) may outclass the local models. Hence, our results show only that local models fulfill some mimimun criterion to be considered for cross-project defect predictions.

7.2 Construct Validity

We identified two major threats to the construct validity of our results. The first threat is the data that was used. Due to problems with the mining procedure used (Jureczko and Madeyski, 2010) or due to inconsistencies (Gray et al, 2011), the data may be noisy and contain misclassifications or wrong metric values. In order to mitigate the influence of this threat, we used the preprocessed version of the NASA MDP data. Moreover, we internally evaluated if the performance on the JSTAT changes if only products with at least 100 classes and at most 95% of the data classified the same way. This was done to check if filtering the data such that small products and extremely unbalanced products are removed changes the results. The results on this subset of 41 products are contained in the replication kit. There is no major deviation between the obtained results in terms of *F-measure*. The AUC on the smaller subset is only mildly improved with the ceiling still being 0.64 and with no improvement greater than 0.065. Moreover, this relatively low performance is in line with other literature on cross-project defect prediction that uses at least 40 products of the JURECZKO data (Amasaki et al, 2015; Kawata et al, 2015). Hence, we submit that the construct in itself is not responsible for the low performance values.

The second threat is that the criteria *recall*, *precision*, *F-measure*, *error*, and AUC which we used for the evaluation of the results might be unsuitable. The best choice of evaluation criteria is still a topic of discussion within the community. Different criteria may yield different results which could lead to a difference in the interpretation of the performance, which in turn might affect our conclusions.

7.3 External Threats

Our main concern regarding the external validity of our results is that the data we used might not be representative for software in general. Although we used a quite large corpus of data within this study, it were still only 79 software products. Therefore, we cannot be sure if our results have any external validity. The fact that other studies have found to some degree similar results does not help here, as the data used in those studies was a subset of the data we used. Hence, we cannot rule out that the effect is random and depends purely on the data that was used.

Furthermore, the data sets we chose only contain either closed source data (MDP) or open source data (JSTAT, JPROC). Data that contains a mixture of closed and open source may yield different results. Additionaly, our data did not contain any social metrics or context factors. The inclusion of these factors may change our results. Further studies with different data that covers these aspects are required.

8 Conclusion

In this article, we performed a conceptual replication study on local models with the focus on cross-project defect prediction. To this aim, we compared four local models, two based on the WHERE clustering algorithm and two based on the EM clustering algorithm with a global model and a transfer learning technique for cross-project predictions based on the k-nearest neighbor algorithm. For both the WHERE and EM clustering algoritm we considered one configuration with normalized data and one without. Within our replication study, we used three data sets and a total of 79 different software versions. Our results show no consistent and statistically significant difference between local and global models. The same holds true for the comparison between the local models and the transfer learning algorithm.

These results are somewhat disappointing, because local models showed promise in previous studies where they were applied to gain insight or in a within-project context. However, we could not find the same positive effects for predictions. This is in line with the conjecture of Bettenburg et al (2014) that local models may be problematic for predictions. This finding seems to hold for the cross-project context. Moreover, our results show that normalization can be helpful, but also detrimental to the performance of local models and that this depends on the clustering.

Additionally, we compared a strict cross-project defect prediction setting, without data from previous versions of the same project allowed, with a mixed setting where such versions are allowed. Our results indicate and advantage in recall for the strict setting, but better overall performance in the mixed setting.

However, the most interesting finding, to our mind, is that the global models actually performed similar and very close to being the best among all six models that we used. While we cannot state that they are the best approach, this finding at least implies that they should always be used for comparisons for the evaluation of any cross-project approach, regardless whether it is a transfer learning technique, local models, or something else.

In the future, we plan to combine local models with transfer learning techniques into *local transfer learning models*, i.e., models that first determine homogeneous regions and then use transfer learning to further harmonize the within-cluster data with a target product. Moreover, if we are able to extend the available training data with meta information about the project and entity context, this may allow us better clustering to determine homogeneous

regions. Furthermore, recent research result by Tantithamthavorn et al (2016) suggest that careful tuning of classifiers may be used for the improvement of cross-project results, which we will investigate for local models in the future. Finally, we plan to further investigate how well global models compare to other transfer learning techniques in order to evaluate the state-of-the-art of cross-project defect predictions.

References

Amasaki S, Kawata K, Yokogawa T (2015) Improving cross-project defect prediction methods with data simplification. In: 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)

Bettenburg N, Nagappan M, Hassan A (2012) Think locally, act globally: Improving defect and effort prediction models. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR), IEEE Computer Society

Bettenburg N, Nagappan M, Hassan A (2014) Towards improving statistical modeling of software engineering data: think locally, act globally! Empirical Software Engineering pp 1–42

Caglayan B, Turhan B, Bener A, Habayeb M, Miranskyy A, Cialini E (2015) Merits of organizational metrics in defect prediction: An industrial replication. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)

Camargo Cruz AE, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE Computer Society

Carver JC (2010) Towards reporting guidelines for experimental replications: A proposal. In: Proceedings of the International Workshop on Replication in Empirical Software Engineering

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

D'Ambros M, Lanza M, Robbes R (2010) An Extensive Comparison of Bug Prediction Approaches. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR), IEEE Computer Society

Dempster AP, Laird NM, Rubin DB (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. J of the Royal Statistical Society Series B (Methodological) 39(1):1–38

Drummond C, Holte RC (2003) C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In: Workshop on Learning from Imbalanced Datasets II

Faloutsos C, Lin KI (1995) FastMap: A Fast Algorithm for Indexing, Datamining and Visualization of Traditional and Multimedia Datasets. SIGMOD Rec 24(2):163–174

Fraley C, Raftery AE (1999) MCLUST: Software for Model-Based Cluster Analysis. Journal of Classification 16(2):297–306

van Gestel T, Suykens J, Baesens B, Viaene S, Vanthienen J, Dedene G, de Moor B, Vandewalle J (2004) Benchmarking Least Squares Support Vector Machine Classifiers. Machine Learning 54(1):5–32

Ghotra B, McIntosh S, Hassan AE (2015) Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)

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: Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE), IET

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter 11(1):10–18

Halstead MH (1977) Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc.

Han J, Kamber M, Pei J (2011) Data Mining: Concepts and Techniques. Morgan Kaufmann

Hassan A (2009) Predicting faults using the complexity of code changes. In: Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, pp 78–88, DOI 10.1109/ICSE.2009.5070510

He P, Li B, Ma Y (2014) Towards cross-project defect prediction with imbalanced feature sets. CoRR abs/1411.4228, URL `http://arxiv.org/abs/1411.4228`

He Z, Shu F, Yang Y, Li M, Wang Q (2012) An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering 19:167–199

He Z, Peters F, Menzies T, Yang Y (2013) Learning from Open-Source Projects: An Empirical Study on Defect Prediction. In: Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM)

Henderson-Sellers B (1996) Object-oriented Metrics; Measures of Complexity. Prentice-Hall Inc.

Herbold S (2013) Training data selection for cross-project defect prediction. In: Proceedings of the 9th International Conference on Predictive Models in Software Engineering (PROMISE), ACM

Herbold S (2015) Crosspare: A tool for benchmarking cross-project defect predictions. In: Proceedings of the 4th International Workshop on Software Mining (SoftMine)

Huang L, Port D, Wang L, Xie T, Menzies T (2010) Text Mining in Supporting Software Systems Risk Assurance. In: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE), ACM

Jelihovschi E, Faria J, Allaman I (2014) ScottKnott: a package for performing the Scott-Knott clustering algorithm in R. TEMA (S˜ao Carlos) 15:3 – 17

Jiang Y, Cukic B, Ma Y (2008) Techniques for evaluating fault prediction models. Empirical Software Engineering 13(5):561–595

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 (PROMISE), ACM

Kawata K, Amasaki S, Yokogawa T (2015) Improving relevancy filter methods for cross-project defect prediction. In: 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI)

Kitchenham B (2008) The role of replications in empirical software engineeringa word of warning. Empirical Software Engineering 13(2):219–221

Kocaguneli E, Menzies T, Keung J, Cok D, Madachy R (2013) Active learning and effort estimation: Finding the essential content of software effort estimation data. Software Engineering, IEEE Transactions on 39(8):1040–1053, DOI 10.1109/TSE.2012.88

Kotsiantis S, Kanellopoulos D, Pintelas P (2006) Data preprocessing for supervised leaning. Int J of Comp Sci 1(2):111–117

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

Madeyski L, Jureczko M (2015) Which process metrics can significantly improve defect prediction models? an empirical study. Software Quality Journal 23(3):393–422, DOI 10.1007/s11219-014-9241-7, URL `http://dx.doi.org/10.1007/s11219-014-9241-7`

Mann HB, Whitney DR (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Ann of Math Stat 18(1):pp. 50–60

McCabe TJ (1976) A complexity measure. IEEE Transactions on Software Engineering 2(4):308–320

Meneely A, Williams L, Snipes W, Osborne J (2008) Predicting failures with developer networks and social network analysis. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, New York, NY, USA, SIGSOFT '08/FSE-16, pp 13–23, DOI 10.1145/1453101.1453106, URL `http://doi.acm.org/10.1145/1453101.1453106`

Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y (2008) Implications of Ceiling Effects in Defect Predictors. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE), ACM

Menzies T, Butcher A, Marcus A, Zimmermann T, Cok D (2011) Local vs. global models for effort estimation and defect prediction. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE Computer Society

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

Menzies T, Pape C, Steele C (2014) tera-promise. http://openscience.us/repo/

Nam J, Kim S (2015) Heterogeneous defect prediction. In: Proceedings of the 10th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), DOI 10.1145/2786805.2786814, URL http://doi.acm.org/10.1145/2786805.2786814

Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: Proceedings of the 35th International Conference on Software Engineering (ICSE)

Ngomo AcN (2009) Low-Bias Extraction of Domain-Specific Concepts. Ph.D. Thesis

Pan SJ, Yang Q (2010) A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering 22(10):1345–1359

Peters F, Menzies T, Gong L, Zhang H (2013) Balancing privacy and utility in cross-company defect prediction. IEEE Transactions on Software Engineering 39(8):1054–1068

Peters F, Menzies T, Layman L (2015) LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)

Premraj R, Herzig K (2011) Network versus code metrics to predict defects: A replication study. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)

Rahman F, Posnett D, Devanbu P (2012) Recalling the "imprecision" of cross-project defect prediction. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE), ACM

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2):131–164

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 (ASE), IEEE Computer Society

Schikuta E, Schikuta E (1993) Grid-Clustering: A Hierarchical Clustering Method for Very Large Data Sets. In: Proceedings of the 15th International Conference on Pattern Recognition

Schölkopf B, Smola AJ (2002) Learning with Kernels. MIT Press

Scott AJ, Knott M (1974) A cluster analysis method for grouping means in the analysis of variance. Biometrics 30(3):pp. 507–512

Shepperd M, Song Q, Sun Z, Mair C (2013) Data Quality: Some Comments on the NASA Software Defect Datasets. IEEE Transactions on Software Engineering 39(9):1208–1215

Shull F, Carver J, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empirical Software Engineering 13(2):211–218

Siegmund J, Siegmund N, Apel S (2015) Views on Internal and External Validity in Empirical Software Engineering. In: 37th International Conference

on Software Engineering
Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalanced data. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)
Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto Ki (2015) The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering, ACM, DOI 10.1145/2884781.2884857
Turhan B, Menzies T, Bener A, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 14:540–578
Watanabe S, Kaiya H, Kaijiri K (2008) Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE), ACM
Xu R, Wunsch I D (2005) Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3):645–678
Zhang F, Mockus A, Keivanloo I, Zou Y (2014) Towards Building a Universal Defect Prediction Model. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), ACM
Zhang F, Mockus A, Keivanloo I, Zou Y (2015) Towards building a universal defect prediction model with rank transformed predictors. Empirical Software Engineering pp 1–39, DOI 10.1007/s10664-015-9396-2, URL `http://dx.doi.org/10.1007/s10664-015-9396-2`
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 European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ACM, pp 91–100

A Metrics

A.1 JSTAT Data

The following metrics are part of the JSTAT data:

- WMC: weighted method count, number of methods in a class
- DIT: depth of inheritance tree
- NOC: number of children
- CBO: coupling between objects, number of classes coupled to a class
- RFC: response for class, number of different methods that can be executed if the class receives a message
- LCOM: lack of cohesion in methods, number of methods not related through the sharing of some of the class fields

- LCOM3: lack of cohesion in methods after Henderson-Sellers (1996)
- NPM: number of public methods
- DAM: data access metric, ratio of private (protected) attributes to total number of attributes in the class
- MOA: measure of aggregation, number of class fields whose types are user defined classes
- MFA: measure of functional abstraction, ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class
- CAM: cohesion among methods of class, relatedness of methods based upon the parameter list of the methods
- IC: inheritance coupling, number of parent classes to which the class is coupled
- CBM: coupling between methods, number of new/redefined methods to which all the inherited methods are coupled
- AMC: average method complexity
- Ca: afferent couplings
- Ce: efferent couplings
- CC: cyclomatic complexity
- Max(CC): maximum cyclomatic complexity among methods
- Avg(CC): average cyclomatic complexity among methods

For a detailed explanation see Jureczko and Madeyski (2010).

A.2 MDP Data

The following metrics are part of the MDP data. This is the common subset of metrics that is obtained by all projects within the MDP data set:

- LOC_TOTAL: total lines of code
- LOC_EXECUTABLE: exectuable lines of code
- LOC_COMMENTS: lines of comments
- LOC_CODE_AND_COMMENT: lines with comments or code
- NUM_UNIQUE_OPERATORS: number of unique operators
- NUM_UNIQUE_OPERANDS: number of unique operands
- NUM_OPERATORS: total number of operators
- NUM_OPERANDS: total number of operands
- HALSTEAD_VOLUME: Halstead volume (see Halstead (1977))
- HALSTEAD_LENGTH: Halstead length (see Halstead (1977))
- HALSTEAD_DIFFICULTY: Halstead difficulty (see Halstead (1977))
- HALSTEAD_EFFORT: Halstead effort (see Halstead (1977))
- HALSTEAD_ERROR_EST: Halstead Error, also known as Halstead Bug ((see Halstead (1977)))
- HALSTEAD_PROG_TIME: Halstead Pro
- BRANCH_COUNT: Number of branches
- CYCLOMATIC_COMPLEXITY: Cyclomatic complexity (same as CC in the JSTAT data)
- DESIGN_COMPLEXITY: design complexity

A.3 JPROC Data

The following metrics are part of the JPROC data:

- CBO: coupling between objects
- DIT: depth of inheritance tree
- fanIn: number of other classes that reference the class
- fanOut: number of other classes referenced by the class
- LCOM: lack of cohesion in methods
- NOC: number of children
- RFC: response for class

- WMC: weighted method count
- NOA: number of attributes
- NOAI: number of attributes inherited
- LOC: lines of code
- NOM: number of methods
- NOMI: number of methods inherited
- NOPRA: number of private attributes
- NOPRM: number of private methods
- NOPA: number of public attributes
- NOPM: number of public methods
- NR: number of revisions
- NREF: number of times the file has been refactored
- NAUTH: number of authors
- LADD: sum of lines added
- max(LADD): maximum lines added
- avg(LADD): average lines added
- LDEL: sum of lines removed
- max(LDEL): maximum lines deleted
- avg(LDEL): average lines deleted
- CHURN: sum of code churn
- max(CHURN): maximum code churn
- avg(CHURN): average code churn
- AGE: age of the file
- WAGE: weighted age of the file

For a detailed explanation see D'Ambros et al (2010).