Training data selection for cross-project defect prediction

Steffen Herbold
Institute of Computer Science, University of Göttingen
Goldschmidtstr. 7, 37077 Göttingen, Germany
herbold@cs.uni-goettingen.de

ABSTRACT

Software defect prediction has been a popular research topic in recent years and is considered as a means for the optimization of quality assurance activities. Defect prediction can be done in a within-project or a cross-project scenario. The within-project scenario produces results with a very high quality, but requires historic data of the project, which is often not available. For the cross-project prediction, the data availability is not an issue as data from other projects is readily available, e.g., in repositories like PROMISE. However, the quality of the defect prediction results is too low for practical use. Recent research showed that the selection of appropriate training data can improve the quality of cross-project defect predictions. In this paper, we propose distance-based strategies for the selection of training data based on distributional characteristics of the available data. We evaluate the proposed strategies in a large case study with 44 data sets obtained from 14 open source projects. Our results show that our training data selection strategy improves the achieved success rate of cross-project defect predictions significantly. However, the quality of the results still cannot compete with within-project defect prediction.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*defect prediction*; I.2.6 [**Artificial Intelligence**]: Learning—*machine learning*

General Terms

Experimentation, Algorithms

Keywords

machine learning, defect-prediction, cross-project prediction

1. INTRODUCTION

The prediction of defect-prone software modules based on software metrics through means of machine learning has been a popular research topic in recent years, e.g., [16, 15, 13, 20]. The results of the defect prediction are a means for the optimization of quality assurance resources and, thereby, improve the efficiency of the quality assurance activities within software projects. For example, inspections of defect-prone modules can be scheduled. The results of the conducted research indicate that it is possible to detect defects with a relatively high accuracy, if historical data from the same context, i.e., the same project is available. This approach is referred to as *within-project* defect prediction. However, within-project prediction has the major drawback that historic data about defects in the project must be available, which is often not the case in practical scenarios.

To work around this issue, researchers turned toward *cross-project* defect prediction. Here, the intent is to predict defect-prone software modules based on data collect from other software projects. Such data can either be collected from finished projects or taken from a repository like PROMISE [12]. The reliable prediction of defect-prone modules with cross-project data is still an open issue. Zimmermann *et al.* [21] determined that only 3.4% of all cross-project defect predictions are successful. This number seems devastatingly low. But it only means that cross-project defect prediction is a difficult problem. Strategies are required that do not perform cross-project defect prediction in any setting, but rather in the settings of the 3.4% where it has been shown to be successful. Additionally, the strategies for the handling of training data as well as predictor models may need to be different than for within-project prediction.

One major issue is the selection of appropriate training data. A study by He *et al.* [9] found out that with suitable training data, the success rate of cross-project defect prediction can be drastically improved to 52.9%[1]. However, in order to achieve this success rate, they select the training data with a posteriori knowledge based on the results from the test set. Hence, the 52.9% should be seen as an upper bound on the possible success rate. He *et al.* also provide a method for the selection of training data. However, run time of this method is exponential and does not scale with large data sets. An efficient solution for the training data selection is an open issue of their work.

Research Problem: The selection of training data with a high success for cross-project defect prediction in a feasible run time.

The aim of our work is the provision of a strategy for the training data selection that solves this problem. The foundation of our strategy is another result of He *et al.* [9]. They showed that there exists a correlation between the distributional characteristics of the training data, the test data, and successful cross-project defect prediction. Building on this, we use distributional characteristics as means to

[1]It is worth noting that He *et al.* used a weaker criterion to determine the success of the prediction than Zimmermann *et al.* Moreover, both research teams used different data for their experiments, which may also lead to a difference in the results.

determine the similarity of data sets. We propose two distance-based strategies for the training data selection, based on the similarity as distance. The first is inspired by a result by Jureczko and Madeyski [10]. They found that clusters can be found within sets of defect prediction data. Based on this, we suggest to select the training data with the EM clustering algorithm [7]. In our second strategy, we propose to choose the data sets that are most similar to the target project as training data. We determine these sets with the *k*-nearest neighbor algorithm. The runtime of both algorithms scales well, even for very large data sets. Hence, the feasibility of the run time is guaranteed with our approach. Therefore, we focus on the following research questions, that analyze the success of the defect predictions.

RQ1 How successful are predictors trained with our training data selection? **The achieved success rate is improved to 0.18, but requires further improvements. However, the mean recall of 0.76 is very high.**

RQ2 How does our training data selection perform in comparison to training with all available data? **Our training data selection outperforms training with all available data and has a 0.09 higher success rate.**

RQ3 How does our training data selection perform in comparison to within-project prediction? **Cross-project prediction with our training data selection is outperformed by within-project prediction and has a 0.19 lower success rate. However, the recall of the cross-project prediction is 0.19 higher than for within-project prediction.**

In our experiments, we utilize a set of learning algorithms that covers many the important predictor classes in machine learning (see Section 4.1.3). We use the results of the different predictors to answer the following research question.

RQ4 Which predictor model performs best in cross-project defect prediction? **A Support Vector Machine (SVM) with an Radial Basis Function (RBF) kernel in combination with a strategy to deal with imbalanced training data yields the best results.**

Finally, based on the performance of training data selection strategies for the best predictor models, we suggest how to apply our approach by answering the following research question.

RQ5 Which training data selection strategy should be used? **The nearest neighborhood strategy with a neighborhood size of 50% to 70% of the available data should be used.**

The remainder of this paper is structured as follows. In Section 2, we discuss the related work. Afterwards, we present our strategies for the training data selection in Section 3. We evaluate these strategies in a case study in Section 4. In Section 5, we discuss our findings, before we analyze the threats to their validity in Section 6. Finally, we summarize our findings and give an outlook on future work to conclude our paper in Section 7.

2. RELATED WORK

To our knowledge, the first work on cross-project defect prediction was performed by Basili *et al.* [2]. They predicted the defects of an open source project based on a logistic regression predictor trained with data from another open source project. However, both projects were developed by the same team, which is an assumption that usually does not hold for cross-project defect prediction. Additionally, their scenario does not include data about multiple projects, which makes the selection of appropriate data sets for the training a moot point.

Zimmermann *et al.* [21] conducted a large study to determine the feasibility and challenges of cross-project defect prediction. They determined that if two different projects are used to predict the defects of each other, the success rate is only about 3.4%. Additionally, they determined that the relationship between data sets for the prediction is not commutative, i.e., if data set A is suitable to train a successful predictor for data set B, it does not follow B can be used to train a good predictor for A. They did not analyze the capability of multiple projects as training data, they only considered training based on a single data set.

Carmago Cruz and Ochimizu [5] propose not to simply reuse data for cross-project defect prediction, but to transform the data such that the underlying distributions are similar. They observe that such a transformation improves the quality of the predictions. Similarly, Watanabe *et al.* [19] used the transformation of metric values to train a defect predictor for a C++ project with data obtained from a Java project. They observe that the transformation is able to increase the recall of defect prediction models used in a different setting. While both approaches are different from our work, the underlying assumption is the same as ours: similar distributions lead to better results for of cross-project predictions. However, we do not transform the data. Rather, we try to select data that is already similar.

An approach similar to our work was used by Turhan *et al.* [18]. They used data from 10 different projects to analyze the capabilities of cross-project defect prediction. Similar to our work, they provide a selection strategy for the training data based on the *k*-nearest neighbor algorithm. They apply the algorithm in a point-wise manner: for each instance in the target project, they select the 10 instances of the available data that are closest to the target instance for the training data. This is inherently different from our approach. Instead of point-wise selection, we use distributional characteristics to select complete data sets from a larger sample of data sets.

Our work is mostly based on the research conducted by He *et al.* [9]. They determined that if the best possible training set of three software projects from a set of available projects is selected via a brute force strategy with a posteriori knowledge, it is possible to achieve a defect prediction success rate of over 50%. Additionally, they analyzed the relationship between distributional characteristics of training and test data. They determined that there is a correlation between the distributional characteristics and successful defect predictions. Furthermore, they provide a strategy for the selection of training data with decision trees over the distributional characteristics. The nature of their method is exponential and does not scale with the number of data sets. In comparison, while the training data selection strategies we provide are also based on the distributional characteristics, they scale with the number of training data sets and are, therefore, also applicable to large data sets.

A different approach is the training of local prediction models. All of the approaches discussed above always consider the complete training and test data for the training and evaluation of prediction models. Menzies *et al.* [11] propose to split the data in order to obtain clusters whose data is very similar. Then, they train a local prediction model for each cluster. Their results are promising and show that these local models are often superior to global models. Bettenburg *et al.* [3] conducted a comparison study between local and global prediction models. They found that local models have a slight advantage over global models in terms of prediction performance. However, they also state that these small advantages offset on average and they, therefore, propose to use global models for defect prediction. The local prediction is related to our approach. We select a "local" cluster or neighborhood of training data with regard to distributional characteristics. However, same as with [18], our notion of locality is on the level of data sets and not point-wise.

All of the above works evaluate the performance of predictors solely based on the prediction accuracy, through measures like recall, precision, and F-score. Recently, Rahman *et al.* [17] proposed a different measure: the cost effectiveness. They submit that even if the precision of cross-project predictors is low, the quality of the predictions is often still sufficient to improve the effectivity of quality assurance efforts if only a subset of the complete software product can be inspected. Based on this, Canfora *et al.* [6] propose a multi-objective prediction model, that allows a trade-off between cost effectiveness and recall of defect predictions. In our work, we do not use the cost effectiveness as a performance measure for reasons stated in Section 4.2.

3. CROSS-PROJECT TRAINING DATA SELECTION STRATEGIES

To understand the general idea of how our training data selection looks like, knowledge about the structure of metric-based defect prediction data is required. The aim is to predict which entities within a software product S are defect-prone. In our case, these entities are Java classes, but other levels of granularity are also possible, e.g., complete components. Then, the entities are measured with metrics $m_1, \dots, m_p$. For our purposes, it is sufficient to consider software metrics as functions that assign numerical values to software entities, i.e., $m_i : S \to \mathbb{R}, i = 1, \dots, p$. From the measurement, we get a p-tuple $(m_1(s), \dots, m_p(s))$ for each software entity $s \in S$.

In order to train a predictor, knowledge about which entities are defect-prone is required. This is modeled as a function $f : S \to \{0,1\}$, such that $f(s) = 1$ means that the software module s is defect-prone and $f(s) = 0$ means that it is not defect-prone. Through the combination of the metric data and the function f, we get the data that we require for the training

$$train(S) = \{(m_1(s), \dots, m_p(s), f(s)) \in \mathbb{R}^p \times \{0,1\} : s \in S\}.$$

In our concrete scenario, we are in the following situation. We assume that training data (i.e., the metric data and the classifications) for n software products is available, i.e., $S^{candidates} = \{S^1, \dots, S^n\}$ and $train(S)$ is known for all $S \in S^{candidates}$. We want to build a defect prediction model for software product S^{target} of which only the metric data is known. The open research problem we address in this paper is, which training data sets $S^{train} \subseteq S^{candidates}$ should be used to train a defect predictor for S^{target}?

Our approach is to select training data whose metric data is structurally similar to S^{target}. We determine the similarity of the metric data based on distributional characteristics, e.g., the mean value, the median, and the standard deviation. Our approach and the selected notion of similarity are based on the findings of He *et al.* [9], who determined that there is a correlation between the distributional characteristics of metric data and the suitability of training data.

Let $C = \{c_1, \dots, c_q\}$ be distributional characteristics. We use the notation $c(m(S))$ to denote the distributional characteristic c of the metric m for the software entities S. We get for each software product S a vector of distributional characteristics

$$C(S) = (c_1(m_1(S))), \dots, c_1(m_p(S)), \dots, c_q(m_1(S)), \dots, c_q(m_p(S))).$$

In the remainder of this work, we refer to $C(S)$ as the characteristic vector of a software product S. The characteristic vector is what we use for our training data selection. Note, that it does not contain information about the defect-proneness of the entities. Hence, the characteristic vector is also available, if no knowledge about the defects is available, which means it is also available for the target project.

In the next paragraphs, we define two strategies for the selection of training data based on the characteristic vectors. Before we apply our training data selection strategies, we normalize the observed values of all distributional characteristics c_i to the interval $[0,1]$. This effectively removes the scales from the characteristics and, hence, the negative impact different scales have on a distance-based approach.

3.1 Strategy 1: EM-Clustering

The first strategy is based on the idea of creating clusters of software entities. Jureczko and Madeyski [10] determined that meaningful clusters of software entities with respect to defect prediction exist and that, among others, the EM clustering algorithm [7] can be used to determine such clusters. From this concept, we derived our first training data selection algorithm. We use the EM algorithm to create clusters of the characteristic vectors of the candidate training data joined with the characteristic vector of the target data. Then, we select those data sets as training data, that are located in the same cluster as the target training data (Algorithm 1).

Algorithm 1: EM clustering training data selection.

```
Input: Candidate data sets $S^{candidate}$, target data set $S^{target}$
$S^{train} \leftarrow \emptyset$;
$C^{all} \leftarrow \{C(S) : S \in S^{candidate} \bigcup \{S^{target}\}\}$;
$EMCluster.createClusters(C^{all})$;
$C^{train} \leftarrow EMCluster.getCluster(C(S^{target}))$;
$S^{train} \leftarrow \{S \in S^{candidate} : C(S) \in C^{train}\}$;
return $S^{train}$;
```

3.2 Strategy 2: Nearest Neighbor Selection

The second strategy is based on the rationale to use the candidate data whose distributional characteristics are the most similar to the target data. We use the euclidean distance of the characteristic vectors to determine these most similar candidates. Concretely, we select the k closest software projects using the k-nearest neighbor algorithm. Algorithm 2 formalizes this selection strategy.

Algorithm 2: k-nearest neighbor training data selection.

```
Input: Candidate data sets $S^{candidate}$, target data set $S^{target}$,
       number of desired training sets $k$
$S^{train} \leftarrow \emptyset$;
while $|S^{train}| < k$ do
    $S' \leftarrow \arg\min_{S \in S^{candidate}} dist(C(S), C(S^{target}))$;
    $S^{train} \leftarrow S^{train} \bigcup S'$;
    $S^{candidate} \leftarrow S^{candidate} \setminus S'$;
end
return $S^{train}$;
```

4. CASE STUDY

We evaluate our approach in a case study. We use the results of our case study to determine the answers for the research questions **RQ1-RQ5**.

Available Data	
44 versions of 14 software projects	
Distributional Characteristics	
Mean value	
Standard deviation	
Neighborhood size of the k-nearest neighbor strategy	
3, 5, 10, 15, 20, 25, 30	
Predictor models	
Logistic Regression	
Naive Bayes	
Bayesian Networks	
SVM with RBF kernel	
J48 Decision Trees	
Random Forest	
Multilayer Perceptron	
Training bias	
Unweighted	The instances of the training data are not weighted
Equally Weighted	The instances of the training data are weighted such that the sum of the weights of the defect-prone instances is equal to the sum of the none-defect-prone instances.

Table 1: Parameters of the experimental setup.

4.1 Data and Experiment Setup

The experimental setup consists of three parts: 1) the description and setup of the candidate training data and the test data; 2) the parameters of the training data selection; and 3) the predictor models applied. The parameters of the experimental setup are summarized in Table 1. In the following, we explain the parameters as well as the rationale for their selection.

4.1.1 Data Setup

The data we use in our experiments is part of the PROMISE repository [12] and has been collected by Jureczko and Madeyski [10]. It consists of 71 versions of 38 different open source software development projects and is a superset of the data used by [9]. The data also contains 6 versions of proprietary projects. We discarded the proprietary projects, because we did not want the difference between open source and proprietary development to influence our results. Jureczko and Madeyski used ckjm[2] to calculate the values of 20 software metrics for each class in the projects. A list with all 20 metrics as well as their descriptions is provided by Jureczko *et al.* [10]. They used the tool BugInfo[3] to add information about the number of defects for each class. We consider a class defect-prone, if it had a defect. Hence, the data consists of a total of 21 attributes: 20 features in form of software metrics and a binary label for the defect proneness.

We filter the available data based on the size, in order to remove small projects. We removed all versions with less than 100 classes for our experiments. This leaves us with 44 versions of 14 software projects. Table 2 lists all the versions, including their total number of classes and the number of defect-prone classes.

For our experiments, we need to select target projects and appropriate candidate training data. As target project, we select each software version once, i.e., we repeat our approach for 44 different cross-project defect predictions. We set up the candidate training data based on the target project. The data often contains multiple versions of the same product, e.g., the versions 1.3–1.7 of *ant*. Because our evaluation focus is on the cross-project prediction, we set up the candidate training data, such that no other versions from the target product are part of the candidates. Hence, when we train a cross-project predictor for *ant 1.3*, the versions *ant 1.4–1.7* are not part of the candidate training data. We visualize this example for the data setup in Figure 1.

[2]http://www.spinellis.gr/sw/ckjm/

[3]https://kenai.com/api/projects/buginfo/

Project	#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%
ant 1.7	745	166	22%
arc	234	27	12%
camel 1.0	339	13	4%
camel 1.2	608	216	36%
camel 1.4	872	145	17&
camel 1.6	965	188	19%
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%
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%
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%
synapse 1.0	157	16	10%
synapse 1.1	222	60	27%
synapse 1.2	256	86	34%
tomcat	858	77	9%
velocity 1.4	196	147	75%
velocity 1.5	214	142	66%
velocity 1.6	220	78	35%
xalan 2.4	723	110	15%
xalan 2.5	803	387	48%
xalan 2.5	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%
Total	17043	5859	34%

Table 2: Software projects that were part of the study, including their total number of classes and the number of defect-prone classes.

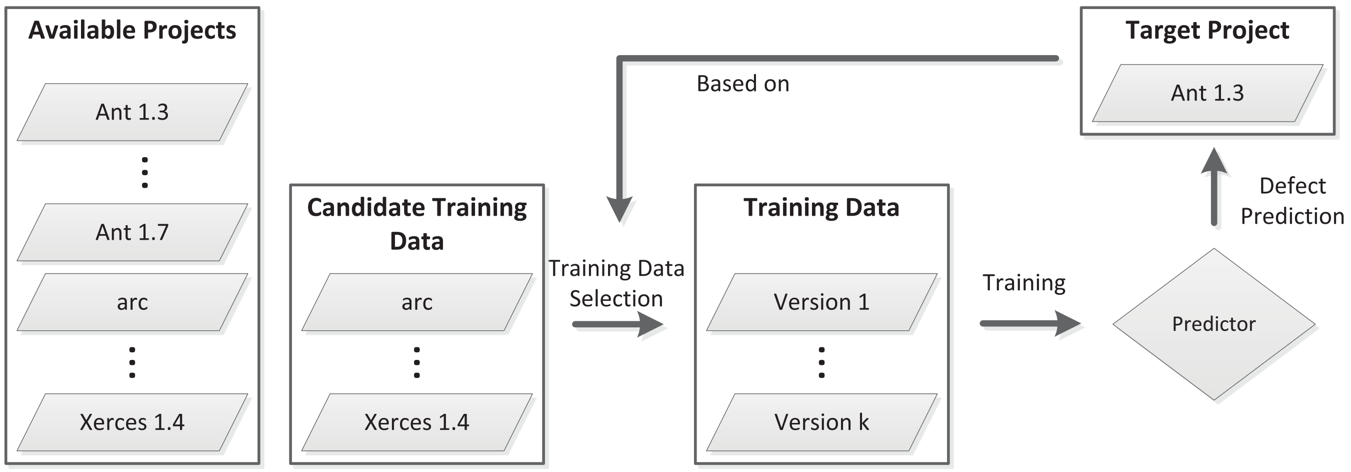


Figure 1: Setup up of the candidate training data $S^{candidates}$ for a target project S^{target}.

4.1.2 *Training Data Selection Parameters*

In the description of our approach in Section 3, we only defined that the training data is selected based on distributional characteristics C, without yet defining a concrete set of characteristics. For their analysis towards the correlation of distributional characteristics of data and defect prediction capabilities, He *et al.* [9] used a total of 16 characteristics. Through their analysis, they could not point out particular distributional characteristics that are good indicators for the defect prediction capabilities. In distance-based selection approaches, like the ones we propose, it is not feasible to use a large set of distributional characteristics. The reason for this is, that the dimension of the characteristic vectors is $\#metrics \cdot \#characteristics$ and gets very large very fast, which is problematic for distance-based techniques, because common distance measures are meaningless in very high dimensional spaces [4, 1]. Therefore, we only use two distributional characteristics: the *mean value* and the *standard deviation*. These two characteristics describe the behavior of the metric values accurately under the assumption that they are following a gaussian distribution.

For the training data selection strategy 1, i.e., the EM-clustering algorithm, we do not have to define any additional parameters. For the training data selection strategy 2, i.e., the nearest neighbor algorithm, we have to select the number of neighbors k we select from the candidate data. We choose a neighborhood size of 3 for comparison with the brute-force training data selection based on the best possible result on the test data by He *et al.* [9] and additionally the sizes 5, 10, 15, 20, 25, and 30 to determine the impact of the neighborhood size on the results.

4.1.3 *Predictor Models*

In our experiment, we choose not to focus on a single predictor model that has been successfully used in the past, but rather use a set of successful predictor models. We use the following 7 predictor models:

- Logistic Regression
- Naive Bayes
- Bayesian Networks
- SVM with RBF kernel
- C4.5 Decision Trees
- Random Forest
- Multilayer Perceptron

For each of these predictor models, we use the implementations provided by WEKA [8] and do not change their default parameters. This prevents overfitting of the algorithm parameters to our experimental setup, but also leads to a possible underestimation of our approach's performance.

The training data is mostly imbalanced with 34% of all available instances classified as defect-prone, which is significantly less than about 50% you would have in balanced data (see Table 2). Imbalanced data usually leads to biased classifiers that favor the class that occurs more often. This means that a bias towards predictions as not defect-prone is likely. However, we are interested in finding the defect-prone instances, i.e., the opposite of the bias. We propose a strategy based on weights to deal with this problem. Our approach is that not each instance has the same weight in the training. Instead, we assign weights to all instances, such that the defect-prone and the not defect-prone instances have the same overall weight in order to offset any bias. Hence, we use two different configurations of the training data.

- Unweighted: the instances of the training data are used without any weighting. This configuration has the advantage that the training data is not modified. However, it ignores the likely bias towards the not defect-prone instances.

- Equally Weighted: the instances of the training data are weighted such that the total weight of the defect-prone instances equals the total weight of the not defect-prone instances. This reduces the possibility of a bias by predictors towards not defect-prone instances.

For the weighting, we use a built-in mechanism of WEKA to assign weights to instances. All of the training algorithms respect these weights during the training of predictors.

4.2 Evaluation Criteria

We evaluate the success of our experiments based on the measures *recall* and *precision*, which are defined as

$$recall = \frac{tp}{tp+fn}$$

$$precision = \frac{tp}{tp+fp}$$

where tp is the number of *true positive* predictions, fp is the number of false positive predictions, and fn is the number of false negative predictions. Analogous to He *et al.* [9], we consider a prediction successful, if the *recall* is at least 0.7 and the *precision* is at least 0.5. We evaluate **RQ1** based on the *success rate*

$$success\ rate = \frac{\#successful\_experiments}{\#experiments},$$

i.e., the fraction of the successful predictions.

Recent works suggested the use of the *cost of inspection* as a measure for the efficiency of defect prediction rather than the precision [17, 6]. The cost of the inspection is measured in terms of the Lines Of Code (LOC) of the instances predicted as defect-prone. While it is a valid approach to analyze the capabilities of cross-project defect-prediction in terms of efficiency rather than precision, we argue that using the cost of inspection oversimplifies the efficiency. If only the cost of inspection is used as a measure for the efficiency of the defect prediction, other factors that are at least as important for the cost efficiency are ignored, e.g., the severity of the detected defects, the costs that occur if defects cause problems after the release, and the cost to fix defects. Therefore, we argue that the precision as an indirect measure for the efficiency is a better choice. The precision weights each found defect equally, while the cost of the inspection favors defects found in small entities and ignores the other factors that influence the cost efficiency.

For the evaluation of **RQ2**, we train predictors based on all the available candidate training data, without applying our training data selection strategies. We then compare the recall and precision with the results we obtain with our selection strategies in order to answer **RQ2**. Similarly, we evaluate **RQ3** by building a within-project predictor and determine its recall and precision based on the data of the actual target project and 10x10 stratified cross-validation. Then we compare the results with the results from our strategy.

We evaluate **RQ4** based on the results from the different predictor models. We say that one algorithms outperforms the others, if it has a significantly better performance on a consistent basis throughout our experiments. Similarly, we evaluate **RQ5** and suggest that the training data selection strategies that yield the best results in the mean should be used.

4.3 Results

We graphically depict the success rate, the mean recall, and the mean precision of the defect prediction in Figure 2. We show the results of the experiments without applying any weights in the left column of the figure and the results of the equally weighted scenario in the right column. We use the following notations in our results figures and tables:

- *EM* to denote the EM clustering training data selection;
- *NN-k* to denote the nearest neighbor strategy with a neighborhood size of *k*;
- *All* to denote the training with all available candidate data (excluding other versions of the same project); and
- *WP* to denote the within-project prediction results obtained with cross-validation.

The results of our experiments show that the predictor models perform quite differently, especially with respect to the equally weighted and the unweighted bias. In case we use the training data without applying any weight, the bayesian network outperforms all other predictor models in terms of success rate and mean recall, regardsless of the concrete training data that is used. In terms of precision, the bayesian network is in the middle of the field. The SVM is the worst predictor model in the unweighted scenario, for most training data not even a mean recall or precision of 0.2 is achieved. The other predictors perform quite similar to each other. Their success rate, mean recall, and mean precision is between the SVM and the bayesian network.

In the equally weighted scenario, the results change. Here, the SVM performs best in terms of success rate and mean recall. The bayesian network is only the second best predictor model. The naive bayes predictor yields the best precision, the precision of all other predictor models is about the same. However, naive bayes has the worst recall of all predictor models and no cross-project predictions are successful with naive bayes. As for the relationship between the unweighted and the equally weighted scenario, the results are consistently and significantly better in the equally weighted scenario.

For the remainder of this results presentation, we focus on the overall best scenario in our cross-project prediction, i.e., the SVM with equally weighted training data. Table 3 depicts the concrete values of the success rate, mean recall, and precision of the SVM. The results show, that the within-project prediction has the largest success rate with 0.37, followed by the nearest neighbor strategy with rather large neighborhoods (20, 25, and 30 neighbors) with a success rate of 0.18. The prediction with all possible data only reaches a success rate of 0.09 and is in the bottom three of all training configurations. As for the recall and precision, our findings are similar to what other researchers (e.g., Turhan et *et al.* [18], He *et al.* [9]) observed. The cross-project prediction performs better than within project predictions in terms of recall, but the precision of the predictions is worse. The nearest neighbor strategy with large neighborhoods has the best recall, with 0.76 for neighborhood sizes 25 and 30. In comparison, the within-project prediction has a recall of 0.57 and the training with all data a recall of 0.64. The precision of the within-project prediction is with the best at 0.63. For all cross-project predictions, the precision is quite similar and between 0.39 and 0.44.

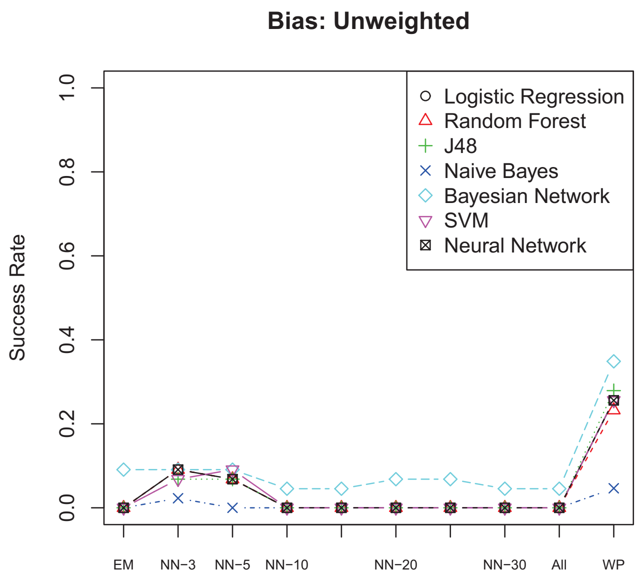
Table 4 shows the comparison of our training data selection strategies with the within-project prediction and Table 5 shows the comparison between cross-project defect prediction with and without training data selection.

5. DISCUSSION

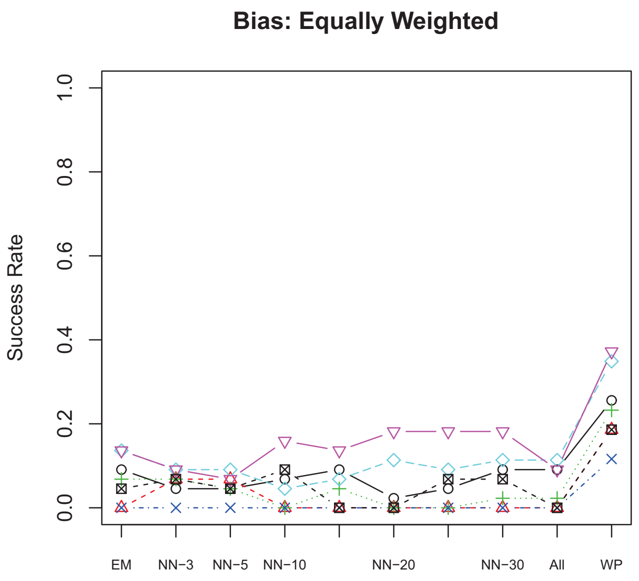
Based on our experiment results, we answer our research questions as follows.

RQ1: How successful are predictors trained with our training data selection? We achieve some success with our training data selection. We improve the success rate of the cross-project defect prediction to 0.18. Additionally, our training data selection yields very high values for the recall with a mean recall of 0.76. However, the problem with the precision other researchers observed, e.g., [18, 9], is still not resolved. The mean precision is only 0.39 in the results with the highest success rate. Hence, our answer to this research question is the following. **The achieved success rate is improved to 0.18, but requires further improvements. However, the recall is with 0.76 in the mean very high.**

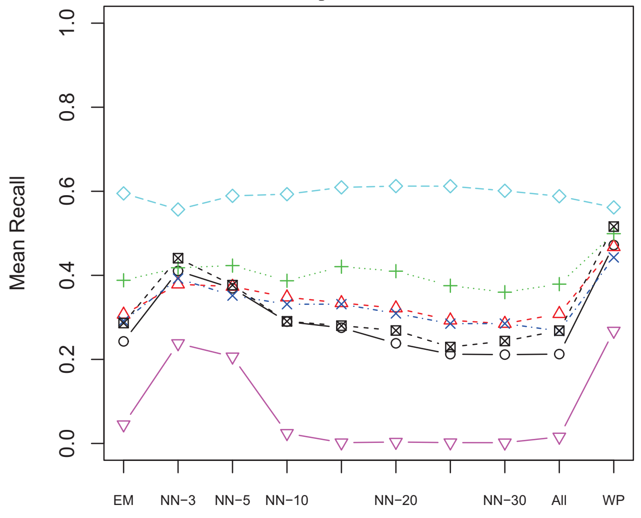
RQ2: How does our training data selection perform in comparison to training with all available data? Our training data



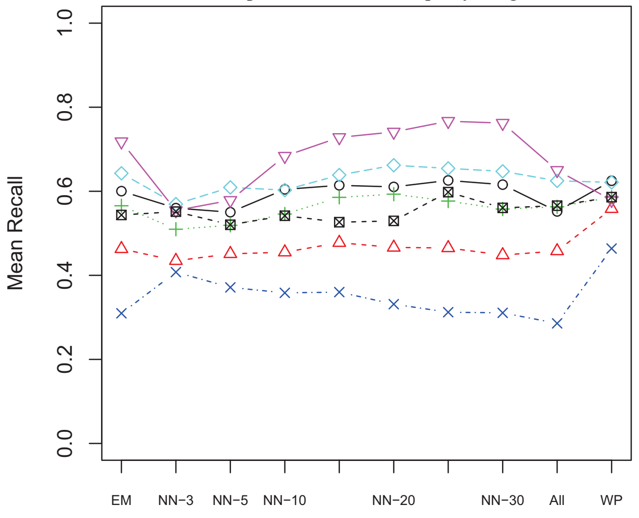
(a) Success rate of the experiments without a bias.



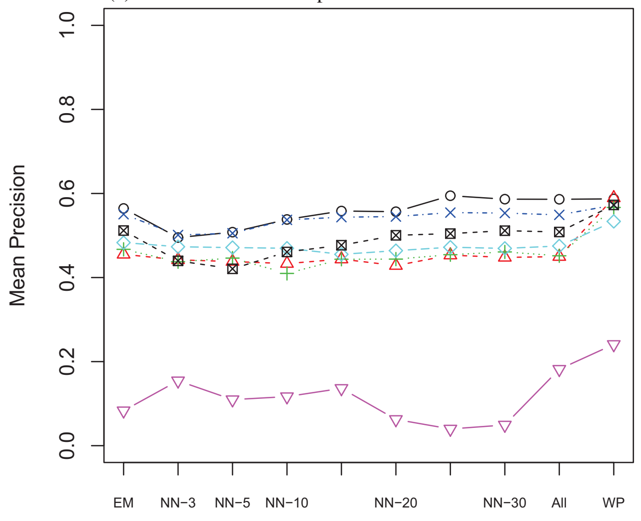
(b) Success rate of the experiments with the equally weighted bias.



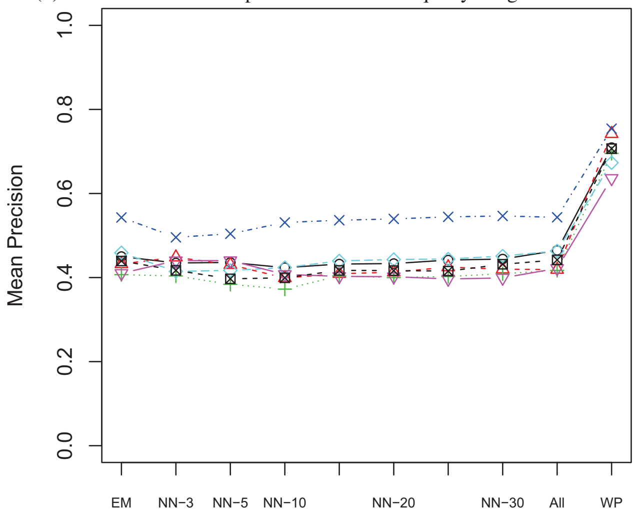
(c) Mean recall of the experiments without a bias.



(d) Mean recall of the experiments with the equally weighted bias.



(e) Mean precision of the experiments without a bias.



(f) Mean precision of the experiments with the equally weighted bias.

Figure 2: Results of the experiments. The figures show the success rate, mean recall, and mean precision of the experiments with and without bias, respectively. The mean values are calculated over the 44 cross-project predictions we conducted.

	WP	All	EM	NN-3	NN-5	NN-10	NN-15	NN-20	NN-25	NN-30
Success Rate	0.37	0.09	0.13	0.09	0.06	0.15	0.13	0.18	0.18	0.18
Mean Recall	0.57	0.64	0.71	0.55	0.57	0.68	0.72	0.74	0.76	0.76
Mean Precision	0.63	0.42	0.40	0.44	0.44	0.40	0.40	0.40	0.39	0.39

Table 3: Success rate, mean recall, and mean precision of the SVM with equally weighted training data.

	EM	NN-3	NN-5	NN-10	NN-15	NN-20	NN-25	NN-30
Success Rate	-0.24	-0.28	-0.31	-0.22	-0.24	-0.19	-0.19	-0.19
Mean Recall	+0.14	-0.02	0.00	+0.11	+0.15	+0.17	+0.19	+0.19
Mean Precision	-0.23	-0.19	-0.19	-0.23	-0.23	-0.23	-0.24	-0.24

Table 4: Change of the success rate, mean recall, and mean precision of the SVM with equally weighted training data with respect to the within-project prediction.

selection outperforms the training with all possible data. The success rate with all possible data is 0.09, whereas we reach a success rate of 0.18 with our training data selection, i.e., we achieve an improvement of 0.09. The main improvement is the recall, which is 0.12 higher than with all data. For the precision, there is a slight drop of 0.03, which does not impact the success rate negatively. Hence, our answer to this research question is the following. **Our training data selection outperforms training with all available data and has a 0.09 higher success rate.**

RQ3: How does our training data selection perform in comparison to within-project prediction? The success of the cross-project prediction is significantly lower than the within-project prediction. In the best case, we achieved a success rate of 0.18 with the cross-project prediction, while we were able to achieve a success rate of 0.37 in the within-project experiment with cross-validation. This is a difference of 0.19 in the success rate that might still be improved with a better strategy for the cross-project prediction. The reason for the lower success rate is that the precision is significantly worse: the SVM achieves a precision of 0.63 for the within-project prediction and only between 0.39 and 0.44 for the cross-project prediction (depending on the data selection strategy). The recall, on the other hand, is better in the cross-project prediction. The within-project prediction only achieves a recall of 0.57, while the cross-project prediction achieves a recall of up to 0.76. This is trade-off between precision and recall has also been observed in previous studies [18, 9]. Hence, our answer to this research question is the following. **Cross-project prediction with our training data selection is outperformed by within-project prediction and has a 0.19 lower success rate. However, the recall of the cross-project prediction is 0.19 higher than for within-project prediction.**

RQ4: Which predictor model performs best in cross-project defect prediction? The best predictor model depends on the handling of the training data. If the training data is used without any pre-processing regarding its balancing and can, therefore, contain a strong bias towards not defect-prone instances, only the bayesian network yields successful results for the defect prediction on a consistent basis, regardless of the training data selection. However, our results clearly show that using a strategy to set of a possible bias towards non-defect-prone instances yields better results for all predictor models.

If such a strategy is in place, the SVM yields the best performance. Especially in terms of recall, there is a gap between the SVM and the second best predictor model, the bayesian network. This also leads to a higher success rate for the SVM. Hence, our answer to this research question is the following. **A SVM with an RBF kernel in combination with a strategy to deal with imbalanced training data yields the best results.**

RQ5: Which training data selection strategy should be used? The results show that the defect prediction performs best with the nearest neighbor strategy with large neighborhoods. Concretely, we achieved the best overall result with the neighborhood size 25, closely followed by 30 and 20. From this, we derive that the neighborhood size should be between 50% to 70% of the available candidate data. The EM clustering strategy performs worse than the best nearest neighbor configurations (0.05 lower success rate, 0.05 lower recall). In general, the success rate of the EM clustering strategy is lower then the nearest neighbor for five of the seven neighborhood sizes. Only the small neighborhood sizes 3 and 5 perform worse than the EM strategy. Hence, our answer to this research question is the following. **The nearest neighborhood strategy with a neighborhood size of 50% to 70% of the available data should be used.**

6. THREATS TO VALIDITY

There are several threats to the validity of our findings. We only conducted our experiments on data from open-source projects written with Java. We did not consider proprietary projects or projects implemented on a platform other than Java. This may impact the generalizability of our findings. Additionally, we only verified the neighborhood sizes we proposed with one set of available projects. Our findings may be local and depend on the number of available projects.

We used the *mean* and *standard deviation* as the only distributional characteristics in our experiments. Our strategies may behave differently if other characteristics are used, e.g., the *median*, *variance*, and *harmonic mean*.

We used a recall of at least 0.7 and a precision of at least 0.5 to define a successful defect prediction. Other definitions are also possible, which may change our observations and conclusions regarding the success of the predictions.

The data we used only contains twenty static code features, i.e., software metrics. The results may change if different features are used, e.g., different metrics.

Finally, the data itself might contain flaws, as the defects were identified through the analysis of comments in the source code versioning system. Jureczko and Madeyski [10] state that this method may make mistakes in the identification and assignment of defects.

7. CONCLUSION

In this work, we have shown that defect prediction training data selection for cross-project defect prediction is a valuable tool to improve the quality of the prediction results. We presented two strategies for the selection of training data based on EM clustering and the nearest neighbor algorithm with distributional characteristics as

	EM	NN-3	NN-5	NN-10	NN-15	NN-20	NN-25	NN-30
Success Rate	+0.04	0.00	-0.03	+0.06	+0.04	+0.09	+0.09	+0.09
Mean Recall	+0.07	-0.09	-0.14	+0.04	+0.08	+0.10	+0.12	+0.12
Mean Precision	-0.02	+0.02	+0.02	-0.02	-0.02	-0.02	-0.03	-0.03

Table 5: Change of the success rate, mean recall, and mean precision of the SVM with equally weighted training data with respect to using all available data for the training.

the foundation. Both strategies are very efficient and can be applied to very large data sets without scalability problems. Furthermore, we showed how weights can be used to successfully deal with biased data. We evaluated the strategies in a large case study with 44 versions of 14 software projects. We showed that the training data selection provides a significant improvement of the success rate of cross-project defect prediction, as well as an improvement of the recall of defects. However, the overall success rate is still to low for the practical application of cross-project predictions. This is similar to the findings of Zimmermann *et al.* [21], Turhan *et al.* [18], and He *et al.* [9].

Though our results are promising, further research is required to increase the success rate of the predictions. Future work will include:

- More training data selection and transformations. Our current approach works on the level of whole data sets and uses only distributional characteristics. The combination of data set selection and a point-wise strategy like the strategy proposed by Turhan *et al.* [18] could improve the efficiency. Additionally, transformation techniques, such as proposed by [5] can also be applied to increase the similarity between data sets and, thereby, the prediction performance.

- More case studies. We will study our approach in additional case studies, with different data sets and different features. We especially want to consider not only static code metrics, but also process metrics, e.g., the change history. Such metrics have been successfully applied in a variety of studies for within-project defect prediction, e.g., [14, 20].

- Local predictors. The application of local predictors has shown promising results in the studies by Menzies *et al.* [11] and Bettenburg *et al.* [3]. The combination of training data selection for data sets with local predictors may further improve the performance of cross-project defect predictors.

8. ACKNOWLEDGEMENTS

I want to thank Marian Jureczko and Lech Madeyski for the effort of collecting the data and the publication of the data in the PROMISE repository. Furthermore, I would like to thank my colleagues Jens Grabowski, Philip Makedonski, Thomas Rings, and Patrick Harms for their insightful input and good discussions about this work.

9. REFERENCES

[1] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Database Theory - ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

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

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

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Database Theory - ICDT 1999*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

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

[6] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella. Multi-objective cross-project defect prediction. In *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2013.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

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

[10] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE)*. ACM, 2010.

[11] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 343–351, 2011.

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

[13] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[14] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the ACM/IEEE 30th International Conference on Software Engineering (ICSE)*, pages 181–190, 2008.

[15] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the ACM/IEEE 28th International Conference on Software engineering*,

ICSE '06, pages 452–461, New York, NY, USA, 2006. ACM.

[16] T. Ostrand, E. Weyuker, and R. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[17] F. Rahman, D. Posnett, and P. Devanbu. Recalling the "imprecision" of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE)*, FSE '12. ACM, 2012.

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

[19] S. Watanabe, H. Kaiya, and K. Kaijiri. Adapting a fault prediction model to allow inter language reuse. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE)*, pages 19–24. ACM, 2008.

[20] E. Weyuker, T. Ostrand, and R. Bell. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15(3):277–295, 2010.

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