An empirical study of public data quality problems in cross project defect prediction

Zhongbin Sun, Junqi Li, Heli Sun
School of Electronic and Information Engineering, Xi’an Jiaotong University
Xi’an, Shaanxi 710049, China
zhongbin725@mail.xjtu.edu.cn, vivi528@stu.xjtu.edu.cn, hlsun@xjtu.edu.cn

ABSTRACT

Background: Two public defect data, including Jureczko and NASA datasets, have been widely used in cross project defect prediction (CPDP). The quality of defect data have been reported as an important factor influencing the defect prediction performance and Shepperd et al. have researched the data quality problems in NASA datasets. However, up to now, there is no research focusing on the quality problems of Jureczko datasets which are most widely used in CPDP. Aims: In this paper, we intend to investigate the problems of identical and inconsistent cases in Jureczko datasets and validate whether removing these problematic cases will make a difference to defect prediction performance in CPDP. Method: The problems of identical and inconsistent cases are reported from two aspects, respectively in each individual dataset and in a pair of datasets from different releases of a software project. Then a cleaned version of Jureczko datasets is provided by removing duplicate and inconsistent cases. Finally three training data selection methods are employed to compare the defect prediction performance of cleaned datasets with that of original datasets. Results: The experimental results in terms of AUC and F-Measure show that most datasets obtain very different defect prediction performance. Conclusions: It is very necessary to study the data quality problems in CPDP and the cleaned Jureczko datasets may provide more reliable defect prediction performance in CPDP.

KEYWORDS
Software defect prediction, cross project, data quality

ACM Reference format:
Zhongbin Sun, Junqi Li, Heli Sun. 2018. An empirical study of public data quality problems in cross project defect prediction. In Proceedings of 12th International Symposium on Empirical Software Engineering and Measurement, Oulu, Finland, October 2018 (ESEM2018), 10 pages.
DOI: 10.1145/nnnnnn.nnmmnn

1 INTRODUCTION

Software systems, ubiquitous in the real world, have been widely used in different applications. With the continuous increase of software size and complexity, software quality assurance has become increasingly important. One way to improve software quality is software defect prediction, which has been an important research topic in the software engineering field for more than 40 years. Software defect prediction aims at finding the defect-prone modules in software and thus helps organizations to allocate the limited resources reasonably, which is an efficient means to relieve the effort in software code inspection or testing. Currently within project defect prediction (WPDP) [2, 22, 27, 36, 37] and cross project defect prediction (CPDP) [6, 12, 43, 48, 50] are two popular but different directions of defect prediction research.

In the field of software defect prediction research, most approaches employ machine learning classifiers to build a prediction model from training data mined from the software repositories, and the model is used to identify software defects in the test data. In WPDP, the training data and the test data come from the same project and thus for the purpose of ensuring the efficiency of WPDP, there are two essential assumptions, namely (1) the training data and the test data should be independently and identically distributed and (2) there are enough historical defect data to build a good prediction model. However, in practice, there are difficulties in collecting and organizing the defect data for many companies and the defect data are usually absent. In addition, the new projects often do not have enough defect data to build a prediction model. Therefore the traditional WPDP is usually inefficient with the scarcity of training data and many software engineering researchers have focused their research on CPDP in recent years.

Instead of using historical defect data from the same project as the training data, the CPDP methods aim to build the prediction model on one project (known as source) with sufficient historical data and use the model to make prediction on the other project (known as target). Specially, Herbold et al. [13] have categorized the CPDP research into mixed CPDP and strict CPDP according to the difference of source data. In mixed CPDP [9, 39, 40], the source data are from old revisions of the same project together with data from other projects while the strict CPDP [31, 38, 44, 49] research only employ data from other projects as the source data. In [38], Turhan et al. report that the CPDP model would have a low prediction performance with all the available source project data and they find that applying the nearest neighbor filtering to the source project data could considerably improve the prediction performance of the resulting CPDP model. Therefore, in the last few years, many studies have focused on the relevance filters [3, 4, 10, 20, 23, 26, 31, 44], which aims to employ different approaches to find the appropriate training data for building CPDP models.

Hosseini et al. [14] report that Jureczko [18, 19] and NASA [27] are the most widely used datasets in current CPDP studies, which occupies the proportion of 65% and 54% respectively. In fact, Shepperd et al. [35] have reported the data quality problems of NASA and provided two cleaned versions of the NASA data, namely DS′...
and $DS''$. The difference between $DS'$ and $DS''$ is that whether the duplicate and inconsistent cases are removed and Rodriguez et al. [34] find that there are obvious performance differences between the two datasets for WPDP. However, to our best knowledge, no CPDP research mentioned above have realized the data quality problems in the Jureczko data sets. The data quality in CPDP is a crucial problem and it may greatly influence the prediction performance, especially for the relevancy filter-based training data selection methods.

In present study we have firstly investigated the data quality problems of Jureczko datasets, including the identical and inconsistent cases in individual datasets and in a pair of datasets from different releases of a software project. Then a data cleaning method is proposed to remove the duplicate and inconsistent cases in the Jureczko datasets and a cleaned version of Jureczko datasets is provided. Finally three popular training data selection methods [Global Filter [26], Burak Filter [38] and Peters Filter [31]] are selected to compare the prediction performance of the cleaned datasets with that of original datasets. The experimental results in terms of AUC and F-Measure show that most problematic datasets obtain very different prediction performance, which indicates the effectiveness and necessity of our data cleaning approach for the Jureczko datasets.

The rest of this paper is organized as follows. Section 2 introduces the related work in cross project defect prediction. Section 3 provides a detailed description of Jureczko datasets and investigates the data quality problems in the Jureczko datasets. We provide a data cleaning method as well as the cleaned version of Jureczko datasets in Section 4. Section 5 describes the comparative experiments between the cleaned datasets and original datasets, and then reports the corresponding results analysis. Finally Section 6 concludes this paper.

## 2 RELATED WORK

In recent years, CPDP has drawn wide attention from software engineering researchers. A wide range of topics including verification of CPDP in different programming languages, different granularities and different companies, have been studied in existing CPDP research [6, 9, 16, 26, 30, 31, 38, 41, 42, 45, 46, 50]. These studies have greatly improved our understanding of the practical value of cross project defect prediction. It seems that CPDP has become an active research area in the field of defect prediction and has achieved promising defect prediction results [9, 15, 29]. However, Herbold et al. [12] and Hosseini et al. [14] draw the conclusion that CPDP is still a challenge and requires more research before trustworthy application in practice.

Many CPDP work mainly focus on verifying the feasibility of the CPDP methods [11, 24, 28, 33, 47]. Briand et al. [6] firstly propose cross project defect prediction, using the open source software Xpose defect data to make predictions for Jwiter. Carmago et al. [7] found that making the underlying distributions between the source code and target data similar may increase the quality of defect prediction. Nam et al. [28] extend a state-of-the-art transfer learning method TCA and propose a method TCA+. They found that TCA+ could significantly improve cross project prediction performance. In the case that different projects may provide different metrics, Nam et al. [29] present the heterogeneous defect prediction approach based on transferring knowledge. For solving the class imbalance problem in CPDP, Jing et al. [16] has proposed an improved SDA based defect prediction framework and it could significantly outperforms related methods. Zimmermann et al. [50] study cross project defect prediction models on a large scale and they investigate three factors that may influence the success of CPDP for the first time, including data, domain and process. They found that the success rate of the CPDP is very low (only 3.4% of 622 models are successful). In addition, Zimmermann et al. also found that the CPDP between the projects is not symmetrical.

The results obtained in [50] has shown the importance of training data selection. Turhan et al. [39] found that blindly selecting training data for CPDP can easily lead to high false negative rates for the prediction results. In addition, Hosseini et al. [15] have shown that the selection of training data can lead to better performance in cross project defect prediction. Therefore many researchers have focused their research on how to select appropriate training data for the target project and there are two lines of training data selection, including source projects selection and instances selection.

In the research of source projects selection, He et al. [9] found that if a best possible training set of three software projects from a set of available projects is selected through a strategy with a posteriori knowledge, a defect prediction success rate of over 50% may be achieved. Then they propose a strategy for the selection of source projects with decision trees over the distributional characteristics of training data and test data. However, their method does not scale with the number of datasets and the runtime of their method is exponential. Therefore based on the assumption that similar distributions lead to better results for cross project defect prediction, Herbold et al. [10] propose two distance-based strategies for the selection of source projects by using the distributional characteristics of available data. In addition, Khoshgoftaar et al. [21] propose to use a combination of multiple classifiers and data from multiple products for CPDP. They demonstrated that the combination of predictions of multiple learners trained on multiple datasets could improve the prediction performance of a learner induced on a single dataset. Aarti et al. [1] investigated the prediction accuracy of cross project defect prediction and indicate that cross project prediction can provide better prediction accuracy by combining different source projects.

Turhan et al. [38] have employed a nearest neighbor filtering method to select appropriate instances for building the CPDP model as they found that the predictive performance would be lower if all available source project data were used. The nearest neighbor filtering method use test instances to guide the selection of training data. After applying the nearest neighbor filtering on source project data, Turhan et al. found that the resulting predictive performance of the CPDP model is significantly improved. Bettenburg et al. [3] propose a local cluster guided selection method with a popular clustering algorithm. Then based on the filter proposed in [3], Menzies et al. [26] proposes a neighbor cluster guided selection method. Kawata et al. [20] introduce a density-based spatial clustering to guide training data filtering. Moreover, Peters et al. [31] propose a training instance guided selection method with k-means clustering algorithm. In this method, the training data and the test
Table 1: A statistic summary of the original Jureczko datasets

| ID  | Dataset | #Case | #Defective | %Defective |
|-----|---------|-------|------------|------------|
| 1   | ant1.7  | 745   | 166        | 22%        |
| 2   | arc     | 234   | 27         | 12%        |
| 3   | berek   | 43    | 16         | 37%        |
| 4   | camel1.0| 339   | 13         | 4%         |
| 5   | camel1.2| 608   | 216        | 36%        |
| 6   | camel1.4| 872   | 145        | 17%        |
| 7   | camel1.6| 965   | 188        | 19%        |
| 8   | ckmn    | 10    | 5          | 50%        |
| 9   | elearning| 64   | 5          | 8%         |
| 10  | forrest0.6| 6   | 1          | 17%        |
| 11  | forrest0.7| 29  | 5          | 17%        |
| 12  | forrest0.8| 32  | 2          | 6%         |
| 13  | intercafe| 27   | 4          | 15%        |
| 14  | ivy1.1  | 111   | 63         | 57%        |
| 15  | ivy1.4  | 241   | 16         | 7%         |
| 16  | ivy2.0  | 352   | 40         | 11%        |
| 17  | jedit3.2| 272   | 90         | 33%        |
| 18  | jedit4.0| 306   | 75         | 25%        |
| 19  | jedit4.1| 312   | 79         | 25%        |
| 20  | jedit4.2| 367   | 48         | 13%        |
| 21  | jedit4.3| 492   | 11         | 2%         |
| 22  | kalkulator| 27  | 6          | 22%        |
| 23  | log4j1.0| 135   | 34         | 25%        |
| 24  | log4j1.1| 109   | 37         | 34%        |
| 25  | log4j1.2| 205   | 189        | 92%        |
| 26  | lucene2.0| 195  | 91         | 47%        |
| 27  | lucene2.2| 247  | 144        | 58%        |
| 28  | lucene2.4| 340  | 203        | 60%        |
| 29  | nieruchomosci| 27  | 10        | 37%        |
| 30  | pdftranslator| 33  | 15        | 45%        |
| 31  | poi1.5  | 237   | 141        | 59%        |
| 32  | poi2.0  | 314   | 37         | 12%        |
| 33  | poi2.5  | 385   | 248        | 64%        |

Data are combined to a dataset and then k-means clustering algorithm is used to obtain different clusters. The clusters that contain at least one test instance are kept and the closest test instance in the same cluster for each training instance is found to label corresponding training instance. Finally, for the test instances, the Euclidean distance is used to select the nearest training case in the training cases that are labelled with the corresponding test case. All the selected training cases are combined into the training data for building a prediction model.

The Jureczko datasets [18, 19] have been widely used in the research mentioned above and Hosseini et al. [14] have reported that Jureczko datasets are the most widely used datasets in current CPDP studies. However, until now the PROMISE repository has provided the metrics and defect of 32 projects except the pbeans project. Therefore hereinafter we will investigate the obtained 32 projects.

In present study, the Jureczko datasets consist of 65 releases of 32 different software projects. In an individual dataset, each case represents a java class of the corresponding release and it contains two parts: features including 20 static code metrics and a labeled feature indicating the number of defects in that class. In this study, we preprocess the Jureczko datasets by considering a class as defective if the value of the labeled feature is equal or greater than 1. Therefore the obtained Jureczko datasets with 20 features and a binary label (defective or defect-free) are considered as the original datasets for future investigation. Table 1 provides a brief summary of the original Jureczko datasets, including the number of datasets.

3 INVESTIGATION

3.1 Jureczko Datasets

The most widely used Jureczko datasets are parts of the PROMISE repository and have been provided by Jureczko et al. [18, 19]. It consists of 33 different open source software development projects. However, until now the PROMISE repository has provided the metrics and defect of 32 projects except the pbeans project. Therefore hereinafter we will investigate the obtained 32 projects.

In present study, the Jureczko datasets consist of 65 releases of 32 different software projects. In an individual dataset, each case represents a java class of the corresponding release and it contains two parts: features including 20 static code metrics and a labeled feature indicating the number of defects in that class. In this study, we preprocess the Jureczko datasets by considering a class as defective if the value of the labeled feature is equal or greater than 1. Therefore the obtained Jureczko datasets with 20 features and a binary label (defective or defect-free) are considered as the original datasets for future investigation. Table 1 provides a brief summary of the original Jureczko datasets, including the number of datasets.

1The data is publicly available online: http://openscience.us/repo/defect/ck/
cases (#Case), the number of defective cases (#Defective) and the percentage of the defective cases (%Defective).

### 3.2 Data Quality Problems

Rodriguez et al. [34] have found that removing the duplicate and inconsistent cases in NASA datasets may have a significant impact on the prediction performance of WPDP. Thus in this section, we will investigate the problems of identical and inconsistent cases in Jureczko datasets. Apart from the data quality problems in each individual dataset, the data quality problems between different releases of a particular software project are also studied as the mixed CPDP uses the old releases of the target project for building the prediction model. In the following, the description of identical cases and inconsistent cases are firstly introduced.

**Identical Cases:** Two or more cases contain identical values for all features as well as the class label.

**Inconsistent Cases:** Two or more cases contain identical values for all features but the class labels differ.

Table 2 provide the detailed data quality analysis for each individual Jureczko datasets, including the number of identical cases (#Inc) and the number of inconsistent cases (#Ide).

#### Table 2: Detailed data quality analysis for each dataset

| Dataset       | #Inc | #Ide | Dataset       | #Inc | #Ide |
|---------------|------|------|---------------|------|------|
| ant1.7        | 0    | 36   | poi1.5        | 19   | 31   |
| arc           | 2    | 28   | poi2.0        | 1    | 48   |
| berk          | 0    | 0    | poi2.5        | 2    | 50   |
| camel1.0      | 0    | 22   | prop1         | 10399| 11860|
| camel1.2      | 38   | 58   | prop2         | 3483 | 13527|
| camel1.4      | 0    | 101  | prop3         | 2645 | 7734 |
| camel1.6      | 4    | 117  | prop4         | 2550 | 6577 |
| ckjm          | 0    | 0    | prop5         | 1687 | 5669 |
| elearning     | 0    | 12   | prop6         | 47   | 372  |
| forrest0.6    | 0    | 0    | redaktor      | 1    | 10   |
| forrest0.8    | 0    | 2    | serapion      | 0    | 2    |
| intercafe     | 0    | 0    | skarbonka     | 0    | 0    |
| ivy1.1        | 1    | 4    | sklebagd      | 0    | 0    |
| ivy1.4        | 0    | 10   | synapse1.0    | 0    | 6    |
| ivy2.0        | 0    | 14   | synapse1.1    | 1    | 12   |
| jedit3.2      | 0    | 8    | synapse1.2    | 0    | 18   |
| jedit4.0      | 1    | 6    | systemdta     | 0    | 4    |
| jedit4.1      | 0    | 8    | termoproject  | 0    | 2    |
| jedit4.2      | 0    | 8    | tomcat        | 0    | 98   |
| jedit4.3      | 32   | 24   | velocity1.4   | 0    | 28   |
| kalkulator    | 1    | 0    | velocity1.5   | 0    | 27   |
| log4j1.0      | 0    | 0    | velocity1.6   | 0    | 31   |
| log4j1.1      | 0    | 0    | workflow      | 0    | 2    |
| log4j1.2      | 0    | 6    | wspomaganiepi | 0    | 0    |
| lucene2.0     | 1    | 4    | xalan2.4      | 0    | 49   |
| lucene2.2     | 2    | 6    | xalan2.5      | 3    | 86   |
| lucene2.4     | 0    | 6    | xalan2.6      | 39   | 189  |
| nieruchomosci | 0    | 2    | xalan2.7      | 0    | 208  |
| pdftranslator | 0    | 0    | xeres1.2      | 34   | 114  |
| poi1.5        | 19   | 31   | xeres1.3      | 1    | 118  |
| poi2.0        | 1    | 48   | xeres1.4      | 5    | 132  |
| poi2.5        | 2    | 50   | xeresinit     | 3    | 19   |
| poi1.5        | 19   | 31   | zuzel         | 1    | 0    |

From Table 2, it could be observed that 52 datasets suffer from the problem of identical cases and 28 datasets suffer from the problem of inconsistent cases. The identical cases in the corresponding 52 datasets ranges from 1.76% (lucene2.4) to 75.44% (prob1). Likewise, the inconsistent cases in the 28 datasets take a proportion of range from 0.22% (xerces1.3) to 56.3% (prob1). Particularly, some datasets are severely affected by both the problems of inconsistent cases and identical cases, such as prop1, prop3 and prob4.

Table 3 provide the detailed data quality analysis for two different releases of a same project. Note that in Table 1 there is only one release in some projects and thus these projects with one release are absent in Table 3.

From Table 3, it could be observed that all the pairs of different releases for a particular software project suffer from the quality problem of identical cases and the number of identical cases range from 1 to 19283. In addition, most of the pairs of different releases also suffer from the problem of inconsistent cases and the number of inconsistent cases range from 1 to 5377.

In a word, there are many identical and inconsistent cases not only in each individual dataset but also in different releases of a particular software project. It is reasonable to doubt that whether so many identical and inconsistent cases may have unknowable effect on the prediction performance of cross project defect prediction. Therefore for the purpose of validating whether such effect is real, the original Jureczko datasets are cleaned in Section 4 and we conduct comparative experiments using the original and cleaned Jureczko datasets in Section 5.

### 4 DATA CLEANING

In this section, a data cleaning method is firstly proposed to deal with the problematic cases in the Jureczko datasets as there are many identical and inconsistent cases found in the Jureczko datasets. Then the cleaned datasets are provided as well as a brief statistic summary of the cleaned datasets.

Algorithm 1 provides the data cleaning method for Jureczko datasets in details. Particularly, the method consists of two main steps: remove duplicate cases and remove inconsistent cases. In Algorithm 1, lines 6-9 deal with the identical cases and remove the subsequent cases duplicate with the current case. The identical cases could be recognized when the features and class label of two cases are same. In addition, lines 10-13 are used to remove the two inconsistent cases which have the identical feature values but different class label values. Note that the order of the two steps can not be swapped, otherwise some inconsistent cases may not be removed. Shepperd et al. [35] has provided a simple example for explaining why these two steps could not be swapped.

By applying the data cleaning method Algorithm 1 to Jureczko datasets, a cleaned version of the datasets could be obtained and it is available from http://gr.xjtu.edu.cn/web/zhongbin725/datasets. Then for the purpose of showing the difference between the original data and cleaned data, Table 4 provides a brief statistic summary of the cleaned Jureczko datasets. Note that different from Table 1, Table 4 shows not only the number of cases (#Case) and defective cases (#Defective) but also the number of deleted cases (#delCase) and deleted defective cases (#delDefective).
Table 3: Detailed data quality analysis for different releases

| Release1 | Release2 | #Identical | #Inconsistent |
|----------|----------|------------|---------------|
| camel1.0 | camel1.2 | 112        | 19            |
| camel1.0 | camel1.4 | 111        | 2             |
| camel1.0 | camel1.6 | 92         | 3             |
| camel1.2 | camel1.4 | 408        | 52            |
| camel1.2 | camel1.6 | 406        | 39            |
| camel1.4 | camel1.6 | 863        | 46            |
| forrest0.6| forrest0.7| 1       | 0            |
| forrest0.6| forrest0.8| 1       | 0            |
|        | forrest0.8| 18        | 0            |
| ivy1.1  | ivy1.4   | 9         | 1             |
| ivy1.1  | ivy2.0   | 3         | 0             |
| ivy1.4  | ivy2.0   | 31        | 0             |
| jedit3.2| jedit4.0  | 71        | 4             |
| jedit3.2| jedit4.1  | 58        | 0             |
| jedit3.2| jedit4.2  | 31        | 0             |
| jedit3.2| jedit4.3  | 8         | 0             |
| jedit4.0| jedit4.1  | 67        | 4             |
| jedit4.0| jedit4.2  | 36        | 0             |
| jedit4.0| jedit4.3  | 8         | 0             |
| jedit4.1| jedit4.2  | 68        | 1             |
| jedit4.1| jedit4.3  | 20        | 0             |
| log4j1.0| log4j1.1  | 46        | 5             |
| log4j1.0| log4j1.2  | 7         | 42            |
| lucene2.0| lucene2.2 | 40       | 24            |
| lucene2.0| lucene2.4 | 12       | 8             |
| lucene2.2| lucene2.4 | 30       | 26            |
| poi1.5   | poi2.0   | 93        | 36            |
| poi1.5   | poi2.5   | 92        | 37            |
| poi1.5   | poi3.0   | 41        | 23            |
| poi2.0   | poi2.5   | 117       | 266           |
| poi2.0   | poi3.0   | 68        | 26            |
| poi2.5   | poi3.0   | 68        | 47            |

From Table 4, we observe that there are 35 datasets with the number of deleted defective cases greater than or equal to 1. Particularly, prop1 is the dataset with biggest number of deleted defective cases 1202. Furthermore, 54 datasets occur with the number of deleted cases greater than or equal to 1. The biggest number of deleted cases is 10899 with the dataset prop2. All in all, many datasets have shown apparent difference in the number of cases from their original datasets, which indicates that there may be performance differences when employing these two datasets for CPDP. Therefore comparative experiments will be conducted to validate whether there are prediction performance differences when using the original and cleaned Jureczko datasets for cross project defect prediction.

5 COMPARATIVE EXPERIMENTS AND RESULTS

For the purpose of validating the essentiality of removing the inconsistent and duplicate cases in Jureczko datasets, the comparative experiments are conducted between the original Jureczko datasets and the cleaned Jureczko datasets with three training data selection approaches. Moreover, in the comparative experiments, three popular classification algorithms as well as two widely used performance evaluation metrics are employed. In the following, we will firstly introduce the three training data selection approaches. Then the experimental setup including classification algorithms and performance evaluation metrics are briefly presented. Finally the experimental results and corresponding analysis are provided detailed.

5.1 Training Data Selection Approaches

In present study, three different training data selection approaches including Global Filter [26], Burak Filter [38] and Peters Filter [31], are selected and will be detailedly introduced later.

Global Filter (GlobalF): This filter method is described by Menzies et al. in [26] and Yi et al. [44] find GlobalF performs better than eight other relevancy filters in the context of CPDP. GlobalF uses all the external project data as the training data to build the CPDP model, which means that GlobalF filter out none of any source project data.

Burak Filter (BurakF): This filter is a popular kind of testing case guided selection method, proposed by Turhan et al. in [38].
Bayes [17], C4.5 [32] and Random Forest [5] have been selected. In this paper, three popular classification algorithms including Naive
Bayes and PetersF and it is a representative kind of training case guided
method. Firstly, PetersF combines all the source data and

for each DS_k ∈ Data do
  for i = 1 to M − 1 do //Step 1: remove duplicate cases
    for j = i + 1 to M do
      \[ DS_k = DS_k \setminus \{DS_k, Value[j][1...N] \} \]
  for i = 1 to M − 1 do //Step 2: remove inconsistent cases
  for j = i + 1 to M do
    if \[ DS_k, Value[i][1...N − 1] ≡ \]
      \[ DS_k, Value[j][1...N − 1] \] and
      \[ DS_k, Value[i][N] \neq DS_k, Value[j][N] \] then
      \[ DS_k = DS_k \setminus \{DS_k, Value[j][1...N] \}; \]
      \[ DS_k = DS_k \setminus \{DS_k, Value[i][1...N] \}; \]
return NewData = \{DS_1, DS_2, · · · , DS_k \};

BurakF firstly uses Euclidean distance to find 10 nearest neighbors in the source data for each testing case in the target data. Then all

Peters Filter (PetersF): This filter is proposed in [31] by Peters et al. and it is a representative kind of training case guided selection method. Firstly, PetersF combines all the source data and
target data to a combined dataset. Then the k-means clustering algorithm is used to obtain different clusters. After that, the clusters
containing at least one test case are retained and each training case in the retained clusters is labelled with the nearest test case in the
same cluster. Finally, for each test case, the Euclidean distance is used to select the nearest training case in the training cases that are labelled with the corresponding test case. All the selected training cases are combined into the training data for building a prediction model.

5.2 Experimental Setup

In this paper, three popular classification algorithms including Naive Bayes [17], C4.5 [32] and Random Forest [5] have been selected. These three algorithms are reported as the most frequently used classifiers in software defect prediction [25] and we have use their Weka [8] implementations with the default values.

We have selected F-Measure and AUC as our performance evaluation metrics in the comparative experiment. Both metrics are two of the most commonly used performance evaluation metrics in CPDP [14], F-Measure is the harmonic mean between recall and precision. AUC estimates the area under the ROC curve, which is defined using the Receiver Operating Characteristic (ROC).

5.3 Experimental Results

Table 5 and Table 6 show the rate of performance change of comparative experiments in terms of F-Measure and AUC, respectively. Particularly, the detailed performance change results for the combination of three classification algorithms and three training data selection methods are provided in Table 5 and Table 6.

From Table 5, it could be observed that average rate of F-Measure change ranges from -2% (Random Forest with PetersF) to 19.86% (Random Forest with GlobalF). Particularly, only Random Forest with PetersF obtains the reduced prediction performance with the cleaned Jureczko datasets. In addition, we could observe from Table 5 that there are performance differences for most employed datasets and many may be very great. This indicates that when using F-Measure as the evaluation metric, the performance differences between the original and cleaned Jureczko datasets are apparent.

From Table 6, it could be observed that the average ratio of AUC change ranges on a small scale compared with the average ratio of F-Measure change in Table 5. In fact, the average ratio of AUC change ranges from -3.02% (C4.5 with PetersF) to 3.09% (Random Forest with BurakF). However, compared with Table 5, we observe that more datasets obtain the performance difference as there are less 0 in Table 6 than in Table 5.

In conclusion, from Table 5 and Table 6, it could be found that most datasets have obtained the different defect prediction performance when using the original and cleaned Jureczko datasets separately. This means that the quality problems of identical and inconsistent cases in these datasets indeed influence the performance of cross project defect prediction, which indicates that for the purpose of obtaining more actual and reliable prediction performance for CPDP, it is necessary to deal with the problematic cases in the original Jureczko datasets.

6 CONCLUSION

Cross project defect prediction has been widely studied in recent decade. In the research of CPDP, Jureczko and NASA datasets are two most widely used public datasets. Researchers have found the importance of selecting appropriate training data for building a CPDP model. However, though the NASA datasets have been reported to have data quality problems, there is still no research focusing on the data quality problems of Jureczko. Therefore in present study, we have firstly investigated the data quality problems in the Jureczko datasets and found that there are many datasets with identical cases and inconsistent cases. Then a data cleaning method is proposed to deal with the original Jureczko datasets and a cleaned version of Jureczko datasets is provided. Finally, the comparative experiments are conducted with the original and cleaned Jureczko datasets. In the comparative experiments, three training data selection methods and two classification algorithms are employed with using F-Measure and AUC as the performance evaluation metrics. The experimental results show that many problematic datasets obtain very different defect prediction performance, which indicates the effectiveness and necessity of our study of public data quality problems in cross project defect prediction.
An empirical study of public data quality problems in cross project defect prediction

Table 4: A statistic summary of the cleaned Jureczko datasets

| ID | Dataset       | #Case | #delCase | #Defective | #delDefective |
|----|---------------|-------|----------|------------|---------------|
| 1  | anti1.7       | 724   | 21       | 166        | 0             |
| 2  | arc           | 213   | 21       | 25         | 2             |
| 3  | berek         | 43    | 0        | 16         | 0             |
| 4  | came1.0       | 327   | 12       | 13         | 0             |
| 5  | came1.2       | 558   | 50       | 205        | 11            |
| 6  | came1.4       | 802   | 70       | 144        | 1             |
| 7  | came1.6       | 878   | 87       | 181        | 7             |
| 8  | ckjm          | 10    | 0        | 5          | 0             |
| 9  | elearning     | 57    | 7        | 5          | 0             |
| 10 | Forrest0.6    | 6     | 0        | 1          | 0             |
| 11 | Forrest0.7    | 28    | 1        | 5          | 0             |
| 12 | Forrest0.8    | 31    | 1        | 2          | 0             |
| 13 | Intercase     | 27    | 0        | 4          | 0             |
| 14 | Iivy1.1       | 107   | 4        | 62         | 1             |
| 15 | Iivy1.4       | 236   | 5        | 16         | 0             |
| 16 | Iivy2.0       | 345   | 7        | 40         | 0             |
| 17 | Jedit3.2      | 268   | 4        | 90         | 0             |
| 18 | Jedit4.0      | 301   | 7        | 5          | 0             |
| 19 | Jedit4.1      | 308   | 4        | 79         | 0             |
| 20 | Jedit4.2      | 363   | 4        | 48         | 1             |
| 21 | Jedit4.3      | 474   | 18       | 7          | 4             |
| 22 | Kalkulator    | 25    | 2        | 5          | 1             |
| 23 | Log4j1.0      | 135   | 0        | 34         | 0             |
| 24 | Log4j1.1      | 109   | 0        | 37         | 0             |
| 25 | Log4j1.2      | 202   | 3        | 187        | 2             |
| 26 | Lucene2.0     | 191   | 4        | 90         | 1             |
| 27 | Lucene2.2     | 238   | 9        | 139        | 5             |
| 28 | Lucene2.4     | 336   | 4        | 199        | 4             |
| 29 | Nieruchomosci | 26    | 1        | 0          | 0             |
| 30 | Pdttranslator | 33    | 0        | 15         | 0             |
| 31 | Poi1.5        | 203   | 34       | 122        | 19            |
| 32 | Poi2.0        | 282   | 32       | 35         | 2             |
| 33 | Poi2.5        | 350   | 35       | 221        | 27            |

REFERENCES

[1] Aarti, Geeta Sikka, and Renu Dhir. 2017. An investigation on the effect of cross project data for prediction accuracy. International Journal of System Assurance Engineering and Management 8, 2 (2017), 1–26.

[2] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannaphitha, Akito Monden, and Solomon Mensah. 2017. MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Transactions on Software Engineering PP, 99 (2017), 1–1.

[3] Nicolas Bettenburg, Metiyappan Nagappan, and Ahmed E. Hassan. 2012. Think locally, act globally: Improving defect and effort prediction models. In Proceedings of the 9th IEEE International Conference on Mining Software Repositories. 60–69.

[4] Nicolas Bettenburg, Metiyappan Nagappan, and Ahmed E. Hassan. 2015. Towards improving statistical modeling of software engineering data: think locally, act globally! Empirical Software Engineering 20, 2 (2015), 294–335.

[5] Leo Breiman. 2001. Random forests. Machine Learning 45, 1 (2001), 5–32.

[6] Lionel C Briand, Walcelio L. Melo, and Jurgen Wust. 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Transactions on Software Engineering 28, 7 (2002), 706–720.

[7] Ana Erika Camargo Cruz and Koichiro Ochimizu. 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. 469–463.

[8] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: An update. ACM SIGKDD Explorations Newsletter 11, 1 (2009), 10–18.

[9] Zhihui He, Fenghui Shu, Ye Yang, Minghwa Li, and Qing Wang. 2012. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering 19, 2 (2012), 167–199.

[10] Steffen Herbold. 2013. Training data selection for cross-project defect prediction. In Proceedings of the 9th International Conference on Predictive Models in Software Engineering, 6–15.

[11] Steffen Herbold. 2017. A systematic mapping study on cross-project defect prediction. arXiv preprint arXiv:1705.06429 (2017).

[12] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering PP, 99 (2017), 1–1.

[13] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. Global vs. local models for cross-project defect prediction. Empirical Software Engineering 22, 4 (2017), 1866–1902.

[14] Seyyedrehab Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2017. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering PP, 99 (2017), 1–1.

[15] Seyedrehab Hosseini, Burak Turhan, and Mika Mntyl. 2016. Search based training data selection for cross project defect prediction. In Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering. 3–12.

[16] Xiaoyuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. 2017. An improved SAN based detection framework for both within-project and cross-project class-imbalance problems. IEEE Transactions on Software Engineering 43, 4 (2017), 321–339.

[17] George H. John and Pat Langley. 1995. Estimating continuous distributions in Bayesian classifiers. In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. 338–345.

[18] Marijan Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software Engineering. 9–18.

[19] Marijan Jureczko and Diomidis Spinellis. 2010. Using object-oriented design metrics to predict software defects. Models and Methods of System Dependability (2010), 69–81.
| Data        | Naive Bayes | C4.5 | Random Forest |
|-------------|-------------|------|---------------|
| an1.7       | 0.39        | 5.39 | -1.59         |
| arc         | 4.44        | 3.28 | -10.59        |
| berek       | 0           | -14.53 | 0.35 |
| camel1.0    | 0           | -12.35 | 33.33 |
| camel1.2    | 4.64        | 3.96 | -3.06         |
| camel1.4    | 1.01        | 0.85 | -0.15         |
| camel1.6    | 3.14        | 5.34 | -1.72         |
| ckjm        | 0           | 0    | 0             |
| elearning   | 0           | 0    | 56            |
| forrest0.6  | 0           | 0    | 20            |
| forrest0.8  | 0           | 0    | 0             |
| intercalfe  | 0           | 11.11 | 24.44 |
| ivy1.1      | 1.43        | 2.67 | 1.30          |
| ivy2.0      | 0           | -1.01 | 2.30 |
| jedit3.2    | 0           | 0    | -11.87        |
| jedit4.0    | 0.81        | 0.85 | 1.34          |
| jedit4.1    | 0           | 0    | -16.67        |
| jedit4.2    | 0           | 0    | -1.03         |
| jedit4.3    | 4.26        | 3.96 | -10.65        |
| kalkulator  | 11.11       | 10   | -100          |
| nieruchomosci | 0      | 0    | 69.23         |
| pdtranslator | 0       | 0    | 4.76          |
| poi1.5      | 13.29       | 12.50 | 10.40 |
| poi2.0      | 2.86        | 2.86 | 5.97          |
| poi2.5      | 19.19       | 10.23 | 4.16 |
| poi3.0      | 8.52        | 8.55 | 42.95         |
| prop1       | 33.67       | 30.40 | 32.35 |
| prop2       | 16.13       | 13.46 | 14.73 |
| prop3       | 28.50       | 12.26 | 0.61  |
| prop4       | 31.79       | 29.09 | 23.96 |
| prop5       | 38.73       | 27.56 | 39.76 |
| prop6       | 52.37       | 9.48  | 10.69         |
| redaktor    | 5.41        | -13.16 | 3.03 |
| serapion    | 0           | -6.25 | 16.58        |
| skarbonka   | 0           | 0    | 7.69          |
| sklebagd    | 0           | 0    | 0             |
| synapse1.0  | 0           | 9.69  | -22.86        |
| synapse1.1  | 1.22        | 0.97  | 47.56         |
| synapse1.2  | 0           | -9.45 | 0.86  |
| systemdata  | 0           | 0    | -26.28        |
| szybkaufcha | 0           | 0    | -5.56         |
| termproject | 0           | -3.51 | 0.0          |
| tocmat      | 0.45        | 0    | -23.69        |
| velocity1.4 | 10.27       | 9.62  | 6.50          |
| velocity1.5 | 6.00        | 5.84  | 6.67          |
| worklflow   | 0           | 0    | 1          |
| wspomaganiepi | 0      | 0    | 355          |
| xalan2.4    | 0           | 1.39  | 3.39          |
| xalan2.5    | 8.11        | 5.86  | -9.39         |
| xalan2.6    | -17.07      | -0.78 | -3.61        |
| xalan2.7    | -9.26       | 2.03  | 9.38          |
| xerces1.2   | 12.04       | 11.50 | 10.61         |
| xerces1.3   | 0.85        | 8.83  | 1.56          |
| xerces1.4   | 14.02       | 13.44 | 6.95          |
| avg         | 4.06        | 2.92  | 5.56          |

Table 5: Rate of performance change (%) in terms of F-Measure
| Data       | Naive Bayes | C4.5 | Random Forest |
|------------|-------------|------|---------------|
| ant1.7     | -0.75       | -0.81| 1.91          |
| arc        | 2.20        | 3.64 | 2.87          |
| berek      | 0.26        | 0    | -9.65         |
| camel1.0   | -0.81       | -1.14| -18.70        |
| camel1.2   | -0.41       | -2.60| 1.86          |
| camel1.4   | -2.48       | -3.61| -8.63         |
| camel1.6   | -0.89       | -2.52| -2.59         |
| ckjm       | 0           | 0    | 0             |
| elearning  | -0.90       | -1.54| 6.79          |
| forest0.6  | -1.65       | -1.09| 21.33         |
| forest0.8  | -4.31       | -0.69| 0             |
| intercalfe | 0           | 0    | -17.11        |
| ivy1.1     | -0.10       | 0.10 | 8.30          |
| ivy1.4     | -0.38       | -0.39| 3.72          |
| ivy2.0     | 0.53        | -0.46| 3.61          |
| jedi3.2    | 0.42        | 0.52 | -2.98         |
| jedi4.0    | 0.93        | 0.97 | -0.05         |
| jedi4.1    | -0.22       | -0.28| 4.78          |
| jedi4.2    | 0.17        | 0.22 | 0.08          |
| jedi4.3    | 39.33       | 45.97| 47.60         |
| kalkulator | 15.88       | 7.37 | -74.93        |
| log4j1.0   | 0.26        | -0.18| 31.50         |
| log4j1.1   | 0           | -0.09| 14.59         |
| log4j1.2   | -2.97       | -2.10| 1.28          |
| lucene2.0  | 0.21        | 0.23 | 2.91          |
| lucene2.2  | 1.27        | 0.52 | 4.50          |
| nichtuchomosci | 0.60 | 0 | 2.12 |
| poi1.5     | 3.67        | 2.15 | 2.93          |
| poi2.0     | -0.08       | -0.02| 13.90         |
| poi2.5     | 2.21        | -0.28| -7.35         |
| poi3.0     | 0.37        | 1.08 | -5.56         |
| prop1      | 7.85        | 8.84 | 10.60         |
| prop2      | 3.92        | 3.78 | 4.40          |
| prop3      | 9.65        | 9.90 | 4.51          |
| prop4      | 10.45       | 11.33| 8.98          |
| prop5      | 7.09        | 5.04 | 6.66          |
| prop6      | 6.14        | 4.56 | 6.65          |
| redaktor   | 4.16        | 3.63 | 4.37          |
| skarbnoka  | -0.83       | 0.21 | 3.86          |
| sklebagd   | 0           | 0    | 12.22         |
| synapse1.0 | -0.65       | -0.37| -33.32        |
| synapse1.1 | -0.28       | -0.07| 14.54         |
| synapse1.2 | -2.11       | -1.84| 9.39          |
| systemdata | 1.84        | -0.01| -3.96         |
| szybkafucha | 0          | 0.28| 13.44         |
| termoproj | -1.09       | -0.25| -20.52        |
| tomcat     | -1.73       | 0    | 1.59          |
| velocity1.4| 5.26        | 3.21 | 85.79         |
| velocity1.5| 1.72        | 0.82 | 1.44          |
| velocity1.6| -1.53       | -2.85| 12.61         |
| workflow   | -3.24       | -2.64| 12.45         |
| wspomaganiepi | 0       | 0 | 9.45 |
| xalan2.4   | -1.03       | -1.59| -0.47         |
| xalan2.5   | 1.34        | 0.87 | 1.48          |
| xalan2.6   | -4.26       | 1.99 | -3.41         |
| xalan2.7   | -4.92       | -2.14| -0.99         |
| xerces1.2  | 3.48        | 5.50 | 4.69          |
| xerces1.3  | -6.19       | -6.16| -8.84         |
| xerces1.4  | 1.25        | -1.79| -3.37         |
| xercesinit | 5.70        | 9.05 | 3.53          |
| zuzel      | 5.71        | 8.02 | 0             |

| AVG        | 1.19        | 1.37 | 2.32          |

Table 6: Rate of performance change (%) in terms of AUC
[20] Kazuya Kawata, Sousuke Amanashi, and Tomoyuki Yokogawa. 2016. Improving relevancy filter methods for cross-project defect prediction. In Applied Computing and Information Technology: 1–12.

[21] Taghi M. Khoshgoftaar, Pierre Rebours, and Naeem Seliya. 2009. Software quality analysis by combining multiple projects and learners. Software Quality Journal 17, 1 (2009), 25–49.

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

[23] Yong Li, Zhiqui Huang, Yong Wang, and Bingwu Fang. 2017. Evaluating data filter on cross-project defect prediction: Comparison and improvements. IEEE Access 5, 99 (2017), 25646–25656.

[24] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. 2012. Transfer learning for cross-company software defect prediction. Information and Software Technology 54, 3 (2012), 248–256.

[25] Ruchika Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing 27 (2015), 504–518.

[26] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33, 1 (2007), 2–13.

[27] Jaechang Nam, Sinno Jialin Pan, and Sung hun Kim. 2013. Transfer defect learning. In Proceedings of the 2013 International Conference on Software Engineering, 382–391.

[28] Jaechang Nam, Fu Wei, Sung hun Kim, Tim Menzies, and Tan Lin. 2015. Heterogeneous Defect Prediction. IEEE Transactions on Software Engineering PP, 99 (2015), 1–23.

[29] Fayola Peters, Tim Menzies, Liang Gong, and Hongyu Zhang. 2013. Balancing privacy and utility in cross-company defect prediction. IEEE Transactions on Software Engineering 39, 8 (2013), 1054–1068.

[30] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better cross company defect prediction. In 26th IEEE Working Conference on Mining Software Repositories, 409–418.

[31] J Ross Quinlan. 2014. C4.5: Programs for machine learning. Elsevier.

[32] Foyuz Rahman, Daryl Posnett, and Premkumur Devanbu. 2012. Recalling the imprecision of cross-project defect prediction. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 61–71.

[33] Daniel Rodriguez, Israel Herrain, Rachel Harrison, and Javier Dolado. 2014. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In International Conference on Evaluation and Assessment in Software Engineering, 43–52.

[34] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data quality: Some comments on the NASA software defect datasets. IEEE Transactions on Software Engineering 39, 9 (2013), 1208–1215.

[35] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. 2011. A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering 37, 3 (2011), 356–370.

[36] Zhongbin Sun, Qinbao Song, and Xiaoyan Zhu. 2012. Using coding-based ensemble learning to improve software defect prediction. IEEE Transactions on Systems, Man and Cybernetics: Part C 42, 6 (2012), 1806–1817.

[37] Burak Turhan, Tim Menzies, Aye Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 14, 5 (2009), 540–578.

[38] Burak Turhan, Ayse Tosun Muzuri, and Ayse Bener. 2013. Empirical evaluation of the effects of mixed project data on learning defect predictors. Information and Software Technology 55, 6 (2013), 1101–1118.

[39] Burak Turhan, Ayse Tosun, and Ayse Bener. 2011. Empirical evaluation of mixed-project defect prediction models. In 17th EUROMICRO Conference on Software Engineering and Advanced Applications, 396–403.

[40] Shinya Watanabe, Haruhiko Kaiya, and Kenji Kajiri. 2008. Adapting a fault prediction model to allow inter language reuse. In International Workshop on Predictor MODELS in Software Engineering, 19–24.

[41] Fei Wu, Xiao-Yuan Jing, Xiwei Dong, Jicheng Cao, Mingwei Xu, Hongyu Zhang, Shi Ying, and Baowen Xu. 2017. Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In IEEE/ACM 39th International Conference on Software Engineering Companion, 195–197.

[42] Xin Xia, David Lo, Sin no Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. HYDRA: Massively compositional model for cross-project defect prediction. IEEE Transactions on Software Engineering 42, 10 (2016), 977–998.

[43] Bin Yi, Kai Zhou, Hongmin Lu, Yuming Zhou, and Baowen Xu. 2017. Training data selection for cross-project defect prediction: Which approach is better? In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 354–363.