Empir Software Eng (2017) 22:3186–3218
DOI 10.1007/s10664-017-9516-2

Data Transformation in Cross-project Defect Prediction

Feng Zhang[1] · **Iman Keivanloo**[2] · **Ying Zou**[2]

Published online: 14 April 2017

Abstract Software metrics rarely follow a normal distribution. Therefore, software metrics are usually transformed prior to building a defect prediction model. To the best of our knowledge, the impact that the transformation has on cross-project defect prediction models has not been thoroughly explored. A cross-project model is built from one project and applied on another project. In this study, we investigate if cross-project defect prediction is affected by applying different transformations (i.e., log and rank transformations, as well as the Box-Cox transformation). The Box-Cox transformation subsumes log and other power transformations (e.g., square root), but has not been studied in the defect prediction literature. We propose an approach, namely Multiple Transformations (MT), to utilize multiple transformations for cross-project defect prediction. We further propose an enhanced approach MT+ to use the parameter of the Box-Cox transformation to determine the most appropriate training project for each target project. Our experiments are conducted upon three publicly available data sets (i.e., AEEEM, ReLink, and PROMISE). Comparing to the random forest model built solely using the log transformation, our MT+ approach improves

Communicated by: Tim Menzies

Electronic supplementary material The online version of this article (doi:10.1007/s10664-017-9516-2) contains supplementary material, which is available to authorized users.

✉ Feng Zhang
feng@cs.queensu.ca

Iman Keivanloo
iman.keivanloo@ieee.org

Ying Zou
ying.zou@queensu.ca

[1] School of Computing, Queen's University, Kingston, Ontario, Canada

[2] Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada

the F-measure by 7, 59 and 43% for the three data sets, respectively. As a summary, our major contributions are three-fold: 1) conduct an empirical study on the impact that data transformation has on cross-project defect prediction models; 2) propose an approach to utilize the various information retained by applying different transformation methods; and 3) propose an unsupervised approach to select the most appropriate training project for each target project.

Keywords Defect prediction · Data transformation · Software metrics · Box-cox

1 Introduction

A defect is an error that can cause a software system to behave in an unexpected way or produce incorrect results. In the last decade, defect prediction has attracted great attention from both researchers and practitioners. Software metrics (e.g., lines of code and the number of method calls) constitute the major part of the input data used to build a defect prediction model. Earlier studies (e.g., Concas et al. 2007; Louridas et al. 2008; Zhang 2009) report that software metrics rarely follow a normal distribution, but a power-law distribution, which threats the fitness of prediction models to provide an accurate prediction (Cohen et al. 2003).

In the literature of defect prediction, researchers widely apply log and rank transformations to improve the normality of software metrics (e.g., Menzies et al. 2007; Jiang et al. 2008; Cruz and Ochimizu 2009; Song et al. 2011; Zhang et al. 2014). The log transformation is basically a mathematical operation that replaces the original metric values by their logarithm, thus suits log-normal data (i.e., normally distributed data after the log transformation). The rank transformation substitutes the original metric values with their ranks.

Despite of the success to improve the normality of software metrics, the aforementioned transformations fail to constantly improve the performance of defect prediction models in a within-project setting (Jiang et al. 2008). A within-project model is built and applied within the same project. However, to the best of our knowledge, the impact that such transformations have on the performance of defect prediction models has not been thoroughly investigated in a cross-project setting. A cross-project model is built using the training data from one project and applied on the target data from another project.

Cross-project prediction is needed when the target project (e.g., a small or new project) does not have sufficient historical data to build a prediction model (Nagappan et al. 2006). Cross-project prediction experiences a great challenge to deal with the heterogeneity between the training and target projects, since software metrics among different projects often exhibit varied distributions (Zhang et al. 2013). Transformations, if learnt from both the training and target projects, have the potential to mitigate the heterogeneity between the training and target projects. For instance, our previous work (Zhang et al. 2014) successfully implements the context-aware rank transformation towards generalizing defect prediction models. In addition, Ma et al. (2012) propose to transform the training project based on the statistical characteristics learnt from the target project. Nam et al. (2013) apply transfer component analysis (TCA) approach to transform both the training and target projects. Although both approaches on average significantly improve the performance of cross-project predictions, it is unclear how to choose an appropriate transformation for a particular pair of training and target projects. Jiang et al. (2008) even show that the benefit of transformations varies with modelling techniques on the same dataset.

Nonetheless, different transformations retain the information of the original data from various perspectives, especially in the cross-project setting. Therefore, in this study, we set out an exploratory study to investigate if using different transformations that retain distinct characteristics of software metrics is beneficial to cross-project defect prediction.

We perform experiments using three publicly available data sets, i.e., AEEEM (D'Ambros et al. 2010), ReLink (Wu et al. 2011), and PROMISE (Jureczko and Madeyski 2010). First, we examine if transformations have the same ability to improve the normality of software metrics. Besides log and rank transformations, we study the Box-Cox transformation (Box and Cox 1964) that represents a family of power transformations (e.g., the log transformation) but has not been investigated in existing studies on defect prediction. Second, we study if different transformations cause distinct predictions on the same file in the cross-project setting. We propose an approach, namely Multiple Transformations (MT), to integrate predictions by multiple models, with each model built using a single transformation. The weight of each model is determined by its accuracy in predicting defective instances on the training data. We further enhance our approach (MT+) by automatically selecting the most appropriate training project for each target project based on the parameter of the Box-Cox transformation. Accordingly, we study four research questions:

RQ1. *Are log, Box-Cox, and rank transformations equally effective in increasing the normality of software metrics?*

All three transformations can significantly improve the normality of software metrics (i.e., reduce both the skewness and kurtosis). The three transformations have similar ability to improve the normality of software metrics, with small or negligible difference indicated by Cliff's δ.

RQ2. *Do different transformations result in distinct predictions in cross-project defect prediction models?*

In general, models built with each of the three transformations do not exhibit significantly difference in terms of the six studied performance measures (i.e., precision, recall, false positive rate, balance, F-measure and AUC values). However, the results of McNemar's test indicate that the three prediction models judge differently about the defect proneness of each file. When a defective file is overlooked by one model, it may be captured by other models.

RQ3. *Can our approaches improve the performance of cross-project defect prediction models?*

The results show that our approach MT+ statistically significantly improves the performance of cross-project defect prediction, comparing to models built with the log transformation. On average, our MT+ approach increases F-measure of cross-project defect prediction models built using logistic regression by 24% (i.e., from 0.34 to 0.42), 11% (i.e., from 0.54 to 0.60), and 29% (i.e., from 0.31 to 0.42) in AEEEM, ReLink, and PROMISE datasets, respectively.

RQ4. *Do our approaches work well for other classifiers?*

We study the generalizability of our approaches using six other classifiers (e.g., Naive Bayes, and random forest), since different classifiers are reported to prefer different transformations (Jiang et al. 2008). We find that our approach MT+ generally outperforms models built with the log transformation.

Our major contributions are: 1) study if data transformation impacts cross-project defect prediction; 2) propose to utilize the various information retained through different transformation methods; and 3) propose to use the parameter of the Box-Cox transformation to select the most appropriate training project for each target project.

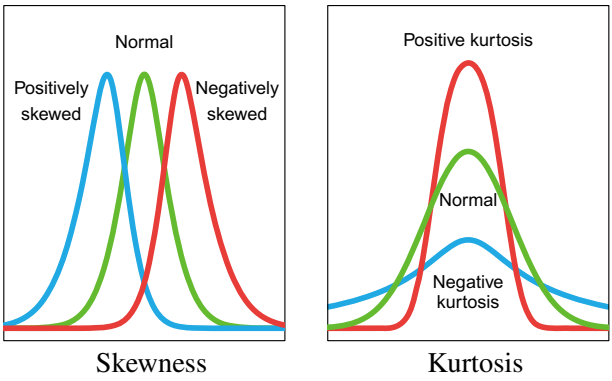

Fig. 1 The illustration of skewness and kurtosis in a distribution

Paper organization. Section 2 presents the three studied transformation methods. The experimental setup is presented in Section 3. Our motivation study is described in Section 4. Our approaches (i.e., MT and MT+) and evaluation are presented in Sections 5 and 6, respectively. The related work is summarized in Section 7. The threats to validity of our work are discussed in Section 8. We conclude the paper and provide insights for future work in Section 9.

2 Background on transformation methods

In this section, we describe two common measurements of data normality, and present the details of the three studied transformation methods.

2.1 Normality measurements

Skewness and kurtosis are two widely applied measurements of data normality. We compute these two measurements to measure the normality of software metrics, using the *R* functions *skewness* and *kurtosis* in the *R*[1] package *e1071*.[2]

a) **Skewness** measures the degree of symmetry in the probability distribution of the values of a software metric. The value of skewness can be positive (indicating a long tail to the right), negative (indicating a long tail to the left), or zero (indicating balanced tails on both sides), as illustrated in Fig. 1a. The ideal value of skewness ranges from −0.80 to 0.80 (Osborne 2010).

b) **Kurtosis** measures the "peakness" (e.g., the width of the peak) in the probability distribution of the values of a software metric. The value of kurtosis can be positive that indicates a more acute peak, or negative that indicates a lower and wider peak. Positive and negative kurtosis are illustrated in Fig. 1b. The ideal value of kurtosis is zero.

[1] https://www.r-project.org

[2] https://cran.r-project.org/web/packages/e1071

2.2 Log transformation

The log transformation is a mathematical operation that computes the logarithm (mostly the natural logarithm) of software metrics to replace the original values. The log transformation is widely used in building software defect prediction models (e.g., Menzies et al. 2007; Song et al. 2011).

The log transformation can only transform numerical values that are greater than zero, due to the limitation of the function "ln(x)". To deal with zero values, a constant is often added, such as "$ln(x + 1)$". An alternative solution is to replace all values under 0.000001 by 0.000001. We apply the following commonly used equation:

$$Log(x) = ln(x + 1) \tag{1}$$

where x is the value of a software metric.

2.3 Rank transformation

The rank transformation replaces the original values by their ranks. The rank transformation is recommended to deal with heavy-tailed distributions (i.e., have high kurtosis) Bishara and Hittner (2014) and Keren and Lewis (1993). In the literature of defect prediction, Jiang et al. (2008) observe that the rank transformation can improve the performance of some classifiers (e.g., Naive Bayes). Moreover, the rank transformation has been successfully applied to mitigate the heterogeneity of software metrics across projects in the cross-project setting (Zhang et al. 2014).

In this study, we convert the original values of each metric into ten ranks, using every $10th$ percentile of the corresponding metric, as defined in (2).

$$Rank(x) = \begin{cases} 1 & \text{if } x \in [0, Q_1] \\ k & \text{if } x \in (Q_{k-1}, Q_k], k \in \{2, \ldots, 9\} \\ 10 & \text{if } x \in (Q_9, +\infty) \end{cases} \tag{2}$$

*where Q_k is the k*10% percentile of the corresponding metric in* ***the union of the training and target projects****.*

2.4 Box-Cox transformation

The Box-Cox transformation represents a family of power transformations, as defined in (3). To the best of our knowledge, the Box-Cox transformation has not been explored in the literature of defect prediction.

$$BoxCox(x, \lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ ln(x) & \text{if } \lambda = 0 \end{cases} \tag{3}$$

where x is the value of a metric, and λ is the only configuration parameter of the Box-Cox transformation.

The parameter λ determines the concrete format of the Box-Cox transformation. For example, "$\lambda = 1.0$" means no transformation, "$\lambda = 0.5$" equals to the square root transformation, "$\lambda = 0.0$" represents the log transformation, and "$\lambda = -1.0$" indicates the inverse transformation. As such, the Box-Cox transformation is often used to transform variables that follow a power law distribution. The Box-Cox transformation is suggested to improve the variance homogeneity, increase the precision of estimation, and simplify models (Shang 2014).

The parameter λ can be estimated from a sample of data points. In the context of cross-project prediction, the parameter λ is estimated from **both the training and target projects**. The details to apply the Box-Cox transformation in our study are presented as follows.

1) **Shifting metric values to 1.0.** As suggested by Guo (2014), we shift the minimum value of a metric in a distribution at exactly 1.0 before applying the Box-Cox transformation. This treatment can increase the accuracy of the Box-Cox transformation (Guo 2014). We use the equation $\tilde{x} = x - min(x) + 1$, where x is the value of a software metric.
2) **Estimating the parameter λ.** The parameter λ is estimated for each metric independently, since different metrics rarely follow the same distribution. To ensure the same transformation applied on both the training and target projects, as aforementioned, we estimate the parameter λ using the values of the corresponding metric from both sets.

   We estimate the parameter λ in an iterative process. First, we select a set of candidate λ values that range from −1.0 to 1.0. Second, we iterate the λ values from −1.0 towards 1.0 with a step of 0.1. At each iteration, we compute the skewness of transformed values. We select the λ value that leads to the minimum skewness (i.e., the absolute skewness value is the closest to zero) of transformed values. The iterative process can be described using the following equation:

$$\widehat{\lambda} = \arg\min_{\lambda \in L} |skewness(BoxCox(\tilde{x}, \lambda))| \quad (4)$$

   where L is a set of candidate λ values from -1.0 to 1.0 with a step by 0.1, and X is a vector of shifted metric values.
3) **Normalizing transformed values.** Normalization creates equal scales of software metrics, and is useful for classification algorithms (Han et al. 2012; Nam et al. 2013). In this study, we choose the min-max method (Han et al. 2012), since it can normalize values exactly into the range of [0, 1]. Based on the benefit of shifting the minimum value to 1.0 (Guo 2014), we slightly modify this method using the following (5).

$$Normalize(\widehat{x}) = \frac{\widehat{x} - \min_{\widehat{x} \in U}(\widehat{x})}{\max_{\widehat{x} \in U}(\widehat{x}) - \min_{\widehat{x} \in U}(\widehat{x})} + 1 \quad (5)$$

   where $\widehat{x}$ is the transformed value by (3) using $\tilde{x}$ and $\widehat{\lambda}$, and U is a set of $\widehat{x}$ from the union of the training and target projects.

3 Experimental setup

In this section, we first describe our subject projects. Then, we present classifiers to build cross-project defect prediction models, and six performance measures used in this study.

3.1 Subject projects

In this study, we choose three publicly available datasets, such as AEEEM (D'Ambros et al. 2010), ReLink (Wu et al. 2011), and PROMISE (Jureczko and Madeyski 2010). The three datasets have been widely used for cross-project defect prediction (e.g., Nam et al. 2013). The diversity of the three datasets can help verify the generalizability of our approach. Table 1 presents the summary of the three datasets.

1) AEEEM dataset was made by D'Ambros et al. (2010), and contains 61 metrics. It has the two largest projects (i.e., Mylyn and PDE) among the three datasets. The ratio of

Table 1 Descriptive statistics of all 18 subject projects from AEEEM, ReLink, and PROMISE datasets

Dataset	Projects		# of Files	# of LOC	# of Buggy Files (%)
AEEEM	(A1)	Eclipse JDT Core	997	224K	206 (20.7%)
	(A2)	Equinox	324	40K	129 (39.8%)
	(A3)	Apache Lucene	691	73K	64 (9.3%)
	(A4)	Mylyn	1862	156K	245 (13.2%)
	(A5)	Eclipse PDE UI	1497	147K	209 (14.0%)
ReLink	(R1)	Apache HTTP Server	194	89K	98 (50.5%)
	(R2)	OpenIntents Safe	56	8K	22 (39.3%)
	(R3)	Zxing	399	27K	118 (29.6%)
PROMISE	(P1)	Ant v1.7	745	209K	166 (22.3%)
	(P2)	Camel v1.6	965	113K	188 (19.5%)
	(P3)	Ivy v1.4	241	59K	16 (6.6%)
	(P4)	Jedit v4.0	306	145K	75 (24.5%)
	(P5)	Log4j v1.0	135	22K	34 (25.2%)
	(P6)	Lucene v2.4	340	103K	203 (59.7%)
	(P7)	POI v3	442	129K	281 (63.6%)
	(P8)	Tomcat v6.0	858	301K	77 (9.0%)
	(P9)	Xalan v2.6	885	412K	411 (46.4%)
	(P10)	Xerces v1.3	453	167K	69 (15.2%)

defective files in this dataset is relatively lower than the other two datasets (i.e., 9.3% to 39.8%).

2) ReLink dataset was collected by Wu et al. (2011), and the defect information in this dataset was manually verified. ReLink dataset has 26 metrics. Projects in this dataset are relatively small (e.g., project OpenIntents Safe has the least number of files). This dataset has a moderate ratio of defective files (i.e., 29.6% to 50.5%).
3) PROMISE dataset was prepared by Jureczko and Madeyski (2010). We select the same ten projects as in our prior study (Zhang et al. 2016). PROMISE dataset has 20 metrics. PROMISE dataset has the most diverse characteristics of projects, such as the number of files ranges from 135 to 965, and the ratio of defective files varies between 6.6 and 63.6%.

3.2 Classifiers for defect prediction

Each classifier has its own advantages when used to build a defect prediction model. For instance, logistic regression is easy to interpret and is widely used (Nam et al. 2013). Naive Bayes is robust for defect prediction using data with observable noises (Kim et al. 2011). In this study, we choose to use logistic regression as the main classifier in RQ2 and RQ3. We further perform the sensitive analysis on the choice of classifiers in RQ4, since not all classifiers are sensitive to data transformations (Kuhn and Johnson 2013). For instance, in the defect prediction literature, data transformations have been reported to have varied impacts on the performance of different classifiers (Jiang et al. 2008; Menzies et al. 2007; Song et al. 2011). In addition to logistic regression, we evaluate the performance of our approaches in RQ4 using six other classifiers such as Bayes net (BN), k-nearest neighbours (IBk), decision tree (J48), naive Bayes (NB), random forest (RF), and random tree (RT).

3.3 Performance measures

In this study, we compute six commonly used measures (i.e., precision, recall, false positive rate, balance, F-measure, and AUC value) to evaluate the performance of cross-project prediction models.

The first five measures can be calculated from the following four numbers: 1) true positive (TP) that counts the number of defective instances successfully predicted as defective instances; 2) true negative (TN) that calculates the number of non-defective instances correctly predicted as non-defective instances; 3) false positive (FP) that is the number of non-defective instances incorrectly predicted as defective instances; and 4) false negative (FN) that measures the number of defective instances wrongly predicted as non-defective instances. The details are described as follows:

Precision (prec) measures the ratio of correctly predicted defective instances. It is defined as: $prec = \frac{TP}{TP+FP}$.

Recall (pd) evaluates the proportion of defective instances that are predicted as defective instances. It is defined as: $pd = \frac{TP}{TP+FN}$.

False Positive Rate (fpr) captures the proportion of non-defective instances that are predicted as defective instances. It is defined as: $fpr = \frac{FP}{FP+TN}$.

Balance is proposed by Menzies et al. (2007) to balance recall and false positive rate. It is defined as: $balance = 1 - \frac{\sqrt{(0-fpr)^2+(1-pd)^2}}{\sqrt{2}}$.

F-measure is the harmonic mean of precision and recall. It is defined as: $F\text{-}measure = \frac{2\times pd\times prec}{pd+prec}$.

The five aforementioned measures depend on the cut-off value, which is used to compute the four numbers TP, TN, FP, and FN. On the other hand, Area Under Curve (AUC) is the area under the receiver operating characteristics (ROC) curve, thus the AUC value is independent of the cut-off value. Therefore, we further compute AUC values to evaluate cross-project defect prediction models as prior studies, such as (Rahman et al. 2012).

4 Motivation study

In this section, we aim to find if the three studied transformation methods have different performances in the context of defect prediction. The investigation is performed from the following two perspectives:

1) if they can equally improve the normality of software metrics.
2) if cross-project defect prediction models built using each of the transformation methods have similar performance.

Accordingly, we formulate two research questions. We now present the findings of each question, along with our motivation and approach.

4.1 RQ1. Are log, Box-Cox, and rank transformations equally effective in increasing the normality of software metrics?

Motivation Data normality can impact the performance of a prediction model, particularly the model that is not tree-based (Kuhn and Johnson 2013). Although log and rank transformations have been applied in defect prediction (e.g., Jiang et al. 2008; Menzies 2007;

Zhang et al. 2014), their capability in improving the normality of software metric values has not been explicitly explored. In addition, the Box-Cox transformation introduced in Section 2.4 has not been used in the defect prediction studies.

To thoroughly examine the impact that transformations have on defect prediction models, it is necessary to investigate if the three transformation methods indeed have different performance in improving the normality of software metric values.

Approach To address this question, software metrics need to be transformed using each of the three transformation methods. As different software metrics exhibit various distributions, we transform the values of each metric independently.

In each project, we apply the log transformation on software metric values to get log transformed values. When applying the Box-Cox transformation, we first apply the steps described in Section 2.4 to estimate the parameter λ using values of a single metric from the same project, and then apply the Box-Cox transformation. To apply the rank transformation, we compute every $10th$ percentile of the distribution of values of a single metric from the same project, and obtain rank transformed values using (2).

On the transformed metric values, the skewness and kurtosis are computed to evaluate the normality. To investigate if transformation improves the normality of software metric values, we test the following null hypothesis for each transformation method:

$H0_{11}$: *there is no difference in the normality of the transformed metric values and the original metric values.*

We conduct paired Wilcoxon rank sum test (Sheskin 2007), with the 95% confidence level (i.e., p-value<0.05). The Wilcoxon rank sum test is a non-parametric statistical test to assess whether two independent distributions are equal. Non-parametric statistical methods make no assumptions about the distribution of assessed variables. If there is a statistical significance, we reject the hypothesis and conclude that the examined transformation significantly changes the normality of software metric values.

Furthermore, we compare the capability of the three transformations in improving the normality of software metric values. We apply paired Wilcoxon rank sum test to evaluate the following null hypothesis, with the 95% confidence level (i.e., p-value<0.05).

$H0_{12}$: *there is no difference in the normality of metric values that are processed by transformations T_a and T_b.*

T_a and T_b denote two different transformations. If there is a statistical significance, we reject the hypothesis and conclude that the corresponding two transformations have different capability in improving data normality. We further compute the Cliff's δ (Romano et al. 2006) to quantify the difference. The Cliff's δ is a nonparametric effect size that does not assume a particular distribution. The difference is negligible if Cliff's $|\delta| < 0.147$, small if Cliff's $0.147 <= |\delta| < 0.330$, medium if Cliff's $0.330 <= |\delta| < 0.474$, and large if Cliff's $|\delta| >= 0.474$.

Findings **All three studied transformations can significantly improve the normality of software metrics**. Figure 2 presents the skewness and kurtosis values of the transformed metric values from all projects. The median skewness and kurtosis of the original metric values among all three datasets are 4 and 22, respectively. It indicates that the original metric values are highly skewed, because the ideal skewness value is between -0.80 and 0.80, and the perfect kurtosis value is zero (see Section 2.1). Using any of the three transformations can make the skewness and kurtosis values become closer to zero (i.e., nearly normally

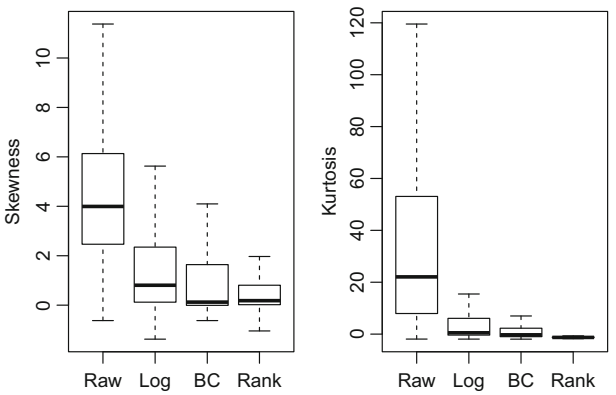

Fig. 2 The boxplot of the skewness and kurtosis values of the metrics that are transformed using each of the three methods on all subject projects ("Raw", "Log", "BC", and "Rank" represent no transformation, the log transformation, the Box-Cox transformation, and the rank transformation, respectively)

distributed). As shown in Table 2, the results of Wilcoxon rank sum tests in the skewness and kurtosis between the transformed values and the original values show statistically significant difference, respectively. Hence, we reject hypothesis $H0_{11}$ for all three transformations. Moreover, the corresponding Cliff's δ is always greater than 0.474 (as shown in Table 2), indicating that each of the studied transformation methods yields a large improvement on the data normality. The capabilities of the three transformations on improving data normality are ordered as: the rank transformation > the Box-Cox transformation > the log transformation. For any pair of the transformations, we reject hypothesis $H0_{12}$ as the p-values of Wilcoxon rank sum tests are always less than 0.05. However, the Cliff's δ is either small or negligible, except the kurtosis value between log and rank transformations and the kurtosis value between Box-Cox and rank transformations.

Regarding the Box-Cox transformation, the estimated parameter λ varies across projects We present the boxplot of the estimated λ values for each project in Fig. 3. The varying λ values across projects suggests that estimating λ values from both the training and target projects can maximize the normality of metrics values in both projects. We observe that few of the estimated λ values are zero ($\lambda = 0$ indicates a log transformation). Therefore, the Box-Cox transformation is not close to the log transformation when dealing with software metrics. In addition, the median value of the estimated λ is often less than zero across projects, showing that most of the estimated λ values are negative. In other words, the

Table 2 The results of comparing cross-project defect prediction models built using the three transformations (n.s. denotes no statistical significance, and **bold** font is used if the corresponding model is better)

Measurement		Log vs. Raw	BC vs. Raw	Rank vs. Raw	Log vs. BC	Log vs. Rank	BC vs. Rank
Skewness	(δ)	**0.620**	**0.694**	**0.833**	0.159	0.267	-0.005
	(p)	1.60e−96	8.05e−96	9.48e−96	7.01e−48	1.84e−55	2.71e−09
Kurtosis	(δ)	**0.594**	**0.672**	**0.859**	0.238	**0.691**	0.476
	(p)	9.11e−95	4.85e−95	1.17e−95	1.15e−74	6.46e−94	5.28e−82

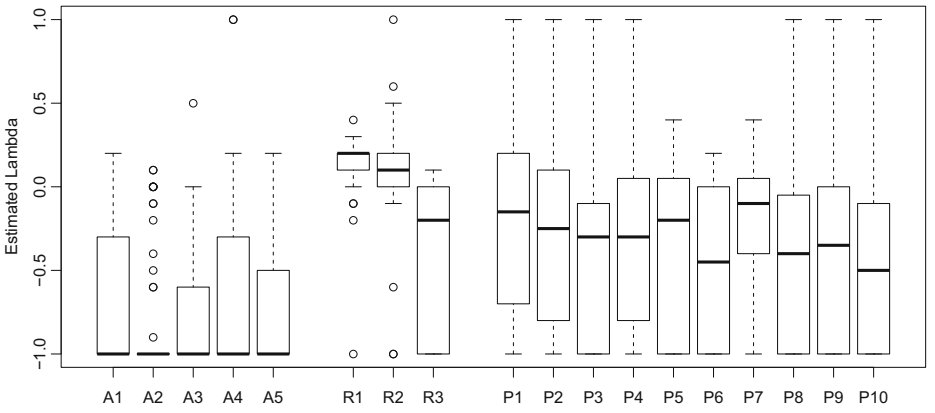

Fig. 3 Boxplot of estimated λ values for metrics in each project. (The full name of each project is presented in Table 1)

Box-Cox transformation tends to reverse the order of metric values, i.e., making larger metric values smaller, and vice versa. The reversed order of metric values does not affect the performance of defect prediction models, since both the training and testing projects are treated in the same way. However, researchers should keep in mind of such possible alteration of the order of metric values when interpreting models built with the Box-Cox transformation.

> *The log, the Box-Cox, and the rank transformations can significantly improve the normality of software metrics values. The three transformations have similar ability to improve data normality, with small or negligible difference.*

4.2 RQ2. Do different transformations result in distinct predictions in cross-project defect prediction models?

Motivation Although all three transformations can effectively improve the normality of software metric values, it is unclear if cross-project defect prediction models are impacted by applying different transformations. We are interested to find if the same performance of cross-project defect prediction models could be achieved, when applying each transformation method. In particular, we want to 1) compare the overall performance (e.g., F-measure and AUC value) of cross-project defect prediction models built using the three transformations; and 2) examine if the three transformations result in distinct predictions in the cross-project setting.

Approach To address this question, we build cross-project prediction models using all possible pairs of the training and target projects in each dataset. In AEEEM, ReLink, and PROMISE datasets, there are 5, 3, and 10 projects, respectively. Therefore, the total number of possible pairs for cross-project defect prediction in the three datasets are 20 ($= 5 \times 4$), 6 ($= 3 \times 2$), and 90 ($= 10 \times 9$), respectively.

Table 3 Contingency matrix to perform McNemar's test

M_1 \ M_2	Correct prediction	Wrong prediction
Correct prediction	N_{cc}	N_{cw}
Wrong prediction	N_{wc}	N_{ww}

For each pair of the training and target projects, we build three models. Each model is built using metrics transformed by one of the three studied transformation methods. As some metrics correlate with other metrics, we perform the correlation analysis to remove the redundancy among software metrics. To measure correlation, we compute Spearman's ρ (Sheskin 2007) that is more robust to outliers (Triola 2004) and preferred in the presence of ties (Sheskin 2007). We define the distance between each pair of software metrics as $1 - \|\rho\|^2$, where ρ is their correlation. We perform hierarchical clustering using R function *hclust*[3] and obtain the metrics with a threshold of $\|\rho\| < 0.8$ (Succi et al. 2005; Selim et al. 2010; Fukushima et al. 2014) using R function *cutree*.[4]

To apply the same Box-Cox transformation on the training and target projects, we estimate λ values for each metric using both projects (see Section 2.4). Similarly, for the rank transformation, we calculate every $10th$ percentile of the values of each metric using both projects.

We apply logistic regression to build cross-project prediction models using transformed values of the training project, and apply the models on the target project. Next, we examine the impact of three transformation methods on cross-project defect prediction from two perspectives:

1) The overall performance: We compute precision, recall, false positive rate, balance, F-measure, and AUC value to measure the overall performance of these models. To compare the overall performance of the models built using the three transformations, we test the following null hypothesis.

$H0_{21}$: *there is no difference between the performance of models built with transformations T_a and T_b.*

T_a and T_b represent two different transformations. We apply paired Wilcoxon rank sum test with the 95% confidence level (i.e., p-value$<$0.05). If there is a statistical significance, we reject null hypothesis $H0_{21}$ and further compute Cliff's δ to quantify the difference.

2) The prediction error: To evaluate if different transformations result in distinct predictions, we compare the prediction errors among models built using the three transformations. To this end, we test the following null hypothesis using McNemar's test with the 95% confidence level (i.e., p-value$<$0.05).

$H0_{22}$: *there is no difference between the error rate of models built with transformations T_a and T_b.*

McNemar's test is commonly used to compare prediction errors of two prediction models (Japkowicz and Shah 2011). As a nonparametric test, it makes no assumptions on the distribution of a subject variable. McNemar's test is applicable only if two models are applied on

[3] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/hclust.html

[4] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/cutree.html

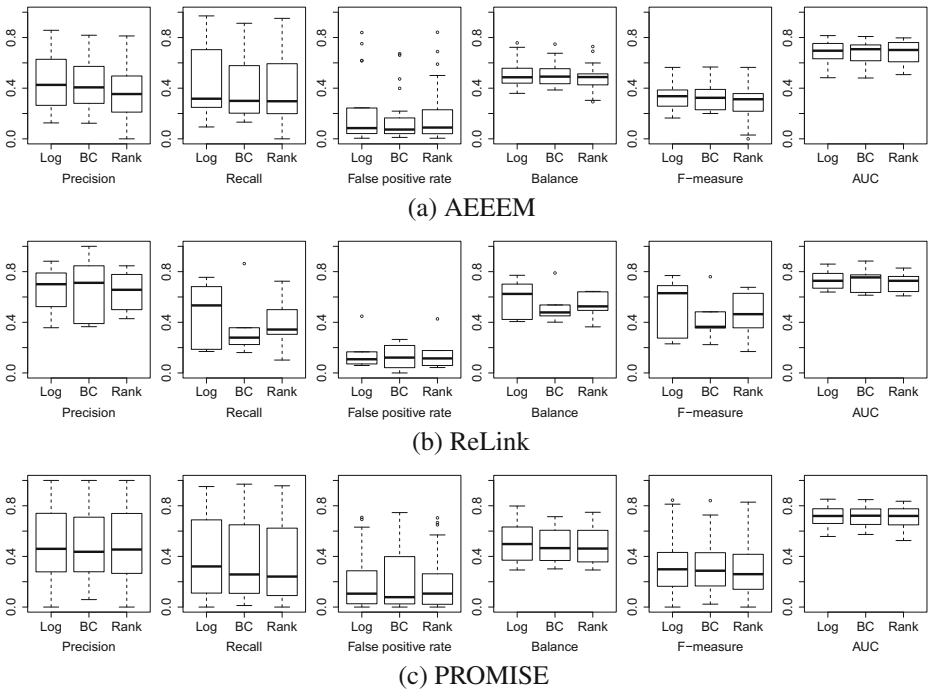

Fig. 4 The boxplots of the six performance measures with the three transformations

the same dataset with separated training and target sets. In this study, our models are built on the same training project, and applied on the same target project that is different from the training project. Therefore, McNemar's test is applicable to our study.

To perform McNemar's test, we need to compute a contingency matrix (see Table 3) based on the predictions produced by two models (i.e., M_1 and M_2). In the contingency matrix, N_{cc} is the number of instances that both models achieve correct predictions; N_{cw} is the number of instances that model M_1 makes correct prediction, but model M_2 has wrong prediction; N_{wc} is the number of instances that model M_1 makes wrong prediction, but model M_2 produces correct prediction; and N_{ww} is the number of instances that both models result in wrong predictions.

The null hypothesis of McNemar's test is that both models M_1 and M_2 have the same error rates. We apply R function *mcnemar.exact* from R package *exact2x2*[5] to perform McNemar's test. The result of McNemar's test can only indicate if there exists a statistical significance, but can not show how much the difference is. Therefore, we further compute odds ratio (OR) to measure the effect size of the difference. The odds ratio measures the degree of the wrong prediction made by one model over the other model. We compute the odds ratio using the equation $OR = \frac{N_{cw}}{N_{wc}}$ (Breslow and Day 1980). An OR = 1 means that both models make the same amount of wrong predictions while the other model makes the correct prediction. An OR<1 indicates that model M_1 makes less wrong predictions than model M_2, and vice versa. An OR greater or less than implies a larger difference in the two models.

[5] https://cran.r-project.org/web/packages/exact2x2/index.html

Findings Applying the three transformation methods yields a similar performance of cross-project prediction models. Figure 4 depicts the boxplots of the six performance measures on the prediction model that is obtained using each of the three transformations. Table 4 presents the average performance measures of models built using the three transformations for each data set. The results of Wilcoxon rank sum test show that in overall there is no significant difference among the three transformation methods. Hence, we can not reject the null hypothesis $H0_{21}$ for all the cases. We conclude that the performance of cross-project prediction models built using the three transformations are similar. This finding is consistent to our previous work (Zhang et al. 2014) that rank and log transformations have a similar power for cross-project predictions, as well as the work of Jiang et al. (2008). Although the rank transformation significantly outperforms the log and the Box-Cox transformations in improving the normality of software metric values in terms of both skewness and kurtosis (see **RQ1**), the predictive power of models built using the rank transformation does not outperform models built using either the log or the Box-Cox transformations. One possible reason is that information is lost after the rank transformation, which may offset potential benefits of using well transformed metrics.

The predicted defective files are not consistent among the results of multiple defect prediction models built using different transformation methods. Although having similar overall performances (e.g., F-measure and AUC value), the three models do not

Table 4 The results of comparing cross-project defect prediction models built using the three transformations (n.s. denotes no statistical significance, and **bold** font is used if the corresponding model is better)

Dataset	Measures	Average Performance			Cliff's δ and p-values of the Comparison		
		Log	BC	Rank	Log v.s. BC	Log v.s. Rank	BC v.s. Rank
AEEEM	prec	0.43	0.43	**0.44**	n.s.	n.s.	n.s.
	pd	0.39	**0.40**	0.39	n.s.	n.s.	n.s.
	fpr	**0.19**	**0.19**	0.18	n.s.	n.s.	n.s.
	balance	0.49	**0.51**	0.50	n.s.	n.s.	n.s.
	F-measure	**0.32**	**0.32**	0.31	n.s.	n.s.	n.s.
	AUC	0.67	**0.68**	0.67	n.s.	n.s.	n.s.
ReLink	prec	0.66	**0.67**	0.60	n.s.	n.s.	n.s.
	pd	**0.51**	0.35	0.41	n.s.	n.s.	n.s.
	fpr	**0.18**	0.11	**0.18**	n.s.	n.s.	n.s.
	balance	**0.61**	0.53	0.53	n.s.	n.s.	n.s.
	F-measure	**0.55**	0.44	0.42	n.s.	n.s.	n.s.
	AUC	**0.73**	0.72	0.68	n.s.	n.s.	n.s.
PROMISE	prec	0.49	0.49	**0.50**	n.s.	n.s.	n.s.
	pd	**0.37**	0.35	0.36	n.s.	n.s.	n.s.
	fpr	**0.18**	**0.18**	**0.18**	n.s.	n.s.	n.s.
	balance	**0.49**	0.48	**0.49**	n.s.	n.s.	n.s.
	F-measure	**0.30**	0.29	**0.30**	n.s.	n.s.	n.s.
	AUC	0.69	**0.70**	**0.70**	n.s.	n.s.	n.s.
The number of wins.		11	8	7	–	–	–

Table 5 The *p*-values of McNemar's test and odds ratio (OR)

Dataset	Training	Target	Log v.s. BC		Log v.s. Rank		BC v.s. Rank	
	Project	Project	OR	p-value	OR	p-value	OR	p-value
AEEEM	Eclipse	Mylyn	–	n.s.	0.519	5.56*e*-04	0.551	8.36*e*-04
	Eclipse	PDE	0.418	3.78*e*-04	0.412	5.46*e*-05	–	n.s.
	Equinox	Eclipse	11.8	4.22*e*-29	–	n.s.	0.056	4.12*e*-33
	Equinox	Lucene	11.6	1.43*e*-28	–	n.s.	0.125	7.03*e*-23
	Equinox	Mylyn	5.77	3.80*e*-19	6.00	1.36*e*-63	3.09	2.57*e*-31
	Equinox	PDE	5.40	9.65*e*-27	0.497	9.83*e*-08	0.164	6.27*e*-43
	Lucene	Eclipse	3.77	1.58*e*-12	0.712	0.03	0.204	4.35*e*-19
	Lucene	Equinox	6.50	0.01	–	n.s.	0.353	0.05
	Lucene	Mylyn	3.96	1.82*e*-12	0.690	0.04	0.248	2.79*e*-16
	Lucene	PDE	–	n.s.	0.470	4.65*e*-07	0.378	1.78*e*-10
	Mylyn	Equinox	–	n.s.	0.091	3.59*e*-05	0.133	2.35*e*-03
	Mylyn	PDE	–	n.s.	–	n.s.	0.478	4.90*e*-03
	PDE	Eclipse	–	n.s.	1.78	4.20*e*-03	1.55	0.04
	PDE	Lucene	–	n.s.	1.88	0.04	3.00	7.17*e*-04
	PDE	Mylyn	–	n.s.	–	n.s.	0.468	1.20*e*-04
	# of significance (%)		**8/20 (40%)**		**10/20 (50%)**		**14/20 (70%)**	
ReLink	Apache	Zxing	0.467	4.27*e*-03	–	n.s.	1.81	0.02
	Zxing	Apache	0.316	3.06*e*-04	0.286	2.47*e*-04	–	n.s.
	# of significance (%)		**2/6 (33%)**		**1/6 (17%)**		**1/6 (17%)**	
PROMISE	Ant	Camel	0.362	6.34*e*-06	–	n.s.	2.50	5.31*e*-05
	Ant	Log4j	0.513	0.02	–	n.s.	2.44	2.67*e*-03
	Ant	Lucene24	4.25	0.01	–	n.s.	–	n.s.
	Ant	Tomcat	3.56	4.31*e*-04	–	n.s.	0.368	1.20*e*-03
	Ant	Xalan	2.70	0.01	0.443	3.74*e*-04	0.311	2.03*e*-07
	Camel	Ivy	–	n.s.	–	n.s.	0.200	0.04
	Camel	Lucene24	0.393	0.01	0.370	0.01	–	n.s.
	Camel	Poi3	0.188	4.43*e*-03	0.250	0.01	–	n.s.
	Camel	Tomcat	3.00	8.58*e*-06	2.17	2.12*e*-03	0.278	0.01
	Camel	Xalan	0.477	0.01	0.527	0.01	–	n.s.
	Ivy	Xalan	0.531	0.04	0.404	7.56*e*-04	0.480	4.70*e*-02
	JEdit	Camel	–	n.s.	0.415	2.23*e*-03	0.435	1.86*e*-03
	JEdit	Lucene24	0.125	0.04	–	n.s.	5.00	0.04
	JEdit	Poi3	0.200	0.04	–	n.s.	2.63	0.02
	JEdit	Tomcat	–	n.s.	0.304	0.01	0.333	0.01
	Log4j	Ant	1.93	3.89*e*-06	2	3.00*e*-03	0.619	2.67*e*-03
	Log4j	Camel	2.99	8.81*e*-17	4.55	5.25*e*-12	0.471	1.83*e*-06
	Log4j	Ivy	6.82	7.37*e*-13	5.20	1.92*e*-04	0.189	3.38*e*-08
	Log4j	JEdit	–	n.s.	4.60	9.12*e*-04	–	n.s.
	Log4j	Lucene24	0.400	1.32*e*-05	–	n.s.	2.94	6.74*e*-05
	Log4j	Poi3	0.210	3.31*e*-18	0.414	2.22*e*-04	5.17	2.26*e*-13
	Log4j	Tomcat	7.68	2.82*e*-29	3.00	7.73*e*-05	0.179	1.35*e*-20

Table 5 (continued)

Dataset	Training Project	Target Project	Log v.s. BC		Log v.s. Rank		BC v.s. Rank	
			OR	p-value	OR	p-value	OR	p-value
	Log4j	Xerces	2.77	1.40*e*-03	–	n.s.	0.278	1.56*e*-04
	Lucene24	Ant	0.240	8.78*e*-04	–	n.s.	3.75	4.72*e*-04
	Lucene24	Camel	0.187	3.01*e*-11	–	n.s.	4.28	7.25*e*-10
	Lucene24	Ivy	41.0	1.96*e*-11	–	n.s.	0.020	9.06*e*-14
	Lucene24	JEdit	8.60	1.37*e*-08	12.0	3.42*e*-03	0.206	2.53*e*-05
	Lucene24	Log4j	0.205	5.54*e*-06	–	n.s.	5.13	1.96*e*-06
	Lucene24	Tomcat	0.042	7.16*e*-18	–	n.s.	14.0	1.08*e*-18
	Lucene24	Xalan	–	n.s.	0.375	2.87*e*-04	0.406	0.01
	Lucene24	Xerces	–	n.s.	–	n.s.	2.83	0.03
	Poi3	Camel	6.36	1.60*e*-14	3.65	1.99*e*-10	–	n.s.
	Poi3	Ivy	12.0	3.42*e*-03	–	n.s.	0.125	4.92*e*-05
	Poi3	JEdit	3.40	3.88*e*-04	–	n.s.	0.462	0.03
	Poi3	Log4j	0.143	1.01*e*-07	–	n.s.	3.90	3.85*e*-05
	Poi3	Tomcat	–	n.s.	–	n.s.	1.82	0.01
	Poi3	Xerces	0.054	2.84*e*-09	–	n.s.	1.85	0.01
	Xalan	Camel	0.269	2.73*e*-24	0.542	0.01	3.57	8.25*e*-21
	Xalan	Ivy	0.019	3.11*e*-15	0.400	0.02	6.43	6.97*e*-08
	Xalan	Log4j	0.323	1.45*e*-03	–	n.s.	2.40	4.60*e*-03
	Xalan	Poi3	0.407	3.13*e*-04	–	n.s.	1.875	0.02
	Xalan	Tomcat	8.25	1.81*e*-12	–	n.s.	0.274	3.59*e*-07
	Xalan	Xerces	2.67	0.01	–	n.s.	–	n.s.
	Xerces	Ant	0.373	1.09*e*-14	–	n.s.	2.55	4.56*e*-13
	Xerces	Camel	0.521	0.01	0.370	3.37*e*-04	–	n.s.
	Xerces	Ivy	0.176	2.58*e*-03	–	n.s.	–	n.s.
	Xerces	JEdit	0.359	1.27*e*-05	0.353	1.64*e*-03	1.70	0.03
	Xerces	Log4j	0.500	0.02	–	n.s.	2.35	3.20*e*-03
	Xerces	Lucene24	3.13	5.78*e*-05	–	n.s.	0.255	1.88*e*-06
	Xerces	Poi3	2.27	0.03	–	n.s.	0.333	3.93*e*-03
	Xerces	Tomcat	0.455	3.69*e*-03	0.302	9.10*e*-06	–	n.s.
	Xerces	Xalan	2.23	2.28*e*-05	–	n.s.	0.532	1.18*e*-03
	# of significance (%)		**45/90 (50%)**		**22/90 (24%)**		**42/90 (47%)**	

necessarily have similar prediction errors. More specifically, when some models make wrong prediction on a file, other models may make correct prediction on the same file. To the best of our knowledge, such distinct prediction existing in models built with various transformation methods is overlooked in prior studies. We conjecture that it might improve the predictive power of cross-project defect prediction models, if integrating multiple models, with each model built using different transformation methods.

The detailed results of McNemar's tests are presented in Table 5. We observe that in AEEEM dataset, the prediction error of using the log transformation is significantly

different from using Box-Cox and rank transformations in 40 and 50% of all cross-project prediction models, respectively. The prediction errors of using Box-Cox and rank transformations are significantly different in 70% of all cross-project prediction models. In cases with a statistically significant difference, we can reject the null hypothesis $H0_{22}$ and conclude that the three models built using each of the three transformation methods do not consistently make the same predictions on the same file. In other words, each transformation method could capture different aspects of the metric values. Similar findings are observed in ReLink and PROMISE datasets.

> *Cross-project prediction models built using the three transformations have a similar overall performance, but their prediction errors are statistically significantly different in many cases. The defect prediction models built using each of the three transformation methods do not consistently make the same prediction on the same file.*

5 Our approach

Transformations may alter the nature of software metrics. Applying various transformations on software metrics captures different perspectives of software metrics. In Section 3, we find that cross-project defect prediction models built using different transformations do not always make the same prediction on the same file. This observation motivates us to integrate models built using different transformations.

In this section, we describe our approach to integrate a set of predictions made by models built with multiple transformations. We present the details of our two approaches: 1) the basic approach MT that integrates multiple models; and 2) the enhanced approach MT+ that selects the most appropriate training project for each target project.

5.1 Our basic approach – MT (multiple transformations)

For a pair of training and target projects, we build multiple defect prediction models. Each model is built using one of the three transformation methods. Our approach aims to utilize the information obtained from multiple models other than a single model. The weight of each model is determined by the accuracy of the model on the training data. Figure 5 illustrates the overview of our approach. The details are described as follows.

1) **Notations.** Let $M = \{M_1, \ldots, M_n\}$ represents a set of prediction models built using n transformations. A file f in a target project is represented as X, a vector of all software metrics. $P_{B,i}(X)$ is the predicted probability of defect proneness on file f by model M_i, and $P_{C,i}(X)$ is the predicted probability of file f as a clean file. Thus, $P_{B,i}(X)+P_{C,i}(X) = 1$. We consider a file is defective, if $P_{B,i}(X)$ is greater than 0.5 (Zimmermann et al. 2009).
2) **Computation of the probability of defect proneness.** We use $P_B(X)$ to denote the final probability of defect proneness on file f using all the n models. We compute $P_B(X)$ in the following two ways: 1) weighting the probability $P_{B,i}(X)$ produced by models that consider file f as defective; or 2) weighting the probability $P_{C,i}(X)$

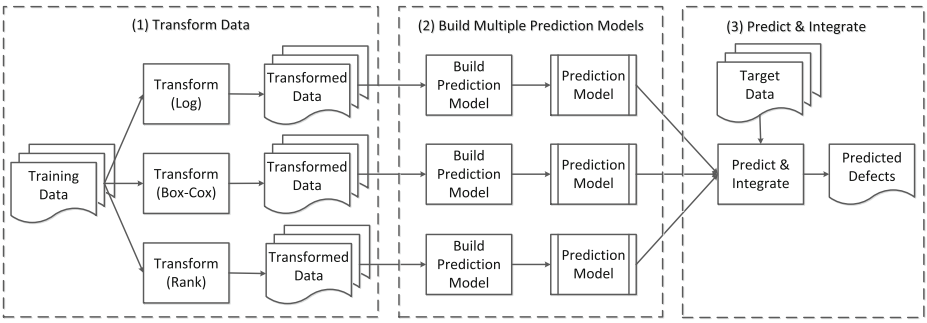

Fig. 5 Overview of our approach MT to integrate models built upon differently transformed data

produced by all models, if no model considers file f as defective. Accordingly, $P_B(X)$ is defined in (6).

$$P_B(X) = \begin{cases} min(1, \frac{\sum_{M_i \in M} w_i \times s_i(X) \times P_{B,i}(X)}{N_B(X)}) & \text{if } N_B(X) > 0 \\ max(0, 1 - \frac{\sum_{M_i \in M} w_i \times P_{C,i}(X)}{n}) & \text{otherwise} \end{cases} \quad (6)$$

where w_i is the weight assigned to model M_i; $s_i(X)$ is the selector for model M_i that determines whether the probability predicted by model M_i is used to compute the final probability of defect proneness or not; and $N_B(X)$ is the number of selected models. The min and max limit $P_B(X)$ in the range [0, 1].

3) **Weight of each model.** A weight is assigned to each model, since the accuracy of different models varies. We consider that models with higher accuracy should be encouraged, and models with lower accuracy should be penalized. Hence, we use the accuracy a_i of a model to obtain the weight w_i for each model M_i. The accuracy a_i is computed on the training data as the proportion of correct predictions (i.e., true positives and true negatives) relative to the total number of predictions. We set $w_i = 0$, if $a_i = 0$. For a model with non-zero accuracy (i.e., $a_i > 0$), we define its weight w_i as $w_i = \frac{a_i}{minAcc}$, where $minAcc$ is the minimum non-zero accuracy among n models.

4) **Selection of each model.** A selector $s_i(X)$ for each model M_i is defined to capture every possible defective file. We consider that a file is defective, if it is predicted as defective by one or more models. As such, the selector $s_i(X)$ is defined in (7). For each file, as shown in (6), the predicted probability of model M_i is used only if the file is predicted as defective by model M_i (i.e., $P_{B,i}(X) > 0.5$).

$$s_i(X) = \begin{cases} 1 & \text{if } P_{B,i}(X) > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

The number of selected models $N_B(X)$ for file f is defined in (8). As applied in (6), $N_B(X)$ is used to normalize the predicted probability of a file that is predicted as defective by at least one model.

$$N_B(X) = \sum_{i=1}^{n} s_i(X) \quad (8)$$

5) **Prediction by the integrated model.** As described in (6), the integrated model considers a file as defective if it is predicted as defective by at least one model. Therefore,

the integrated model increases the recall of defective files but introduces false positives as well. To mitigate the inflation of false positive rate, we increase the cut-off value by multiplying a factor of α, where $\alpha > 1$. In this study, we empirically set $\alpha = 1.2$, thus the default cut-off value 0.5 is increased to 0.6 (=0.5×1.2). Changing the value of α only affects performance measures that reply on the cut-off value of the predicted probability of defect proneness, but does not alter the performance measure AUC. Moreover, automatic determination on the cut-off value has been proposed in our previous work (Zhang et al. 2015).

5.2 Our enhanced approach MT+

Our basic approach integrates defect prediction models built with various transformation methods, but it can not select the most appropriate training project for each target project. If there exist many candidate training projects, it is still unclear how to choose the best project to build the model.

To this end, we further enhance our approach by automatically selecting the most appropriate training project for each target project in unsupervised fashion. For the automatic selection, we propose to apply the knowledge learned from the Box-Cox transformation, as the Box-Cox transformation has a parameter λ reflecting the distribution of the values of the original metrics. For each candidate training project, we compute the best $\widehat{\lambda}$ value of each metric using (4). Then we choose the training project with its average $\widehat{\lambda}$ value close to zero. Note that $\lambda = 0$ indicates a log-normal distribution, therefore the distribution of the metric values of the selected training project is the closest to the log-normal distribution. A log-normal distribution can be observed in many natural growth processes where small changes are accumulated. We conjecture that a software project that grows similar as natural growth processes is more appropriate to build a cross-project defect prediction model.

As our approach in based on MT, we term our enhanced approach as MT+.

6 Evaluation of our approaches

In this section, we evaluate the effectiveness of using our approaches in cross-project defect prediction.

6.1 RQ3. Can our approaches improve the performance of cross-project defect prediction models?

Motivation When building defect prediction models, usually only one transformation method is applied. The findings of **RQ2** suggest that cross-project defect prediction models built using the three transformation methods could make different predictions on the same file. To improve the predictive power of cross-project defect prediction models, we propose the approach MT to integrate multiple transformations (see Section 5) and the enhanced approach MT+ to select the most appropriate training project. In this question, we aim to investigate if our approaches can achieve better performance of cross-project predictions, comparing to models built using only one transformation method.

Approach To evaluate the performance of our approaches, we choose five baselines: 1) *Raw* models built without any transformation; 2) *Raw+* models built without any transformation but the most appropriate training project is selected using the same strategy as MT+;

3) *Min-max* models built using the min-max scaling method (i.e., $min\text{-}max = \frac{x-x_{min}}{x_{max}-x_{min}}$, where x is the raw value of a metric); 4) *Z-score* models built using the normalization method (i.e., $z\text{-}score = \frac{x-\mu}{\sigma}$, where x is the raw value of a metric, μ is the mean of the metric, and σ is the standard deviation of the metric); and 5) *Log* models built using the log transformation.

Similar to **RQ2**, we build cross-project defect prediction models using all possible pairs of the training and target projects in each dataset. We perform the seven transformations on each training project, and build seven logistic regression models, respectively. We apply the seven models on the target project to obtain predictions on each file of the target project. As described in Section 5, we use our approach MT to integrate the predictions of the three models built with log, the Box-Cox, and rank transformations. We use the method described in Section 5.2 to select the most appropriate training projects for Raw+ and MT+. The most appropriate training projects in the three datasets are: "Eclipse JDT Core" followed by "Eclipse PDE UI" (AEEEM), "Apache HTTP Server" followed by "OpenIntents Safe" (ReLink), and "POI v3" followed by "Camel v1.6" (PROMISE).

To evaluate the performance of these models, we calculate precision, recall, false positive rate, balance, F-measure, and AUC value. To investigate if our approaches can improve the performance of cross-project prediction, we test the following null hypothesis for each performance measure:

$H0_{31}$: *there is no difference between the performance of two types of models.*

For instance, we test the null hypothesis $H0_{31}$ between models built with our approach MT and models built with the log transformation. We apply paired Wilcoxon rank sum test with the 95% confidence level (i.e., p-value<0.05). If there is a statistical significance, we reject the hypothesis and conclude that our approaches have statistically significant improvement in the performance of cross-project predictions.

Findings In general, our approaches MT and MT+ statistically significantly improve the performance of cross-project defect prediction, in terms of recall, balance, and F-measure. Table 6 shows the detailed Cliff's δ between the performance measures of each pair of assessed models. When using logistic regression to build the model, our strategy for selecting the most appropriate training project (i.e., Raw+) can statistically significantly improve the performance of models built without transformation (i.e., Raw). Using the min-max scaling method yields statistically significantly lower performance in terms of recall and AUC than using no transformation. We do not observe any significance between the performance of models built with z-score normalization method and without transformation, in all six studied performance measures. Similarly, there is no significant difference in the performance of models built with our approach MT and Raw+. However, comparing to models built without transformation or with Min-max/Z-score/Log transformation, our approach MT significantly improves the performance of cross-project defect prediction in terms of recall, balance, F-measure and AUC, and the only exceptional case is that our approach MT achieves similar performance as log transformation. From Table 6, we can observe that our enhanced approach MT+ can further improve the performance of cross-project defect prediction than Raw+ and MT in terms of recall and F-measure.

Table 7 presents the average value of the six measures for all possible cross-project predictions. In particular, the average F-measure of the models built using the log transformation in AEEEM dataset is 0.34. Log transformation is the most widely used transformation in the literature of defect prediction. Comparing to log transformation, our approach MT+ achieves an improvement of 24% in the average F-measure (i.e., from 0.34 to 0.42). In

Table 6 The Cliff's δ between the performance measures of two assessed models. ('*' indicates p-value< 0.05, '**' for p-value< 0.01, and '***' for p-value< 0.001)

Compared Transformations			prec	pd	fpr	balance	F-measure	AUC
Raw+	v.s.	Raw	n.s.	0.340*	n.s.	0.377*	0.321*	n.s.
Min-max	v.s.	Raw	n.s.	−0.241*	−0.377*	n.s.	n.s.	−0.272*
Z-score	v.s.	Raw	n.s.	n.s.	n.s.	n.s.	n.s.	n.s.
Log	v.s.	Raw	n.s.	0.099*	n.s.	0.111*	n.s.	0.327***
MT	v.s.	Raw	−0.123*	0.321***	0.309**	0.355***	0.302***	0.370***
MT	v.s.	Raw+	n.s.	n.s.	n.s.	n.s.	n.s.	n.s.
MT	v.s.	Min-max	n.s.	0.549***	0.580**	0.438***	0.426***	0.583***
MT	v.s.	Z-score	n.s.	0.525**	0.494*	0.494**	0.432**	0.488***
MT	v.s.	Log	−0.167***	0.216***	0.315***	0.238***	0.231***	n.s.
MT+	v.s.	Raw	−0.216*	0.519**	0.309*	0.481***	0.463***	0.414**
MT+	v.s.	Raw+	n.s.	0.324**	0.216*	n.s.	0.117*	0.259**
MT+	v.s.	Min-max	n.s.	0.617***	0.426**	0.630***	0.568***	0.525***
MT+	v.s.	Z-score	−0.204*	0.562***	0.389**	0.642***	0.593***	0.451**
MT+	v.s.	Log	−0.228**	0.494**	0.327**	0.414**	0.401***	n.s.
MT+	v.s.	MT	n.s.	0.364*	0.216*	0.278**	0.235**	n.s.

ReLink and PROMISE datasets, we have similar observations that the average F-measures are improved from 0.54 to 0.60 (i.e., 11% improvement), and 0.31 to 0.42 (i.e., 29% improvement), respectively. Comparing to the models built with the log transformation, our approach MT only achieves a trivial improvement in terms of F-measure. The p-values of Wilcoxon test on three performance measures (i.e., recall, balance, and F-measure) between models built with the log transformation and our approach MT+ are $1.93e$-03, $1.58e$-03 and $3.28e$-04, respectively. Hence, we reject the null hypothesis $H0_{31}$ for the three performance measures and conclude that our enhanced approach MT+ achieves statistically significant improvement in these measures. We conclude that solely integrating models built multiple transformations is not sufficient to improve the predictive power. It is more effective to improve the performance of cross-project defect prediction by selecting the most appropriate training project for each target project (i.e., MT+).

Furthermore, we present F-measure and the AUC values of cross-project defect prediction models for each project in Table 8. We consider a model wins if it achieves the best performance among all models. We observe that models built without any transformation win one or zero times in terms of F-measure and the AUC value. Models built with Raw+, the log transformation and our basic approach MT achieve similar efficiency. However, models built with our enhanced approach MT+ win 11 and 10 times in terms of F-measure and the AUC value, respectively. This indicates that the strategy used in MT+ to select the most appropriate training project is efficient. We recall that our strategy is to find the project whose metrics are more likely to follow the log-normal distribution as the training project. Our results confirm our assumption that a software project that grows similar as natural growth processes is more appropriate to build a cross-project defect prediction model. In other words, whether metrics of the training project follow the log-normal distribution is an important factor to determine if a defect prediction model succeeds in cross-project defect prediction.

Table 7 Average performance measures of cross-project defect prediction models obtained using five baseline transformations and our approaches (MT and MT+). (Note: **bold** font is used if the corresponding model is better)

Dataset	Measures	Raw	Raw+	Min-max	Z-score	Log	MT	MT+
AEEEM	prec	0.45	**0.58**	0.42	0.50	0.45	0.40	0.48
	pd	0.42	0.30	0.40	0.29	0.43	**0.50**	0.41
	fpr	0.22	**0.06**	0.18	0.11	0.21	0.26	0.11
	balance	0.50	0.50	0.53	0.48	0.51	0.53	**0.57**
	F-measure	0.32	0.37	0.34	0.29	0.34	0.35	**0.42**
	AUC	0.66	0.71	0.65	0.65	0.68	0.69	**0.74**
ReLink	prec	0.62	0.54	0.55	0.57	**0.66**	0.61	0.59
	pd	0.52	0.61	0.25	0.40	0.48	0.55	**0.63**
	fpr	0.23	0.35	**0.10**	0.22	0.16	0.22	0.28
	balance	0.60	**0.63**	0.46	0.52	0.59	0.61	**0.63**
	F-measure	0.54	0.57	0.32	0.41	0.54	0.57	**0.60**
	AUC	0.68	0.67	0.65	0.68	**0.73**	**0.73**	**0.73**
PROMISE	prec	0.50	0.39	0.48	**0.51**	0.50	0.48	0.37
	pd	0.34	0.61	0.31	0.35	0.38	0.45	**0.73**
	fpr	0.16	0.39	**0.15**	0.17	0.18	0.22	0.47
	balance	0.49	**0.59**	0.46	0.48	0.50	0.53	0.58
	F-measure	0.29	0.39	0.27	0.29	0.31	0.35	**0.42**
	AUC	0.67	0.68	0.64	0.67	0.71	**0.72**	0.71
The number of wins		0	4	2	1	2	3	9

A recent work by Nam et al. (2013) has a similar concept as our approach, i.e., using transformations (namely TCA+) to improve the performance of cross-project defect prediction. In particular, Nam et al. (2013) propose a set of rules to automatically select the most appropriate normalization method (e.g., min-max and z-score) for each pair of projects. The TCA+ approach successfully improves the average F-measure by 28% (i.e., 0.32 to 0.41) in the AEEEM dataset, and 24% (i.e., 0.49 to 0.61) in the ReLink dataset. Although TCA+ and MT+ achieve similar improvement, our approach MT+ can automatically determine the most appropriate training project for each target project, therefore relieves practitioners from experimenting with every pair of training and target projects. Moreover, with the training projects selected by our approach MT+, the F-measure of TCA+ can be further improved (i.e., from 0.41 to 0.43 in the AEEEM dataset and from 0.61 to 0.62 in the ReLink dataset).

The false positive rate is increased by our approach MT+ in the ReLink and the PROMISE datasets, but it is controllable We observe that our approach MT+ has a higher false positive rate than the models built with the log transformation in two datasets. For instance, the average false positive rate of the models built using our approach MT+ in the ReLink dataset is acceptable (i.e., 0.28, which is less than 0.3 (Moser et al. 2008)). In the PROMISE dataset, the false positive rate is increased from 0.18 to 0.47. We conjecture that an appropriate cut-off may reduce the excessive false positive rate, since there the AUC value is the same (i.e., both are 0.71 in the PROMISE dataset). Our previous

Table 8 The F-measure and AUC values of cross-project defect prediction models for each project (**bold** font is used if the corresponding model is better)

Dataset	Project	Raw	Raw+	Min-max	Z-score	Log	MT	MT+
(a) F-measure								
AEEEM	Eclipse JDT Core	0.46	**0.57**	0.46	0.33	0.48	0.46	0.56
	Equinox	0.31	0.35	0.40	0.29	0.32	0.40	**0.42**
	Apache Lucene	0.26	0.34	0.27	0.32	0.29	0.29	**0.39**
	Mylyn	0.28	0.32	0.27	0.24	0.28	0.29	**0.34**
	Eclipse PDE UI	0.27	0.27	0.28	0.28	0.31	0.34	**0.38**
ReLink	Apache HTTP Server	0.70	0.68	0.35	0.33	0.69	**0.71**	0.70
	OpenIntents Safe	0.61	0.64	0.44	0.48	0.67	0.69	**0.80**
	Zxing	0.30	0.38	0.17	**0.42**	0.25	0.30	0.30
PROMISE	Ant v1.7	0.37	0.43	0.33	0.33	0.36	0.40	**0.48**
	Camel v1.6	0.18	0.32	0.16	0.24	0.22	0.25	**0.33**
	Ivy v1.4	0.19	0.19	0.20	0.20	0.21	**0.22**	0.17
	Jedit v4.0	0.37	0.34	0.28	0.31	0.40	0.43	**0.46**
	Log4j v1.0	0.26	0.49	0.41	0.36	0.31	0.43	**0.51**
	Lucene v2.4	0.30	0.72	0.31	0.30	0.32	0.37	**0.74**
	POI v3	0.30	0.17	0.19	0.27	0.34	**0.35**	0.30
	Tomcat v6.0	**0.33**	0.28	0.28	0.29	0.32	**0.33**	0.27
	Xalan v2.6	0.38	0.51	0.37	0.31	0.37	0.43	**0.53**
	Xerces v1.3	0.28	**0.43**	0.17	0.28	0.28	0.31	0.38
The number of wins		1	2	0	1	0	4	11
(b) AUC value								
AEEEM	Eclipse JDT Core	0.71	**0.82**	0.71	0.70	0.74	0.71	0.81
	Equinox	0.69	0.74	0.62	0.63	0.68	0.72	**0.80**
	Apache Lucene	0.68	0.69	0.67	0.67	**0.73**	0.72	**0.73**
	Mylyn	0.58	0.60	0.60	0.59	0.59	0.58	**0.61**
	Eclipse PDE UI	0.66	0.71	0.66	0.67	0.67	0.69	**0.74**
ReLink	Apache HTTP Server	0.72	0.70	0.71	0.71	**0.73**	**0.73**	0.70
	OpenIntents Safe	0.77	0.78	0.69	0.71	0.82	0.83	**0.86**
	Zxing	0.55	0.53	0.55	0.61	**0.65**	0.62	0.64
PROMISE	Ant v1.7	0.73	0.71	0.69	0.72	0.75	0.77	**0.79**
	Camel v1.6	0.58	0.57	0.57	0.59	**0.62**	**0.62**	0.59
	Ivy v1.4	0.64	**0.78**	0.62	0.65	0.71	0.72	0.73
	Jedit v4.0	0.69	0.54	0.65	0.68	**0.74**	**0.74**	**0.74**
	Log4j v1.0	0.69	0.79	0.67	0.71	0.75	0.75	**0.84**
	Lucene v2.4	0.64	0.70	0.63	0.64	0.68	0.68	**0.73**
	POI v3	0.69	0.61	0.68	0.70	**0.73**	**0.73**	0.59
	Tomcat v6.0	0.76	0.78	0.71	0.74	0.79	0.80	**0.82**
	Xalan v2.6	0.59	0.54	0.61	0.59	**0.65**	**0.65**	0.56
	Xerces v1.3	0.71	**0.74**	0.61	0.69	**0.74**	0.73	0.72
The number of wins		0	3	0	0	8	5	10

work (Zhang et al. 2015) describes two concrete and practical solutions to reduce the false positive rate by automatically determining the cut-off. Therefore, the inflated false positive rate is controllable.

> *Our approach MT+ can significantly improve the performance of cross-project predictions, in terms of recall, balance, and F-measure. Our approach MT+ is efficient to select the most appropriate training project for each target project.*

6.2 RQ4. Do our approaches work well for other classifiers?

Motivation We have demonstrated the effectiveness of our approach using a single classifier (i.e., logistic regression). However, there are many other classifiers (e.g., random forest and Naive Bayes) that are also frequently used to build defect prediction models (e.g., Jiang et al. 2008; Kim et al. 2011; Menzies et al. 2007; Song et al. 2011). To understand the generalizability of our approaches, it is necessary to study if our approaches can achieve a similar improvement using other classifiers as using logistic regression.

Approach We follow the same approach as in **RQ3**, but using different classifiers to build cross-project prediction models. As described in Section 3.2, we study six classifiers, i.e., Bayes net (BN), k-nearest neighbours (IBk), decision tree (J48), naive Bayes (NB), random forest (RF), and random tree (RT).

To investigate if our approaches can improve the performance of cross-project prediction, we test the following null hypothesis for each classifier. We apply paired Wilcoxon rank sum test with the 95% confidence level (i.e., p-value$<$0.05).

$H0_{41}$: *there is no difference between the performance of our approach and the models built with the log transformation, when using classifier C to build the model.*

Classifier C represents one of our studied classifiers. Same as in **RQ3**, we choose five baselines: *Raw*, *Raw+*, *Min-max*, *Z-score*, and *Log*.

Findings In general, our approach MT+ can improve the performance of cross-project defect prediction models. However, the improvement varies with classifiers. Table 9 presents the average F-measures and AUC values of models built with the log transformation and our approaches using each of the six classifiers. Comparing to models built without any transformation, building models with a single transformation method (i.e., Min-max, Z-score, or log transformation) generally can not improve the performance in terms of F-measure and AUC values. From this perspective, improving the normality of software metrics may not be sufficient to improve the performance of cross-project defect prediction models. However, our approach MT generally improves the performance, indicating that the integration of models built with multiple transformation is beneficial. Moreover, our strategy of selecting the most appropriate training project (i.e., both Raw+ and MT+) can further improve the performance. Therefore, although using a single transformation is not proved beneficial, it is worth integrating models built multiple transformations and selecting the most appropriate training project based on the estimated parameter of the Box-Cox transformation.

Table 9 Average F-measures and AUC values of cross-project defect predictions obtained using the log transformations and our approach (**bold** font is used if the corresponding model is better)

Dataset	Classifier	Raw	Raw+	Min-max	Z-score	Log	MT (%)	MT+ (%)
(a) F-measure								
AEEEM	BN	0.43	0.43	**0.45**	0.42	0.43	0.43 (0%)	**0.45 (5%)**
	IBk	0.27	0.29	0.29	0.27	0.27	0.34 (26%)	**0.37 (37%)**
	J48	0.25	0.25	0.27	0.25	0.25	0.36 (44%)	**0.37 (48%)**
	LR	0.32	0.37	0.34	0.29	0.34	0.35 (3%)	**0.42 (24%)**
	NB	0.40	0.39	**0.41**	**0.41**	**0.41**	0.40 (-2%)	**0.41 (0%)**
	RF	0.30	**0.34**	0.33	0.27	0.29	0.29 (0%)	0.31 (7%)
	RT	0.29	0.28	0.31	0.30	0.29	**0.35 (21%)**	**0.35 (21%)**
	Avg.	0.32	0.34	0.34	0.32	0.33	0.36 (9%)	**0.38 (15%)**
ReLink	BN	0.36	0.46	0.41	0.49	0.36	0.44 (22%)	**0.56 (56%)**
	IBk	0.44	0.43	0.46	0.51	0.44	**0.57 (30%)**	**0.57 (30%)**
	J48	0.31	0.46	0.38	0.39	0.31	0.52 (68%)	**0.64 (106%)**
	LR	0.54	0.57	0.32	0.41	0.54	0.57 (6%)	**0.60 (11%)**
	NB	0.55	0.58	0.34	0.46	0.54	**0.61 (13%)**	0.60 (11%)
	RF	0.43	0.46	0.36	0.50	0.41	0.53 (29%)	**0.65 (59%)**
	RT	0.47	0.51	0.39	0.48	0.39	0.60 (54%)	**0.61 (56%)**
	Avg.	0.44	0.50	0.38	0.46	0.43	0.55 (28%)	**0.60 (40%)**
PROMISE	BN	0.40	**0.46**	0.38	0.39	0.40	0.41 (2%)	**0.46 (15%)**
	IBk	0.30	0.35	0.29	0.31	0.31	0.39 (26%)	**0.41 (32%)**
	J48	0.33	**0.43**	0.32	0.32	0.32	0.37 (16%)	0.41 (28%)
	LR	0.29	0.39	0.27	0.29	0.31	0.35 (13%)	**0.42 (35%)**
	NB	0.36	0.41	0.37	0.36	0.45	**0.48 (7%)**	0.47 (4%)
	RF	0.29	**0.42**	0.27	0.28	0.28	0.29 (4%)	0.40 (43%)
	RT	0.32	0.40	0.31	0.30	0.32	0.41 (28%)	**0.44 (38%)**
	Avg.	0.33	0.41	0.32	0.32	0.34	0.39 (15%)	**0.43 (26%)**
(b) AUC value								
AEEEM	BN	0.71	0.70	**0.73**	0.72	0.71	0.72 (1%)	0.72 (1%)
	IBk	0.57	0.58	0.57	0.56	0.56	0.60 (7%)	**0.63 (12%)**
	J48	0.55	0.55	0.57	0.55	0.55	**0.62 (13%)**	**0.62 (13%)**
	LR	0.66	0.71	0.65	0.65	0.68	0.69 (1%)	**0.74 (9%)**
	NB	0.67	0.65	0.67	0.67	**0.68**	**0.68 (0%)**	**0.68 (0%)**
	RF	0.72	0.70	**0.73**	0.71	0.72	**0.73 (1%)**	0.69 (-4%)
	RT	0.57	0.56	0.58	0.58	0.58	**0.60 (3%)**	**0.60 (3%)**
	Avg.	0.64	0.64	0.64	0.63	0.64	0.66 (3%)	**0.67 (5%)**
ReLink	BN	0.63	0.69	0.65	**0.70**	0.63	0.65 (3%)	**0.70 (11%)**
	IBk	0.57	0.56	0.58	0.59	0.54	0.59 (9%)	**0.61 (13%)**
	J48	0.57	0.64	0.59	0.59	0.57	0.64 (12%)	**0.73 (28%)**
	LR	0.68	0.67	0.65	0.68	**0.73**	**0.73 (0%)**	**0.73 (0%)**
	NB	0.67	0.67	0.61	**0.69**	**0.69**	**0.69 (0%)**	0.68 (-1%)
	RF	0.68	0.71	0.68	0.70	0.68	0.71 (4%)	**0.74 (9%)**
	RT	0.62	**0.65**	0.56	0.60	0.56	0.61 (9%)	0.62 (11%)
	Avg.	0.63	0.66	0.62	0.65	0.63	0.66 (5%)	**0.69 (10%)**

Table 9 (continued)

Dataset	Classifier	Raw	Raw+	Min-max	Z-score	Log	MT (%)	MT+ (%)
PROMISE	BN	0.70	**0.72**	0.70	0.70	0.70	0.70 (0%)	**0.72 (3%)**
	IBk	0.56	0.55	0.56	0.57	0.57	**0.59 (4%)**	0.56 (-2%)
	J48	0.59	**0.64**	0.57	0.57	0.59	0.61 (3%)	0.57 (-3%)
	LR	0.67	0.68	0.64	0.67	0.71	**0.72 (1%)**	0.71 (0%)
	NB	0.66	0.69	0.66	0.67	0.72	**0.73 (1%)**	0.72 (0%)
	RF	0.71	**0.72**	0.70	0.70	0.71	**0.72 (1%)**	**0.72 (1%)**
	RT	0.57	**0.59**	0.56	0.56	0.57	**0.59 (4%)**	0.58 (2%)
	Avg.	0.64	0.66	0.63	0.63	0.65	**0.67 (3%)**	0.65 (0%)

Note: Due to space limit, we only present average performance in the paper. We provide in our supplement file that the boxplots of all six performance measures of cross-project defect prediction models built on each dataset using every classifier

In terms of F-measure, our approach MT can achieve statistically significant improvement over models built with the log transformation, when using IBk (26% to 30%), J48 (16% to 68%), logistic regression (3% to 13%), and random tree (21% to 54%). In terms of the AUC value, four classifiers can benefit from our approach MT, i.e., IBk (4% to 9%), J48 (3% to 13%), random forest (1% to 4%) and random tree (3% to 9%). In all cases, the AUC values remain the same or are increased by using our approach MT.

Our enhanced approach MT+ statistically significantly improves the F-measure for almost all studied classifiers (except Naive Bayes), such as BayesNet (5% to 56%), IBk (30% to 37%), J48 (28% to 106%), logistic regression (11% to 35%), random forest (7% to 59%), and random tree (21% to 56%). The AUC values are statistically significantly improved for one classifier Bayes net (1% to 11%). As shown in Table 9, our enhanced approach MT+ achieves the same or higher AUC values in most cases, except four cases.

In summary, our approaches generally improve the performance of cross-project defect prediction models for multiple classifiers, although the improvement varies with classifiers.

In most cases, our approaches can achieve improvement for multiple classifiers, but the improvement varies with classifiers.

7 Related work

In this section, we first describe prior studies on data transformation in defect prediction, and then present related work regarding cross-project defect prediction.

7.1 Data transformation in defect prediction

Data normality benefits both parametric and non-parametric statistical methods (Osborne 2008), and improves the performance of linear models (Kuhn and Johnson 2013). Transformation is a common method to reduce skewness and improve data normality

(Bishara and Hittner 2014; Gaudard and Karson 2000). Data transformation is essential in defect prediction studies, since many software metrics follow power law distributions (Zhang 2009).

To build a defect prediction model, researchers often apply the natural log transformation (e.g., Menzies et al. 2007; Jiang et al. 2008; Cruz and Ochimizu 2009; Song et al. 2011) and the rank transformation (e.g., Jiang et al. 2008; Zhang et al. 2014) on software metrics.

However, applying the log transformation only benefits some classifiers (e.g., Naive Bayes) (Menzies et al. 2007; Song et al. 2011). For some other classifiers (e.g., decision tree), there is no statistically significant difference (Menzies et al. 2007; Song et al. 2011). Jiang et al. (2008) further compare the performance of defect prediction models built using log and rank transformations. After examining ten classifiers, Jiang et al. (2008) conclude that different classifiers prefer different transformations. For instance, random forest performs better if using the log transformation; and Naive Bayes performs better if using rank transformation. There are also studies that use different transformations for different classifiers. He et al. (2013) apply the rank transformation for Naive Bayes, but use original value for random forest and logistic regression.

7.2 Cross-project defect prediction

The major challenge in cross-project defect prediction is the heterogeneity between the training and target projects (Zimmermann et al. 2009). To address this problem, Menzies et al. (2013) and Bettenburg et al. (2012) investigate if it is beneficial to build models based on instances (e.g., files or classes) that are similar with the target project. Both studies observe that using only the instances that are similar to the target project achieves better performance in cross-project defect prediction models than using all instances. Turhan et al. (2013) recommend to mix the within-project and cross-project data to build a model, since models built with the mixed data outperform models built with only cross-project data. As a summary, the aforementioned studies propose to select similar instances to reduce the heterogeneity between the training and target projects.

An alternative solution is to transform the training and target projects to mitigate their heterogeneity (e.g., Ma et al. 2012; Nam et al. 2013). For instance, Ma et al. (2012) propose an approach to transform the training project using statistical characteristics extracted from the target project. Nam et al. (2013) propose an approach based on transfer component analysis (TCA) to transform the training and target projects together. Jing et al. (2015) propose to unify metric representation between the training and target projects based on the canonical correlation analysis (CCA). These three approaches achieve significant improvement for cross-project predictions. Furthermore, in our prior work (Zhang et al. 2014), we propose a context-aware rank transformation to convert the values of metrics to exactly the same scales across projects. The rank transformation enables us to build a generalized model that on average provides comparable performance as within-project models. Different from the aforementioned studies, we focus on a in-depth analysis of three simple transformation methods (i.e., log, Box-Cox, and rank) in the cross-project setting. We perform a thoroughly analysis on the capability of these transformations on improving the normality of software metrics (i.e., RQ1), and on the benefits of applying these transformations for cross-project predictions (i.e., RQ2).

In addition to reducing the heterogeneity between the training and target projects, there are few other approaches aiming to improve cross-project predictions. For instance, Jing et al. (2016) propose a feature learning method, namely subclass discriminant analysis

(SDA), to effectively solve the class-imbalance problem in cross-project defect prediction. In the case where the training and target projects do not share the same set of metrics, Nam and Kim (2015) provide a solution. Canfora et al. (2013) propose to build multiple models other than a single model. Similarly, we propose to integrate predictions by models built using the three transformations (i.e., RQ3 and RQ4).

8 Threats to validity

In this section, we describe the threats to validity of our study under common guidelines by Yin (2002).

Threats to conclusion validity concern the relation between the treatment and the outcome. The main threats come from our implementation of the three transformations. For instance, we normalize metric values transformed by the Box-Cox transformation to [1, 2]. We clearly describe our treatments of the three transformations, so that researchers can replicate our work and yield the same conclusion when applying the same treatments as our study.

Threats to internal validity concern our selection of subject systems and analysis methods. We choose subjects from the three publicly available data sets that have been used in many other studies (He et al. 2012; Nam et al. 2013; Tantithamthavorn et al. 2016). The selected projects have diversity in size and ratio of defectiveness. The threats to our analysis method come from our choice of random forest to study RQ2 and RQ3. Thus, in RQ4, we examine the effectiveness of our approach using six other classifiers.

Threats to external validity concern the possibility to generalize our results. Our approach is based on log, Box-Cox, and rank transformations. All the three transformations are applicable to software metrics, since many metrics follow power law distributions (Concas et al. 2007; Louridas et al. 2008; Zhang 2009). The diversity in size and defect-proneness of our subject projects helps verify the generalizability of our approach. Nevertheless, further validations on other open source projects and even commercial projects are welcome.

Threats to reliability validity concern the possibility of replicating this study. All three data sets used in this study are publicly accessible. We also provide all necessary details of our experiments on the internet.[6]

9 Conclusion

Cross-project defect prediction is still a challenging problem, since the heterogeneity between the training and target projects (Nam et al. 2013; Zimmermann et al. 2009). For instance, the values of software metrics exhibit varied distributions across projects (Zhang et al. 2013). To this end, (Ma et al. 2012; Nam et al. 2013), and (Zhang et al. 2014) successfully apply appropriate transformations to improve the performance of

[6] http://www.feng-zhang.com/replications/EMSEtransformation.html

cross-project defect prediction models. Apart from such complex transformations, several simple transformations are overlooked. Therefore, we set out to investigate the impact of three simple transformations (i.e., log, Box-Cox, and rank transformations) in the cross-project setting.

In this paper, we observe that all three transformation methods have a similar power to significantly improve the normality of software metrics. Moreover, cross-project prediction models built with each of the three transformation methods achieve similar performances (i.e., precision, recall, false positive rate, balance, F-measure and AUC value). However, we find that these models do not always make the same prediction on the same file, since the results of McNemar's tests clearly show that these models can experience significantly different error rates.

Therefore, we propose an approach MT (Multiple Transformations) to integrate predictions by the cross-project defect prediction models built using each of the three transformation methods (i.e., log, Box-Cox and rank). We further enhance our approach (i.e., MT+) by automatically selecting the most appropriate training project for each target project. We perform an experiment using three public data sets, such as AEEEM (D'Ambros et al. 2010), ReLink (Wu et al. 2011), and PROMISE (Jureczko and Madeyski 2010). The results show that, comparing to the models built with only one transformation method (i.e., the widely used log transformation), our enhanced approach MT+ statistically significantly improves recall, balance, and F-measure in all three data sets. For instance, the average F-measures are improved by 24, 11 and 29% in AEEEM, ReLink, and PROMISE datasets, respectively. Our approaches also leads to the performance improvement in cross-project defect prediction models for various classifiers under study (e.g., random forest). Furthermore, we compare the performance of using untransformed values (Raw), untransformed values but with the selection of the most appropriate training project (Raw+), rescaled values by the min-max method (Min-max), normalized values by the z-score method (Z-score), transformed values by logarithm, and our approaches MT and MT+. We find that using a single transformation method usually can not improve the performance of cross-project defect prediction models. Instead, the models built with multiple transformations should be integrated, and more importantly it is necessary to select the most appropriate training project which can be done in an unsupervised way (i.e., by estimating the parameter of the Box-Cox transformation).

For the future work, we recommend future studies to experiment with our approach for potential gains in the predictive power of cross-project defect prediction models, since our approach introduces little overhead by only adding simple mathematical operations. We are interested to apply more advanced ensemble learners (e.g., Xia et al. 2015; Misirli et al. 2011; Panichella et al. 2014) to enhance our approach. In addition, it worths studying if it is beneficial to apply multiple transformation methods when building other types of prediction models (e.g., effort estimation).

References

Bettenburg N, Nagappan M, Hassan AE (2012) Think locally, act globally: improving defect and effort prediction models. In: Proceedings of the 9th IEEE working conference on mining software repositories, MSR '12, pp 60–69

Bishara AJ, Hittner JB (2014) Reducing bias and error in the correlation coefficient due to nonnormality. Educational and Psychological Measurement http://epm.sagepub.com/content/early/2014/11/10/0013164414557639.full.pdf+html

Box GEP, Cox DR (1964) An analysis of transformations. J R Stat Soc Ser B Methodol 26(2):211–252

Breslow NE, Day NE (1980) Statistical methods in cancer research. vol. 1. the analysis of case-control studies. International Agency for Research on Cancer Scientific Publications 1(32):338

Canfora G, De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2013) Multi-objective cross-project defect prediction. In: 2013 IEEE sixth international conference on software testing, verification and validation (ICST), pp 252–261

Cohen J, Cohen P, West S, Aiken L (2003) Applied multiple Regression/Correlation analysis for the behavioral sciences, 3rd edn. Lawrence Erlbaum, Mahwah, NY, USA

Concas G, Marchesi M, Pinna S, Serra N (2007) Power-laws in a large object-oriented software system. IEEE Trans Softw Eng 33(10):687–708

Cruz A, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects. In: ESEM 2009. 3rd international symposium on empirical software engineering and measurement 2009, pp 460–463

D'Ambros M, Lanza M, Robbes R (2010) An extensive comparison of bug prediction approaches. In: Proceedings of the 7th IEEE working conference on mining software repositories, MSR'10, pp 31–41

Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N (2014) An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the working conference on mining software repositories, ACM, MSR'14, pp 172–181

Gaudard M, Karson M (2000) On estimating the box-cox transformation to normality. Commun Stat Simul Comput 29(2):559–582. doi:10.1080/03610910008813628

Guo W (2014) A unified approach to data transformation and outlier detection using penalized assessment. PhD thesis University of Cincinnati, Arts and Sciences: Mathematical Sciences

Han J, Kamber M, Pei J (2012) Data Mining: concepts and techniques, 3rd edn. Morgan Kaufmann, Boston

He Z, Shu F, Yang Y, Li M, Wang Q (2012) An investigation on the feasibility of cross-project defect prediction. Autom Softw Eng 19(2):167–199

He Z, Peters F, Menzies T, Yang Y (2013) Learning from open-source projects: an empirical study on defect prediction. In: 2013 ACM/IEEE international symposium on empirical software engineering and measurement, pp 45–54

Japkowicz N, Shah M (2011) Evaluating learning algorithms: a classification perspective. Cambridge University Press, New York, NY, USA

Jiang Y, Cukic B, Menzies T (2008) Can data transformation help in the detection of fault-prone modules? In: Proceedings of the 2008 workshop on defects in large software systems, DEFECTS '08, pp 16–20

Jing X, Wu F, Dong X, Qi F, Xu B (2015) Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering, ACM, New York, NY, USA, ESEC/FSE 2015, pp 496–507

Jing XY, Wu F, Dong X, Xu B (2016) An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans Soft Eng PP(99):1–1

Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th international conference on predictive models in software engineering, PROMISE '10, pp 9:1–9:10

Keren G, Lewis C (1993) A handbook for data analysis in the behavioral sciences: statistical issues. Lawrence Erlbaum Hillsdale, NY, USA

Kim S, Zhang H, Wu R, Gong L (2011) Dealing with noise in defect prediction. In: Proceedings of the 33rd international conference on software engineering, ICSE '11, pp 481–490

Kuhn M, Johnson K (2013) Data pre-processing. In: Applied predictive modeling. Springer, New York, pp 27–59

Louridas P, Spinellis D, Vlachos V (2008) Power laws in software. ACM Trans Softw Eng Methodol 18(1):2:1–2:26

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

Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng (TSE) 33(1):2–13

Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T (2013) Local versus global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng 39(6):822–834

Misirli AT, Bener AB, Turhan B (2011) An industrial case study of classifier ensembles for locating software defects. Softw Qual J 19(3):515–536

Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th international conference on software engineering, ICSE '08, pp 181–190

Nagappan N, Ball T, Zeller A (2006) Mining metrics to predict component failures. In: Proceedings of the 28th international conference on software engineering, ACM, ICSE '06, pp 452–461

Nam J, Kim S (2015) Heterogeneous defect prediction. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering, ACM, New York, NY, USA, ESEC/FSE, 2015, pp 508–519

Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: Proceedings of the 2013 international conference on software engineering, ICSE '13, pp 382–391

Osborne JW (2008) 13 best practices in data transformation: the overlooked effect of minimum values, 0 edn, SAGE Publications, Inc., pp 197–205

Osborne JW (2010) Improving your data transformations: applying the box-cox transformation. Practical Assessment Research & Evaluation 15(12)

Panichella A, Oliveto R, De Lucia A (2014) Cross-project defect prediction models: L'union fait la force. In: 2014 software evolution week - IEEE conference on software maintenance, reengineering and reverse engineering (CSMR-WCRE), pp 164–173

Rahman F, Posnett D, Devanbu P (2012) Recalling the "imprecision" of cross-project defect prediction. In: Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering, FSE '12, pp 61:1–61:11

Romano J, Kromrey JD, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In: meeting of the Florida association of institutional research, pp 1–33

Selim G, Barbour L, Shang W, Adams B, Hassan A, Zou Y (2010) Studying the impact of clones on software defects. In: Proceeddings of the 17th working conference on reverse engineering, pp 13–21

Shang H (2014) Selection of the optimal box–cox transformation parameter for modelling and forecasting age-specific fertility. J Popul Res pp 1–11

Sheskin DJ (2007) Handbook of parametric and nonparametric statistical procedures, 4th edn. Chapman & Hall/CRC

Song Q, Jia Z, Shepperd M, Ying S, Liu J (2011) A general software defect-proneness prediction framework. IEEE Trans Softw Eng 37(3):356–370

Succi G, Pedrycz W, Djokic S, Zuliani P, Russo B (2005) An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. Empir Softw Eng 10(1):81–104

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th international conference on software engineering, ACM, ICSE'16, pp 321–332

Triola M (2004) Elementary statistics. Pearson/Addison-Wesley

Turhan B, Misirli AT, Bener AB (2013) Empirical evaluation of the effects of mixed project data on learning defect predictors. Inf Softw Technol 55(6):1101–1118

Wu R, Zhang H, Kim S, Cheung SC (2011) Relink: recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on foundations of software engineering, ESEC/FSE '11, pp 15–25

Xia X, Lo D, Shihab E, Wang X, Yang X (2015) Elblocker: predicting blocking bugs with ensemble imbalance learning. Inf Softw Technol 61:93–106

Yin RK (2002) Case study research: design and methods, 3rd edn. SAGE Publications

Zhang F, Mockus A, Zou Y, Khomh F, Hassan AE (2013) How does context affect the distribution of software maintainability metrics? In: Proceedings of the 29th IEEE international conference on software maintainability, ICSM '13, pp 350–359

Zhang F, Mockus A, Keivanloo I, Zou Y (2014) Towards building a universal defect prediction model. In: Proceedings of the 11th working conference on mining software repositories, MSR '14, pp 41–50

Zhang F, Mockus A, Keivanloo I, Zou Y (2015) Towards building a universal defect prediction model with rank transformed predictors. Empir Soft Eng pp 1–39

Zhang F, Zheng Q, Zou Y, Hassan AE (2016) Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th international conference on software engineering, ICSE '16, pp 309–320

Zhang H (2009) Discovering power laws in computer programs. Inf Process Manag 45(4):477–483
Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ESEC/FSE '09, pp 91–100

Feng Zhang is currently a postdoctoral research fellow with the Department of Electrical and Computer Engineering at Queen's University in Canada. He obtained his PhD degree in Computer Science from Queen's University in 2016. His research interests include empirical software engineering, software re-engineering, mining software repositories, source code analysis, and defect prediction. His research has been published at several top-tier software engineering venues, such as the International Conference on Software Engineering (ICSE), and the Springer Journal of Empirical Software Engineering (EMSE). More about Feng and his work is available online at http://www.feng-zhang.com.

Iman Keivanloo is currently a Post-doctoral researcher in the Department of Electrical and Computer Engineering at Queens University, Canada. He received his MSc in Computer Science from Sharif University of Technology, Iran in 2008. He graduated with his PhD from Concordia University, Montreal, Canada in 2013. His research focuses on the area of source code similarity search and clone detection. He has published over 20 papers in major refereed international journals, conferences and workshops. Dr. Keivanloo has also served as a committee member on several international conferences and workshops in the area of clone detection and program comprehension.

Ying Zou is a Canada Research Chair in Software Evolution. She is an associate professor in the Department of Electrical and Computer Engineering, and cross-appointed to the School of Computing at Queen's University in Canada. She is a visiting scientist of IBM Centers for Advanced Studies, IBM Canada. Her research interests include software engineering, software reengineering, software reverse engineering, software maintenance, and service-oriented architecture. More about Ying and her work is available online at http://post.queensu.ca/~zouy.