An empirical study on predicting defect numbers

Mingming Chen[1,2] and Yutao Ma[2,3,*]
1. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
2. School of Computer, Wuhan University, Wuhan 430072, China
3. WISET Automation Co., Ltd., Wuhan Iron and Steel Group Corporation, Wuhan 430080, China
*E-mail: ytma@whu.edu.cn

Abstract—**Defect prediction is an important activity to make software testing processes more targeted and efficient. Many methods have been proposed to predict the defect-proneness of software components using supervised classification techniques in within- and cross-project scenarios. However, very few prior studies address the above issue from the perspective of predictive analytics. How to make an appropriate decision among different prediction approaches in a given scenario remains unclear. In this paper, we empirically investigate the feasibility of defect numbers prediction with typical regression models in different scenarios. The experiments on six open-source software projects in PROMISE repository show that the prediction model built with Decision Tree Regression seems to be the best estimator in both of the scenarios, and that for all the prediction models, the results yielded in the cross-project scenario can be comparable to (or sometimes better than) those in the within-project scenario when choosing suitable training data. Therefore, the findings provide a useful insight into defect numbers prediction for those new and inactive projects.**

Keywords: defect prediction; predictive analytics; cross-project scenario; regression model

I. INTRODUCTION

Nowadays, defect prediction has attracted much attention from both academia and industry because of its importance in software quality assurance. It has been widely recognized that the defect-proneness of software components (such as classes and code modules) is closely related to a considerable number of software metrics (the so-called features) [1], e.g., static code metrics, code change history, process metrics and network metrics [2], all of which are easy to collect now. Therefore, many defect prediction approaches using statistical methods or machine learning techniques have been proposed to forecast defect-prone software components [3].

To the best of our knowledge, the vast majority of prior defect prediction approaches predict whether a given software component is defect-prone by means of binary classification techniques [3, 4]. However, estimating the defect-proneness of a given set of software components is not enough for software testing in practice due to plenty of criticisms of practicality [4]. Furthermore, if we are able to predict the exact number of bugs in each software component to be tested, software developers or software testers will pay special attention to those software components that contain more defects, which can make testing processes more efficient in the case of limited time and human resources. Thus, defect prediction is not just a simple binary classification problem but a specific problem with predictive analytics [5].

Due to the limitations of data analysis techniques and available training data, the focus of early studies about this topic was on building linear prediction models based on the correlations between defects and some important code features, e.g., lines of code [6] and McCabe's cyclomatic complexity. After object-oriented design metrics and process quality metrics were proposed in the 1990s, the researchers in this filed developed many new prediction models by means of multiple regression analysis [7]. With the development of data mining and machine learning techniques, a few of complex prediction models for the number of defects were presented in recently published literature. For example, Wang *et al.* proposed an approach based on a defect state transition model [8]. The experimental results on open-source software projects showed that the complex models outstripped other competing models in terms of evaluation measures such as the mean absolute error, but their generality and construction cost were questioned.

Interestingly, several recent studies with respect to software defect classification [3, 9, 10] have found that simple classifiers, e.g., Naïve Bayes and Logistic Regression, were able to perform well in both within-project and cross-project scenarios, though those complex ones always achieved high precision. As we know, newly created or unpopular software projects have little historical data available to train any classifiers, which is very similar to the typical problem *cold start* in recommender systems [11]. Hence, cross-project defect prediction (CPDP) emerges as a promising solution to the above issue. Overall, it applies the prediction model learned from other selected projects to a target project [12], and the feasibility of CPDP has been widely examined by the researchers in this field [13, 14]. However, compared with within-project defect prediction (WPDP), in most cases the prediction performance of CPDP is relatively poor on account of the diversity of data distributions between source and target projects [10, 12-14].

As far as we know, very few prior studies evaluated and compared the existing approaches that predict defect numbers in different scenarios. How to make an appropriate decision among those prediction approaches in a given scenario remains unclear. Inspired by the recent studies on software defect classification, in this paper our goal is to validate the feasibility of CPDP approaches (based on regression models) to predicting defects, and to investigate which frequently-used regression model can achieve the best result in both within- and cross-project scenarios. With the pre-designed experiments that were conducted on six open-source software projects in the famous PROMISE repository, we hope our empirical findings could refine the previous work on predicting software defects.

(DOI reference number: 10.18293/SEKE2015-132)

In particular, we provide a more comprehensive and detailed suggestion about choosing appropriate predictive modeling approaches and training data to build a simple, highly cost-effective prediction model according to specific requirements.

The remainder of this paper is organized as follows: Section II introduces the related work; the research questions to be discussed are presented in Section III; Section IV and Section V show the experimental setups and results, respectively; in the end, Section VI concludes this paper and gives a research agenda for our future work.

II. RELATED WORK

Defect prediction has been an active research topic in software engineering for decades [3]. For the theme of this paper, earlier studies focused on analyzing the relationship between defects and code complexity metrics (such as lines of code) with the methods of linear regression [15]. With the rapid development of object-oriented programming and software process management techniques, some of new prediction models began to utilize more types of metrics to predict defect numbers by means of multiple regression analysis [7, 16]. According to a recent survey [1] conducted on 106 papers that were published between 1991 and 2011, the proportions of object-oriented, source code, and process metrics used are about 49%, 27%, and 24%, respectively. Radjenovic *et al.* also find that the Chidamber and Kemerer (CK) metrics are the most commonly used metrics [1]. However, the accuracy of these prediction models is not completely satisfying.

The rise of data mining and machine learning techniques fosters a few complex prediction models using random forest [17], linear discriminant analysis (LDA) [18], artificial neural networks [19], k-nearest neighbors (KNN) algorithm [20], Bayesian networks [21], support vector machines (SVM) [22], and so on. For example, Nguyen *et al.* proposed a similarity-based approach employing an improved KNN algorithm to predict defect numbers [5]. So far, they have been widely used to estimate the defect-proneness of software components, and more details of these approaches can refer to the recent surveys [3, 4]. On the other hand, considering a large number of software metrics, feature subset selection and dimensionality reduction techniques have also been applied to these new defect prediction methods [22, 23], and many empirical studies have demonstrated that they are able to achieve higher accuracy and computing efficiency by removing redundant and irrelevant software metrics [10].

Generally speaking, most of the above-mentioned studies about defect prediction are conducted in WPDP settings, because this is intuitive and easy to use. But, WPDP is not always practicable when lacking historical defect data in a given project. So, CPDP models are investigated to predict defect-prone software components according to the information or knowledge extracted from other similar software projects.

To the best of our knowledge, CPDP was first introduced to this field by Briand *et al.* [24]. They trained a prediction model according to the Xposem project, and used it to predict the defect-proneness of the Jwriter project. The experimental result showed that CPDP outperformed a random prediction model. Then, Zimmermann *et al.* [12] employed 622 cross-project combinations among 12 open-source software projects to validate the feasibility of CPDP, but they found that only 21 out of 622 combinations worked successfully. In fact, the quality of cross-project training data, rather than the total quantity of data available from other projects, is more likely to affect the performance of CPDP models to some extent [25]. Hence, how to select the most appropriate cross-project data for a target project has recently become an interesting problem [26]. For example, Turhan *et al.* [26] applied a nearest neighbor filtering technique to filter out those irrelevant project data in the setting of CPDP, leading to a better prediction performance. More discusses on the comparison between WPDP and CPDP please refer to [4, 10, 27, 28]. Unfortunately, very few prior studies paid attention to the issue in question in CPDP settings.

III. RESEARCH QUESTIONS

Regression analysis is a commonly-used, effective method for predictive analytics, which analyzes current and historical data to make predictions on future or unknown events by estimating the relationships among different variables. In this paper, we investigate the problem related to defect numbers prediction by means of regression analysis. More specifically, we attempt to find empirical evidence to address the following two research questions:

- **RQ1**: *Which type of regression models is the most suitable approach to predicting the number of defects in different scenarios?*

As mentioned earlier, many prediction models, built with regression analysis (such as linear regression, Bayesian regression, and SVM regression), have been used to predict the number of defects. They are a group with excellence as well as shortcomings. So, we should take into consideration various factors (rather than just accuracy) when applying them to different types of actual projects with limited resources, which is required to make an optimal (or near-optimal) tradeoff among generality, performance and construction cost. That is, we want to find one or more appropriate regression models that can be used in different scenarios, because the previous studies about defect-proneness prediction have showed that the classifiers which are simple and easy to use tend to perform well in both within- and cross-project scenarios [10, 27]. In particular, is this still practicable for defect numbers prediction?

- **RQ2**: *Are the accuracies of CPDP regression models under discussion comparable to those of WPDP regression models?*

It is acknowledged that the defect prediction models trained from the same project are, in general, better than those trained from other similar projects, because different projects may have different contextual characteristics, e.g., development process, developers, and project organization. But, does it definitely happen when training data and test data have similar distributional features? So, we attempt to empirically compare the performance differences among the regression models discussed in this paper in both within- and cross-project scenarios. Note that, if the distributions of two sets of prediction results (*A* and *B*) have no statistically significant difference, in our opinion, *A* is comparable to *B*.

IV. Experimental Setups

A. Data Collection

To validate the feasibility of defect numbers prediction in the cross-project scenario, six open-source software projects with 26 releases collected from the online, publicly available PROMISE repository are used as our experimental data set. The brief introduction to all releases of the projects is shown in Table I, where *#Instances* indicates the number of instances (class files), *#Defects* denotes the total number of defects in the release, *%Defect* represents the percentage of defect-prone instances, and *Max* is the maximum value of defects.

Due to space limitations, the list of software metrics used as features in different regression models please refer to [10]. In our experiments, there are 20 independent variables (such as the CK metrics suite and LOC) and one dependent variable (the number of degects), and the goal of our experiments is to estimate the prediction results calculated using six commonly used regression models in different scenarios.

TABLE I. Brief Introduction to the Experimental Data Set

Project	Release	#Instances	#Defects	%Defect	Max
Ant	Ant-1.3	125	33	16.0%	3
	Ant-1.4	178	47	22.5%	3
	Ant-1.5	293	35	10.9%	2
	Ant-1.6	351	184	26.2%	10
	Ant-1.7	745	338	22.3%	10
Camel	Camel-1.0	339	14	3.4%	2
	Camel-1.2	608	522	35.5%	28
	Camel-1.4	872	335	16.6%	17
	Camel-1.6	965	500	19.5%	28
Forrest	Forrest-0.6	6	1	16.7%	1
	Forrest-0.7	29	15	17.2%	8
	Forrest-0.8	32	6	6.3%	4
Jedit	Jedit-3.2	272	382	33.1%	45
	Jedit-4.0	306	226	24.5%	23
	Jedit-4.1	312	217	25.3%	17
	Jedit-4.2	267	106	13.1%	10
	Jedit-4.3	492	12	2.2%	2
Prop	Prop-1	18471	5493	14.8%	37
	Prop-2	23014	4096	10.6%	27
	Prop-3	10274	1640	11.5%	11
	Prop-4	8718	1362	9.6%	22
	Prop-5	8516	1930	15.3%	19
	Prop-6	660	79	10.0%	4
Synapse	Synapse-1.0	157	21	10.2%	4
	Synapse-1.1	222	99	27.0%	7
	Synapse-1.2	256	145	33.6%	9

B. Experiment Design

To answer the two research questions presented in Section III, the overall framework of our experiments and empirical analysis is shown in Figure 1.

First, for a target release (test data) of a given project, two training data selection approaches (viz. application scenarios) are considered in our experiments. (1) WPDP: all historical releases prior to the target release within the same project are used as training data; and (2) CPDP: all available releases from the most suitable project (rather than from the same project) are used as training data. Take Ant-1.7 as an example, the four releases from the same project, namely Ant-1.3, Ant-1.4, Ant-1.5, and Ant-1.6, are selected as training data in the within-project scenario, and all of the releases from the Camel project are chosen as training data in the cross-project scenarios (see the example in Figure 1).

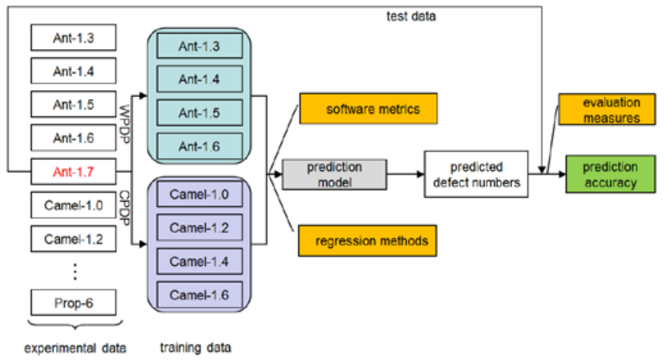

Figure 1. Overall Framework of Our Experiments: An Example of Ant-1.7

Second, we determine the number of experiments according to the experimental data and the training data selection methods. Because the first release of each project is impossible to be test data in the within-project scenario, there are 20 (4 + 3 + 2 + 4 + 5 + 2 = 20) groups of tests among all the releases of the six projects. To keep the comparison between WPDP and CPDP in the same condition, we select only 20 groups of corresponding tests for CPDP, though there is a total of 26 test data sets.

Third, according to the 20 software metrics (independent variables) and the number of bugs (dependent variable), we build different prediction models based on six commonly used regression methods (see the upcoming subsection) and apply them to 12 (2 × 6 = 12) cases. For each case (in either WPDP or CPCP scenario) in question, we conduct the above-mentioned experiment (viz. prediction) 20 times.

Fourth, we preprocess the predicted defect numbers of all experiments before estimating the experimental results. Since the number of bugs in every software component must be a non-negative integer, we make appropriate adjustments for those original defect numbers if necessary. That is, if the predicted defect number is negative, it will be set to 0; if the result is a positive decimal, it will be set to an integer by a rounding method.

Finally, we compute the accuracy of a prediction model in terms of several evaluation measures, and compare the differences on the distributions of prediction results using statistical methods such as the Wilcoxon signed-rank test.

C. Regression Models

Regression methods assess the expectation of a dependent variable according to a set of given independent variables. More specifically, a regression model can help understand how a dependent variable changes when independent variables vary in training data sets, and tries to estimate the expectation of the dependent variable with those given independent variables in test data sets. For the issue discussed in this paper, we assume that the vector **X** of software metrics includes all independent variables and the numerical value of defect number *y* is the dependent variable. Each regression model under discussion learns a function *y* = *f*(**X**) according to the relationship between **X** and *y* extracted from training data, and then uses the function *f* to predict defect numbers of test data. Considering the motivation of this paper, we select only six typical regression models, excluding those complex ones. To avoid reinventing the wheel all the time, we build and implement these regression models based on the python machine learning library *sklearn*. Unless otherwise specified, the default parameter settings for different regression models used in our experiments are specified by *sklearn*. That is, we do not perform additional optimization for each regression model. The brief introduction to the six regression models is described as follows:

- LR (Linear Regression): it is always used to solve the least squares function of the linear relationship between one or multiple independent variables and one dependent variable.
- BRR (Bayesian Ridge Regression): it is similar to the classical Ridge Regression. The hyper parameters of such type of models are introduced by prior probability and then estimated by maximizing the marginal log likelihood with probabilistic models.
- SVR (Support Vector Regression): it is extended from the well-known Support Vector Machines, which only depends on a subset of training data, because the cost function for building a SVR model ignores any training data close to the prediction results of the model.
- NNR (Nearest Neighbors Regression): it is based on the k-nearest neighbors algorithm, and the regression value of an instance is calculated by the weighted average of its nearest neighbors. And, the weight is settled proportional to the inverse of the distance between the instance and its neighbors.
- DTR (Decision Tree Regression): it learns simple decision rules to approximate the curve of a given training data set, and then predicts the target variable.
- GBR (Gradient Boosting Regression): it is in the form of an ensemble of weak prediction models. Several base estimators are combined with a given learning algorithm in order to improve the prediction accuracy over a single estimator.

D. *Evaluation Measures*

To evaluate the prediction results of our experiments, in this paper we utilize the metrics precision (P) and root mean square error (RMSE), which are described as follows:

- P: Precision addresses the percentage of correctly predicted instances to the total number of test data. The higher the precision, the better accuracy a prediction model achieves.

$$\mathrm{P} = \frac{N_{\hat{r}=r}}{N}, \tag{1}$$

where N is the number of instances in test data, and $N_{\hat{r}=r}$ indicates the number of instances whose predicted values ($\hat{r}$) are equal to their real values (r).

- RMSE: It measures the difference between the values predicted by a model or an estimator and the values actually observed. Compared with the mean absolute error, because of being scale-dependent, RMSE is a good measure of accuracy to compare prediction errors of different models for a given variable, e.g., the number of defects in this paper.

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}|\hat{r}_i - r_i|^2}{N}}. \tag{2}$$

V. Experimental Results

A. *Answer to RQ1*

First, for each prediction model built with a regression model in question, we conduct the pre-designed experiment 20 times in the scenario of WPDP. The prediction results of an example are shown in Table II, where each record in the table is denoted by a two-tuple/pair (P, RMSE). In this example, the last release of each project is used to be test data, and all of the previous releases in the same project are selected as training data. The number in bold in this table indicates the best estimator among the six prediction models. Note that BRR and GBR achieve the same value of precision, but the former is better than the latter because of a smaller RMSE value.

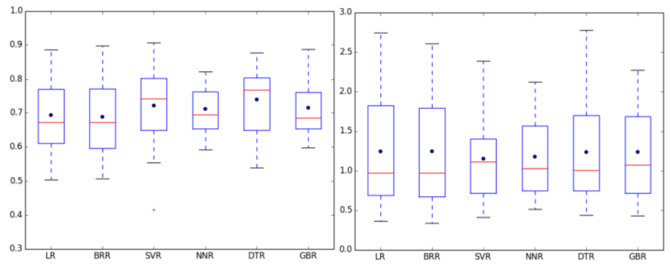

Figure 2. Standardized box plots of evaluation measures for the six prediction models in the scenario of WPDP (P: left, RMSE: right)

Moreover, the distributions of the values of P and RMSE for the 20 experiments are shown in Figure 2, where the X-axis means the prediction models under discussion and the Y-axis indicates the value of an evaluation measure. The legends from the bottom to the top of a standardized box plot are minimum, first quartile, median (red line), third quartile, and maximum. Note that the small cycle within each box is the mean value of the 20 predictions. It is obvious from Figure 2 that the median and mean of DTR are larger than those of the other five prediction models with respect to precision. On the other hand, considering the distribution of RMSE values, DTR is similar to LR and BRR, closely followed by GBR and NNR. To sum up, in the scenario of WPDP, DTR seems to be the best estimator for the experimental data in this paper.

Second, in the scenario of CPDP, we conduct the pre-designed experiment 20 times in a similar way as described for WPDP. The prediction results of an example corresponding to the above example are listed in Table III. In this example, for each target release, we select all available releases from the most appropriate project as training data. That is, each record in this table represents the best outcome among the other projects. Surprisingly, the results about precision in Table III are, on average, higher than those in Table II, implying that CPDP achieves better performance than WPDP in this example. This finding is different from the results of many prior studies [3, 9].

TABLE II. ESTIMATING PREDICTION RESULTSS IN THE SCENARIO OF WPDP: AN EXAMPLE

Project	LR	BRR	SVR	NNR	DTR	GBR
Ant-1.7	0.740, 0.924	0.753, 0.924	**0.771**, 1.227	0.719, 0.987	0.761, 0.943	0.754, 0.921
Camel-1.6	0.616, 1.868	0.614, 1.813	0.550, 1.761	0.694, 1.775	**0.739**, 1.862	0.662, 1.686
Forrest-0.8	0.750, 0.728	0.531, 0.952	**0.906**, 0.810	0.812, 0.847	0.718, 1.457	0.714, 1.794
Jedit-4.3	0.597, 1.954	0.583, 1.941	0.706, 0.832	0.691, 1.651	**0.798**, 1.666	0.680, 2.068
Prop-6	0.886, 0.421	**0.887, 0.419**	0.868, 0.447	0.809, 0.554	0.868, 0.483	0.887, 0.431
Synapse-1.2	0.632, 1.011	**0.668**, 0.988	0.656, 1.208	0.636, 1.077	0.652, 1.034	0.652, 1.019

TABLE III. ESTIMATING PREDICTION RESULTS IN THE SCENARIO OF CPDP: AN EXAMPLE

Project	LR	BRR	SVR	NNR	DTR	GBR
Ant-1.7	0.762, 1.016	0.764, 1.015	0.756, 1.132	0.745, 1.128	0.756, 1.132	**0.775**, 1.113
Camel-1.6	0.771, 1.752	0.773, 1.770	**0.842**, 1.599	0.754, 1.807	0.778, 1.795	0.764, 1.145
Forrest-0.8	0.931, 0.780	**0.938**, 0.791	0.894, 0.792	0.875, 0.829	0.927, 0.740	0.786, 1.804
Jedit-4.3	0.821, 0.666	0.823, 0.664	0.749, 0.687	0.829, 0.813	0.880, 0.486	**0.896**, 0.789
Prop-6	0.779, 0.529	0.826, 0.488	**0.891**, 0.421	0.818, 0.486	0.885, 0.478	0.849, 0.478
Synapse-1.2	0.668, 1.084	0.648, 1.046	**0.711**, 1.240	0.648, 1.137	0.684, 1.093	0.664, 1.108

Similarly, Figure 3 shows that in the scenario of CPDP DTR is also the best estimator when considering precision. With regard to RMSE, DTR is similar to LR, which is next to BRR. Note that any data not included between the whiskers is plotted as a cross in Figure 3.

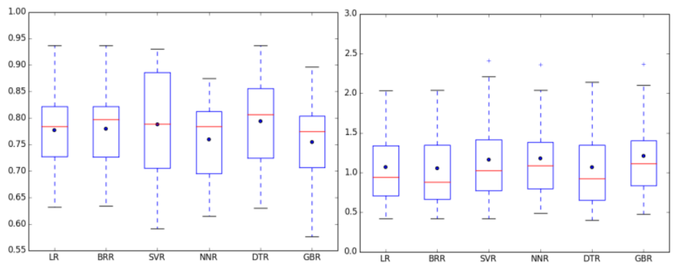

Figure 3. Standardized box plots of evaluation measures for the six prediction models in the scenario of CPDP (P: left, RMSE: right)

The prediction model based on Decision Tree Regression is the best estimator for defect numbers in different scenarios.

B. *Answer to RQ2*

In both within- and cross-project scenarios, the results of a prediction model are actually two groups of samples, and the goal of RQ2 is to assess whether their population mean ranks differ. The Wilcoxon signed-rank test, also known as a non-parametric statistical hypothesis test, is suitable for this case, because the population cannot be assumed to be normally distributed (see Figures 2 and 3, the median and mean within each box are not equal). The software program for such tests is realized based on *scipy*, which is a Python-based software ecosystem for mathematics, science, and engineering.

Assuming that two groups of samples are drawn from the same distribution (*null hypothesis*), the Wilcoxon signed-rank test is executed with an alternative/opposite hypothesis. A *p*-value generated in the test is used to reject the *null hypothesis* in favor of the opposite hypothesis. But, if the *p*-value is more than 0.01, one cannot reject the *null hypothesis*. The test results are shown in Table IV, which highlights that there are no significant differences among WPDP and CPDP models, indicated by the majority of $p > 0.01$ (10/12) for these models evaluated by the two measures, though two exceptions (whose numbers are in bold font) exist with respect to precision. More interestingly, for the two exceptions, the Cliff's effect sizes for NNR and DTR are negative, which suggests that the results of CPDP models are better. From the perspective of statistical analysis, the finding indicates that in this paper CPDP models are comparable to (or sometimes better than) WPDP models with regard to accuracy, largely due to that in some cases we select a large and mature project (Prop) as training data in the scenario of CPDP. Hence, the selection of historical data from those suitable mature projects to train a prediction model for defect numbers is a significant factor for the success of CPDP.

TABLE IV. A COMPARISON OF THE DISTRIBUTIONS OF PREDICTION RESULTS IN BOTH SCENARIOS USING THE WILCOXON SIGNED-RANK TEST

Measure	LR	BRR	SVR	NNR	DTR	GBR
Precision	0.011	0.044	0.030	**0.002**	**0.001**	0.033
RMSE	0.314	0.108	0.941	0.970	0.084	0.296

CPDP models are comparable to (or sometimes better than) WPDP models with respect to prediction performance.

C. *Threats to Validity*

Although we obtain some interesting findings to answer the research questions, there are still potential threats to the validity of our work, one of which concerns the generalization of the results obtained in this paper. The main reasons include the following four aspects: (1) we randomly select six projects (a very small subset of all of the projects) from the PROMISE repository when conducting the experiments; (2) we only use the 20 static code metrics when building prediction models, because the six projects do not include the information of process metrics and social network measures [29]; (3) we select training data in a simple way, though there are many time-consuming but effective selection methods [30]; and (4) we only utilize six typical regression methods without additional optimization for a given data set.

VI. Conclusion and Future Work

Defect numbers prediction is an interesting problem in the field of defect prediction. Compared with the prior studies on defect classifiers (viz. supervised classification models), in this paper we empirically investigate the feasibility of defect numbers prediction using regression methods in both within-project and cross-project defect scenarios. The experiments on six open-source software projects show that those simple, typical regression methods are also able to perform well in different scenarios, e.g., the prediction model built with Decision Tree Regression is proven to be the best estimator in our experiments, and that the six prediction models under discussion can achieve similar (or sometimes even better) results in both within- and cross-project scenarios. The findings would provide useful empirical evidence for software developers or software testers to choose appropriate training data and regression methods in the case of urgent deadlines and limited resources.

Our future work will further investigate the issues related to defect numbers prediction so as to improve prediction precision, including (1) building the best prediction model according to the actual distribution of defects in a given software project, and (2) selecting the most suitable training data for defect numbers prediction using transfer learning techniques [31] in the scenario of CPDP.

Acknowledgment

This work is supported by the National Basic Research Program of China (973 Program) (No. 2014CB340401), the National Natural Science Foundation of China (Nos. 61272111and 61273216), the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232), and the open foundation of Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System in China (No. znss2013B017).

References

[1] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, "Software fault prediction metrics: A systematic literature review," Information and Software Technology, 2013, 55(8): 1397−1418.

[2] A. Meneely, L. Williams, W. Snipes, J. Osborne, "Predicting failures with developer networks and social network analysis," in: Proc. of the 16th ACM SIGSOFT FSE, Atlanta, Georgia, USA, 2008, pp. 13–23.

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

[4] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, 2015, 27: 504–518.

[5] T.T. Nguyen, T.Q. An, V.T. Hai, T.M. Phuong, "Similarity-based and rank-based defect prediction," in: Proc. of the 2014 Int'l Conf. on Adv. Technol. for Commun., Hanoi, Vietnam, 2014, pp. 321−325.

[6] J.E. Gaffney Jr., "Estimating the Number of Faults in Code," IEEE Transactions on Software Engineering, 1984, 10(4): 459−465.

[7] N.E. Fenton, M. Neil, "A Critique of Software Defect Prediction Models," IEEE Transactions on Software Engineering, 1999, 25(5): 675−689.

[8] J. Wang, H. Zhang, "Predicting defect numbers based on defect state transition models," in: Proc. of the 6th ACM-IEEE Int'l Symp. on Empirical Softw. Eng. and Measurement, Sweden, 2012, pp. 191−200.

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

[10] P. He, B. Li, X. Liu, J. Chen, Y.T. Ma, "An empirical study on software defect prediction with a simplified metric set," Information and Software Technology, 2015, 59: 170−190.

[11] A.I. Schein, A. Popescul, L.H. Ungar, D.M. Pennock, "Methods and metrics for cold-start recommendations," in: Proc. of the ACM SIGIR SIGIR'02, Tampere, Finland, 2002, pp. 253−260.

[12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in: Proc. of the ESEC/FSE'09, Netherlands, 2009, pp. 91−100.

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

[14] F. Rahman, D. Posnett, P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in: Proc. of the 20th ACM SIGSOFT FSE, Cary, NC, USA, 2012, p. 61.

[15] H.Y. Zhang, "An Investigation of the Relationships between Lines of Code and Defects," in: Proc. of the 25th IEEE Int'l Conf. on Softw. Maintenance, Edmonton, Alberta, Canada, 2009, pp. 274−283.

[16] F. Lanubile, A. Lonigro, G. Visaggio, "Comparing models for identifying fault-prone software components", in: Proc. of the 7th Int'l Conf. on Softw. Eng. and Knowl. Eng., USA, 1995, pp. 312−319.

[17] A. Kaur, R. Malhotra, "Application of random forest in predicting fault-prone classes," in: Proc. of the ICACTE'08, Thailand, 2008, pp. 37-43.

[18] J.C. Munson, T.M. Khoshgoftaar, "The detection of fault-prone programs," IEEE Trans. Softw. Eng., 1992, 18(5): 423–433.

[19] A. Kaur, P. Sandhu, A. Bra, "Early software fault prediction using real time defect data," in: Proc. of the 2nd Int'l Conf. on Machine Vision, Dubai, UAE, 2009, pp. 242 –245.

[20] G.D. Boetticher, "Nearest neighbor sampling for better defect prediction," in: Proc. of the IEEE PROMISE'05, Missouri, USA, 2005, pp. 1-6.

[21] G. Pai, J. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," IEEE Transactions on Software Engineering, 2007, 33(10): 675–686.

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

[23] K. Gao, T.M. Khoshgoftaar, H. Wang, N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," Softw., Pract. Exper., 2011, 41(5): 579−606.

[24] L.C. Briand, W.L. Melo, J. Wüst, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Trans. Softw. Eng., 2002, 28(7): 706–720.

[25] P. He, B. Li, D. Zhang, Y.T. Ma, "Simplification of Training Data for Cross-Project Defect Prediction," CoRR, abs/1405.0773, 2014.

[26] B. Turhan, T. Menzies, A. Bener, J.D. Stefano, "On the relative value of cross-company and within-company data for defect prediction," Empirical Softw. Eng., 2009, 14(5): 540–578.

[27] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, "Towards building a universal defect prediction model," in: Proc. of the IEEE MSR'14, Hyderabad, India, 2014, pp. 182−191.

[28] B. Turhan, A.T. Misirli, A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," Information and Software Technology, 2013, 55(6): 1101−1118.

[29] T. Zimmermann, N. Nagappan, "Predicting defects using network analysis on dependency graphs," in: Proc. of the 30th Int'l Conf. on Softw. Eng., Leipzig, Germany, 2008, pp. 531–540.

[30] F. Peters, T. Menzies, A. Marcus, "Better cross company defect prediction," in: Proc. of the 10th Workshop on Mining Software Repositories, San Francisco, CA, USA, 2013: 409−418.

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