Multi-project Regression based Approach for Software Defect Number Prediction*

Qiguo Huang ★♯ Chao Ni ★♯ Xiang Chen ★☆ Qing Gu ★ Kaibo Cao ☆

★ State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
☆ School of Computer Science and Technology, Nantong University, Nantong 226019, China

Abstract

Software defect prediction can make software quality assurance (SQA) process more efficient, economic and targeted. Previous studies mainly focused on classifying software modules as defect-prone or not. However, prediction the number of defects for a new software module is rarely investigated. Moreover, these studies built models independently for each project, which may ignore the relatedness among multiple projects. To effectively utilize the relatedness, we propose a novel approach MPR (multi-project regression) for SDNP (software defect number prediction). To verify the effectiveness of MPR, we perform experimental studies on 30 real-world projects and compare our approach with 6 state-of-the-art baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). AAE (Average absolute error) and ARE (average relative error) performance measures are used to evaluate the performance of MPR. The results show MPR can achieve better performance in most cases, which indicates the competitiveness of MPR in the context of SDNP.

1 Introduction

Software defects are introduced unconsciously during the development process of software projects. After the software projects are deployed, defects in the software will produce unexpected behaviors, even cause huge economic loss in worst cases. Therefore project managers want to use software quality assurance methods to detect defects as many as possible. Due to the limitation of testing resources, project managers hope that they can resort to effective methods to identify potential defective modules as early as possible. Software defect prediction [1–3] is one of such effective methods. It helps to identify defective software modules before the testing phase by analyzing some underlying characteristics of the software system and thus subsequently helps in allocating software testing resources optimally and economically. It constructs defect prediction models by mining software repositories (such as version control systems, bug tracking systems) and uses the constructed models to identify potential defective modules in the new projects.

Many studies have been conducted for software defect prediction by using different statistical and machine learning techniques. Most of these studies have focused on classifying whether a software module has defects or not. That means previous studies discreted defect number of program modules into two categories: defective or non-defective [4]. However, such simple data preprocessing may lead to information loss. In addition, predicting defect number for program modules can assist in sorting program modules and then allocating more testing resources to these modules with more defects. In this way, the allocation of testing resources can be further optimized.

To the best of our knowledge, previous studies mainly focused on classifying software modules as defect-prone or not. Only a few studies built and evaluated models for defect number prediction [5–7]. These studies built models independently for each project, which may ignore the relatedness among multiple projects. In this paper, to fully utilize the relatedness among projects and propose a novel approach MPR (multi-project regression) for defect number prediction. To verify the effectiveness of MPR, we perform experimental studies on 30 real-world open-source projects and compare our approach with 6 state-of-the-art baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). AAE (Average absolute error) and ARE (average relative error) are used as performance measures. The results show MPR can achieve better performance in most cases.

The main contributions of the paper can be summarized as follows: (1) to our best knowledge, we firstly propose a novel approach MPR for predicting defect number on multiple projects. This approach can utilize the relatedness among projects and builds many predictors simultaneously. (2) we conduct empirical studies on real-world software projects to demonstrate the effectiveness of MPR. The final results show that MPR can achieve a statistical improvement over classical baselines.

2 Related Work

Previous studies mainly investigated regression based methods for this SDNP. Graves et al. [5] considered a generalized linear regression method and conducted empirical studies on a large-scale telecommunication system. They found a significant correlation among module's age, changes made to the modules and the age of changes. Ostrand et al. [6] employed negative binomial regression (NBR) method. Wang and Zhang [8] proposed BugState,

DOI Reference Number 10.18293 / SEKE2019-019

*: Corresponding Author: Qing Gu and Chao Ni. ♯: Equally contributing authors.

which is based on a defect state transition model. Janes et al. [9] considered three methods (i.e., NBR, zero-inflated NBR, Poisson regression) for 5 real-time telecommunication systems and found that zero-inflated NBR method achieved the best performance. Then Gao and Khoshgoftaar [10] further performed empirical studies on two industrial software projects and their results confirmed the best performance of zero-inflated NBR. Chen et al. [11] conducted empirical studies for 6 regression algorithms. Then found that using decision tree regression can achieve the highest performance in both within-project prediction scenario and cross-project prediction scenario. Yu et al. [12] explored resampling (i.e., SMOTEND and RUSND) and ensemble learning (i.e., AdaBoost.R2) methods. Then they proposed two hybrid methods (i.e., SMOTENDBoost and RUSNDBoost) and these two methods can achieve higher performance. Rathore and Kumar [13] explored the capability of decision tree regression in two different scenarios (i.e., intra-release prediction scenario and inter-release prediction scenario). They [14] compared six methods for SDNP, such as genetic programming, multi-layer perceptron, linear regression, decision tree regression, zero-inflated Poisson regression, and negative binomial regression. Recently, they [15, 16] further considered ensemble learning methods for SDNP.

Based on the above analysis, when performing SDNP for many related projects, we found that the models proposed in previous studies can merely build one model for one project per time and they can not use useful information in other related projects. Different from previous studies, we propose a novel method to build multiple regression models simultaneously for many related projects.

3 Our Proposed MPR Approach

This section first gives the preliminaries on multiple linear regression based on multi-task learning [17]. Then it describes our proposed MPR approach in detail.

3.1 Preliminaries

Definition 1 *(Multiple Liner Regression) Suppose there are m projects to be learnt $\{P^i\}_{i=1}^{m}$, where all the projects or at least a subset of them are related. A multiple linear regression model, which aims to improve the learning of a model for P^i by using the knowledge contained in the m projects, can be represented as the following: $Y^i = X^i W^i + B^i, i = 1, \cdots, m$, which can build many liner regression models simultaneously.*

In this definition, $Y^i = (y_1^i, \cdots, y_{n^i}^i) \in \mathbb{R}^{n^i}$ represents the responds for the i-th project, regressed on the training instance matrix $X^i = (x_1^i, \cdots, x_{n^i}^i) \in \mathbb{R}^{n^i \times F^i}$, where n^i and F^i represent the number of training instances of the i-th project and the feature space of i-th project respectively. $B^i \in \mathbb{R}^{n^i}$ is a bias vector. There are m tasks in total. For convenience, we transform these quantities into matrices. That is, $Y \in \mathbb{R}^{n^i \times m}$ represents the responses of the regression model, $W \in \mathbb{R}^{F^i \times m}$ represents the regression parameters and $B \in \mathbb{R}^{n^i \times m}$ represents the noise.

According to this definition, it is not hard to find that the parameter W is very important since it contains the relatedness among different projects. For convenience, we represent the parameters W as a matrix, where each column corresponds to a project, and each row corresponds to a feature. Meanwhile, not only different projects have the common characteristics, but also they have the personalized characteristics. Therefore, both the shared information and non-shared information are included in this parameter W. It seems to be that any one structure might not fully capture all of those information, but a decomposition of the structure might [18]. Therefore the matrix W should be decomposed into two component matrices P and Q, i.e., $W = P + Q$. In particular, some rows of W would have many non-zero entries, which are corresponding to features shared by several projects ("shared information"). We use a row-sparse [18, 19] matrix P to represent such shared information, where each row is either all zero or mostly non-zero, and the number of non-zero rows is small. Some rows of W would be project-sensitive, since they correspond to those features, which are relevant for some projects but not all ("non-shared information"). We use an elementwise sparse matrix Q to represent such non-shared information. It is obvious that some rows of W would have all zero entries, since they correspond to those features that are not relevant to any project.

The proposed method MPR uses the multiple linear model [18] to explicitly model the relatedness among different projects. MPR estimates a sum of the two parameter matrices P and Q with different regularizations for each to encourage row-sparsity in P and elementwise sparsity in Q, therefore MPR can effectively capture shared and non-shared information among related projects.

3.2 The details of MPR Approach

Previous methods can build one regression model for one target project at a time [11, 13, 14], while MPR can build multiple regression models for multiple target projects at a time. Fig. 1 presents the overall framework of MPR. In particular, some data preprocessing operations are performed on these projects. For example, we normalize the numerical features for all these projects. The number of related projects to be learnt equals to the number of models outputted by MPR. Relatedness of different projects is a core concept in multiple linear regression learning. In this paper, we simply consider two type of relatedness. The one is whether these project are developed by the same organization. The other is whether these project have the same feature space.

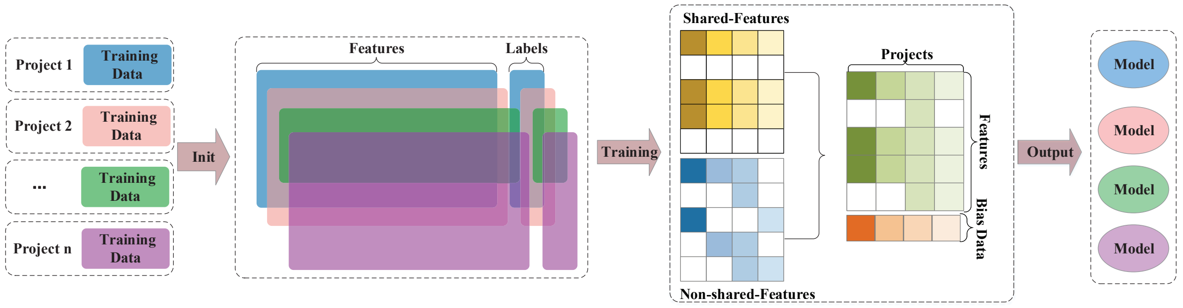

Figure 1: The Framework of MPR

The process of MPR can be described in Algorithm 1. In particular, MPR firstly needs to initialize the start points. In our implementation, we simply initialize the start points with a zero matrix (Line 1). In fact, the starting points can be initialized to a guess value computed from data or a specified point. MPR defines some variables which control the process of searching for optimal value for S, NS and B (Line 2). Then, MPR goes into a loop block with $maxIter$ iteration times (Lines 3-24). In this loop, MPR updates $Shared$,$NonShared$ and $Bias$ with the search factor α (Lines 4-5). Later, it computes performance value and gradients of the current search points (Line 6). Next, MPR goes further into a inner loop in which it can only be terminated when a specific terminating condition is satisfied (Lines 7-18). In this inner loop, the Armijo Goldstein line search scheme [20] is used to compute the optimal shared and non-shared information structure together with the optimal bias data (Lines 8-17). After reaching the termination of the inner loop, it stores previous search points and does preparation for the next iteration. If the difference between the latest two performance meets the requirements of maximum tolerance, MPR can jump out of the outer loop block and return in advance (Lines 19-21). If not, MPR will go to the next iteration with the updated change factor (i.e., $t^{'}$,t) (Lines 22-23). At last, it returns the best value for S, NS and B after given iterations (Line 25).

4 Experiment Design and Result Analysis

4.1 Experimental Subjects

In our empirical studies, 30 projects from PROMISE are used to verify the performances of MPR. These projects can be downloaded from PROMISE and they are widely used in previous empirical studies [11, 12, 15, 16, 21, 22]. The characteristic of these projects are shown in Table 1, which includes project name, number of modules, number (percentage) of defective modules and the maximum defects contained in the modules. Each project use 20 metrics (i.e., features) to measure extracted modules, which are designed based on the code complexity and the characteristic of object oriented program. Notice that the granularity of extracted program modules is set to class.

[0]http://openscience.us/repo/

Table 1: Statistic of Experimental Subjects

Project (Versions)	# Modules	# Defective Modules	% Defective Modules	Max
ant(1.3,1.4,1.5,1.6,1.7)	125-745	33-338	10.92%-26.21%	(3,3,2,10,10)
camel(1.0,1.2,1.4,1.6)	339-965	14-522	3.83%-35.53%	(2,28,17,28)
ivy(1.1,1.4,2.0)	111-352	18-233	6.64%-56.76%	(36,3,3)
jedit(3.2,4.0.,4.1,4.2)	272-367	106-382	13.08%-33.09%	(45,23,17,10)
synapse(1.0,1.1,1.2)	157-256	21-145	10.19%-33.59%	(4,7,9)
xalan(2.4,2.5,2.6)	723-885	156-625	15.21%-48.19%	(7,9,9)
xerces(1.2,1.3)	440-453	115-193	15.23%-16.14%	(4,30)
prop(1,2,3,4,5,6)	660-23014	79-5493	9.6%-15.3%	(37,27,11,22,19,4)

4.2 Performance Measures

To evaluate the performance of MPR for SDNP, average absolute error (AAE) and average relative error (ARE) performance measures are used.

AAE measures the average magnitude of the errors in a set of predictions. It shows the difference between the predicted value and the actual value and is defined as:

$$AAE = \frac{1}{n}\sum_{i=1}^{n}\left|\bar{Y_i} - Y_i\right| \qquad (1)$$

ARE calculates how large the absolute error is compared with the total size of the object measured and is defined as:

$$ARE = (1/n)\sum_{i=1}^{n}\left|\bar{Y_i} - Y_i\right|/(Y_i + 1) \qquad (2)$$

In Formulation 1 and Formulation 2, $\bar{Y_i}$ is the predicted number of defects for i-th software module, Y_i is the actual number of defects for this module and n represents the number of modules. For ARE, sometimes Y_i may be zero. To mitigate this issue, we add the actual value of Y_i to one at the denominator to make the definition always well-defined [10]. Notice that the smaller the value of AAE and ARE, the better performance of the constructed regression model [23].

4.3 Experimental Setup

To evaluate the performance of MPR, we consider 6 baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). To

Algorithm 1 Multiple Projects Regression (MPR)

```
Input:
    $X^{(n\times F)\times m}$: all the data of related projects;
    $Y^{(n\times 1)\times m}$: the corresponding responses of all related projects;
    $maxIter$: the number of maximum iteration;
    $maxTol$: the tolerance of the performance between two iterations;
Output:
    $Shared^{F\times m}(S)$: all shared information among projects;
    $NonShared^{F\times m}(NS)$: non-shared information for each project;
    $Bias^{m\times 1}(B)$: the bias data for regression;
 1: Initial the start points: filling $Shared$, $NonShared$, $Bias$ with a zero matrix;
 2: Let $t$=1, $t^{'}$=$iter$=0, $\gamma$=1, $\gamma_{inc}$=2, $\alpha$=Initial search factor, which controls the search for optimal value of $S, NS, B$;
 3: while ($iter < maxIter$) do
 4:     Set $\alpha = (t^{'} - 1)/t$;
 5:     Find next search point for $S$, $NS$, $B$ with $\alpha$, and store them into $S_{tmp}, NS_{tmp}, B_{tmp}$;
 6:     Compute the whole loss value and gradients of the current search points with $S_{tmp}$, $NS_{tmp}$ and $B_{tmp}$ , and return $gW_{tmp}, gC_{tmp}$ and $L_{tmp}$;
 7:     while (true) do
 8:         $S_t = proximalL_{1,\infty}norm(S_{tmp} - \frac{gW_{tmp}}{\gamma}, \frac{BRP.\alpha}{\gamma})$;
 9:         $NS_t = proximalL_{1,1}norm(NS_{tmp} - \frac{gW_{tmp}}{\gamma}, \frac{BRP.\beta}{\gamma})$;
10:         $B_t = B_{tmp} - gC_{tmp}/\gamma$;
11:         Evaluate performance and save it in list of $funcVal$;
12:         Use the Frobenius norm operation on the delta of $\{S_t - S_{tmp}, NS_t - NS_{tmp}, B_t - B_{tmp}\}$, respectively, then sum them up;
13:         if (reach termination conditions) then
14:             break;
15:         else
16:             $\gamma = \gamma \times \gamma_{inc}$;
17:         end if
18:     end while
19:     if $(abs(funcVal(end) - funcVal(end - 1)) \leqslant maxTol \times funcVal(end - 1))$ then
20:         break;
21:     end if
22:     $iter$ ++;
23:     Update $t^{'}$ and $t$, i.e., $t^{'} = t, t = 0.5 \times (1 + \sqrt{1 + 4 \times t^2})$;
24: end while
25: return $S, NS, B$
```

avoid the errors in model implementation, we use these regression models implemented by scikit-learn library, which is a popular machine learning toolkit written by python language. Notice our experiments use the default hyper-parameter settings for different regression models. That is, we do not perform hyper-parameter optimization for each regression model. 6 regression models are briefly described as follows:

Linear Regression (LR) is always used to solve the least squares function of the linear relationship between one or multiple independent variables and one dependent variable.

Nearest Neighbors Regression (NNR) 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. Then, the weight is set proportional to the inverse of the distance between the instance and its neighbors.

Support Vector Regression (SVR) is the extended from the 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.

Decision Tree Regression (DTR) learns simple decision rules to approximate the curve of a given training data set, and then predicts the target variable.

Bayesian Ridge Regression (BRR) 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.

Gradient Boosting Regression (GBR) is in the form of an ensemble of weak prediction models. Several base estimators are combined with a given learning algorithm to improve the prediction accuracy over a single estimator.

For MPR, we train and test the regression model on all projects simultaneously. Besides, 10-fold cross validation (CV) are used in our experiments. In particular, we split the original dataset into 10 folds equally. Then, the 9 folds are treated as the training data, while the remaining one fold is treated as the test data. For those baselines, we also use 10-fold CV to train and test regression models on all each project independently. To overcome possible bias in the data split process, we perform 10-fold CV for 10 times and report the average performance values.

To check the significance of performance comparison, we conduct Wilcoxon signed-rank test [24], which is a non-parametric statistical hypothesis test. For all the tests, the null hypotheses are that there is no difference between the trained predictors, and the significance level α is set to 0.05. If p-value is smaller than 0.05, we reject the null hypotheses, otherwise we accept the null hypotheses.

4.4 Result Analysis

RQ: *How effective is our proposed approach MPR for SDNP? How much improvement can it achieve over the baselines?*

Motivation. Our main goal is to propose an approach to improve the performance of model for addressing regression issues by using the knowledge contained in related projects. However, to show the feasibility of this approach, we must analyze how effective it is in its prediction performance and whether it can perform better than baselines. The answer for this RQ would shed light on how much our approach advances the state of the art of SDNP.

Approach. To answer this RQ, we compare MPR with LR, NNR, SVR, DTR, BRR and GBR to investigate whether MPR has competitiveness over these baselines. AAE and ARE are used to evaluate the performance of these approaches. In addition, we run all these approaches 100 times independently.

Results. Table 2 and Table 3 show the comparison results among MPR and 6 state-of-the-art regression approaches (i.e., LR, NNR, SVR, DTR, BRR and GBR) based on AAE and ARE respectively. In both of these two tables, the first column presents the names of projects. Here we assume that all projects have a certain relatedness, since they consider the same features. The remaining 7 columns list the performance values of seven approaches in terms of different performance measures (i.e., AAE and ARE). In each row, the best results are in bold. Notice that all these results of different methods are calculated by 100 independent runs.

Table 2: Comparison of MPR and Baseline Methods on PROMISE Dataset in terms of AAE

Project	MPR	LR	NNR	SVR	DTR	BRR	GBR
ant-1.3	**0.32**	0.43	0.36	0.44	0.46	0.38	0.45
ant-1.4	**0.26**	0.40	0.38	0.42	0.42	0.41	0.41
ant-1.5	0.29	0.22	0.19	0.26	0.18	0.21	**0.18**
ant-1.6	**0.28**	0.58	0.50	0.68	0.55	0.55	0.51
ant-1.7	**0.20**	0.49	0.48	0.61	0.58	0.48	0.50
camel-1.0	0.16	0.09	**0.07**	0.16	0.09	0.08	0.09
camel-1.2	**0.15**	1.06	1.05	0.96	1.08	1.06	1.01
camel-1.4	**0.14**	0.59	0.55	0.53	0.63	0.57	0.55
camel-1.6	**0.14**	0.79	0.79	0.67	0.83	0.77	0.73
ivy-1.1	**0.28**	1.68	1.65	1.93	1.96	1.39	1.50
ivy-1.4	0.24	0.15	0.12	0.21	**0.11**	0.13	0.13
ivy-2.0	0.24	0.26	**0.23**	0.31	0.27	0.23	0.23
jedit-3.2	**0.27**	1.54	1.56	1.54	1.82	1.46	1.60
jedit-4.0	**0.28**	0.83	0.88	0.90	1.02	0.81	0.87
jedit-4.1	**0.29**	0.70	0.77	0.86	0.83	0.69	0.78
jedit-4.2	**0.26**	0.38	0.41	0.45	0.43	0.39	0.36
synapse-1.0	**0.19**	0.29	0.21	0.28	0.23	0.24	0.22
synapse-1.1	**0.19**	0.57	0.53	0.59	0.55	0.53	0.50
synapse-1.2	**0.19**	0.65	0.60	0.67	0.72	0.59	0.64
xalan-2.4	**0.05**	0.32	0.30	0.37	0.33	0.31	0.30
xalan-2.5	**0.04**	0.63	0.59	0.61	0.64	0.64	0.59
xalan-2.6	**0.06**	0.62	0.56	0.62	0.58	0.64	0.53
xerces-1.2	**0.07**	0.43	0.38	0.38	0.33	0.43	0.35
xerces-1.3	**0.06**	0.77	0.54	0.55	0.49	0.74	0.46
prop-1	**0.18**	0.45	0.39	0.39	0.43	0.45	0.42
prop-2	**0.24**	0.29	0.25	0.27	0.24	0.29	0.28
prop-3	**0.23**	0.26	0.25	0.25	0.26	0.26	0.26
prop-4	0.44	0.26	0.23	0.26	**0.23**	0.25	0.24
prop-5	**0.21**	0.35	0.32	0.34	0.35	0.35	0.34
prop-6	0.18	0.21	0.19	0.22	**0.17**	0.20	0.19

Table 3: Comparison of MPR and Baseline Methods on PROMISE Dataset in terms of ARE

Projects	MPR	LR	NNR	SVR	DTR	BRR	GBR
ant-1.3	0.28	0.31	**0.23**	0.29	0.31	0.26	0.32
ant-1.4	**0.24**	0.30	0.27	0.31	0.30	0.29	0.31
ant-1.5	0.27	0.18	0.14	0.21	**0.13**	0.17	0.13
ant-1.6	**0.23**	0.38	0.30	0.38	0.31	0.36	0.31
ant-1.7	**0.15**	0.32	0.28	0.34	0.34	0.31	0.30
camel-1.0	0.16	0.07	**0.05**	0.14	0.06	0.06	0.07
camel-1.2	**0.11**	0.64	0.60	0.45	0.61	0.64	0.60
camel-1.4	**0.11**	0.37	0.32	0.28	0.39	0.35	0.34
camel-1.6	**0.11**	0.49	0.46	0.31	0.47	0.46	0.43
ivy-1.1	**0.13**	0.74	0.61	0.60	0.70	0.64	0.61
ivy-1.4	0.21	0.10	0.08	0.17	**0.08**	0.09	0.10
ivy-2.0	0.21	0.19	**0.16**	0.22	0.20	0.16	0.16
jedit-3.2	**0.18**	0.97	0.75	0.48	0.83	0.91	0.81
jedit-4.0	**0.21**	0.52	0.47	0.38	0.58	0.52	0.51
jedit-4.1	**0.22**	0.42	0.41	0.39	0.42	0.41	0.44
jedit-4.2	**0.22**	0.26	0.26	0.26	0.27	0.27	0.23
synapse-1.0	**0.08**	0.23	0.15	0.21	0.17	0.18	0.15
synapse-1.1	**0.17**	0.39	0.32	0.36	0.37	0.36	0.33
synapse-1.2	**0.10**	0.41	0.35	0.40	0.42	0.37	0.39
xalan-2.4	**0.05**	0.22	0.20	0.26	0.22	0.21	0.20
xalan-2.5	**0.04**	0.42	0.38	0.38	0.39	0.43	0.39
xalan-2.6	**0.05**	0.40	0.34	0.37	0.34	0.41	0.33
xerces-1.2	**0.06**	0.30	0.25	0.25	0.22	0.30	0.24
xerces-1.3	**0.05**	0.52	0.28	0.26	0.25	0.50	0.25
prop-1	**0.16**	0.29	0.23	0.21	0.25	0.29	0.26
prop-2	0.23	0.20	0.17	0.17	**0.15**	0.20	0.18
prop-3	0.18	0.18	0.17	**0.17**	0.17	0.18	0.18
prop-4	0.25	0.18	0.15	0.17	**0.14**	0.17	0.15
prop-5	**0.18**	0.24	0.22	0.21	0.24	0.24	0.23
prop-6	**0.12**	0.15	0.14	0.17	0.13	0.15	0.14

Considering the performance of AAE in Table 2, MPR can achieve better performance in most cases. In particular, MPR achieves 4/5 wins, 3/4 wins, 1/3 wins, 4/4 wins, 3/3 wins, 3/3 wins, 2/2 wins and 4/6 wins on ant, camel, ivy, jedit, synapse, xalan, xerces and prop, respectively. MPR has dominate performance in jedit, synapse, xalan and xerces. However, MPR achieves pool performance (e.g., 0.44 on prop-4). Besides, DTR achieves good performance in three projects (i.e., 0.11 on ivy-1.4, 0.23 on prop-4 and 0.17 on prop-6). Considering the performance of ARE in Table 3, MPR can achieve better performance in most cases. In particular, MPR achieves 3/5 wins, 3/4 wins, 1/3 wins, 4/4 wins, 3/3 wins, 3/3 wins, 2/2 wins and 3/6 wins on ant, camel, ivy, jedit, synapse, xalan, xerces and prop, respectively. MPR also achieves dominate performance in jedit synapse, xalan, xerces and prop. However, MPR obtains pool performance (e.g., 0.27 on ant-1.5). Besides, DTR achieves good performance in four projects (i.e., 0.13 on ant-1.5, 0.08 on ivy-1.4, 0.15 on prop-2 and 0.14 on prop-4).

To check whether the performance differences among MPR and six baseline approaches are significant, we conduct the Wilcoxon signed-rank test. Table 4 shows the results in detail. Considering thirty projects from PROMISE, the p-values of "MPR vs LR", "MPR vs NNR", "MPR vs SVR", "MPR vs DTR", "MPR vs BRR" and "MPR vs GBR" are all less than 0.05, which indicates that MPR performs statistically better than the baseline methods.

Conclusion: By utilizing the relatedness among projects,

Table 4: p-Value of the Wilcoxon Signed-Rank Test Among MPR and Baseline Methods.

Performance Measure	MPR vs LR	MPR vs NNR	MPR vs SVR	MPR vs DTR	MPR vs BRR	MPR vs GBR
AAE	7.76E-07	1.37E-05	2.08E-07	1.08E-05	3.06E-06	8.19E-06
ARE	5.75E-06	2.65E-04	6.28E-06	9.99E-05	2.48E-05	1.23E-04

MPR can achieve better and stable performance for SDNP in most projects.

5 Conclusion and Future Work

In this paper, we propose a novel approach MPR, which mainly based on multiple linear regression. To show the effectiveness of MPR, we use two widely used regression performance measures (i.e., AAE and ARE) to assess the capability of MPR for SDNP on 30 projects. The results show the competitiveness of MPR when compared with six state-of-the-art baselines.

Acknowledge

Qiguo Huang and Chao Ni have contributed equally for this work. This work was supported in part by National Natural Science Foundation of China (Grant Nos.61373012, 61202006, 91218302, 61321491) and China Scholarship Council (Grant No. 201806190172), which supports overseas studies for Chao Ni to Monash University in Australia.

References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[2] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *Journal of Computer Science and Technology*, vol. 32, no. 6, pp. 1090–1107, 2017.

[3] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019.

[4] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in *Proceedings of International Conference on Mining Software Repositories*, 2017, pp. 135–145.

[5] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2002.

[6] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.

[7] L. Yu, "Using negative binomial regression analysis to predict software faults: A study of apache ant," *International Journal of Information Technology & Computer Science*, vol. 4, no. 8, pp. 63–70, 2012.

[8] J. Wang and H. Zhang, "Predicting defect numbers based on defect state transition models," in *Proceedings of Acm-Ieee International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 191–200.

[9] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, "Identification of defect-prone classes in telecommunication software systems using design metrics," *Information Sciences*, vol. 176, no. 24, pp. 3711–3734, 2006.

[10] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.

[11] M. Chen and Y. Ma, "An empirical study on predicting defect numbers," in *The International Conference on Software Engineering and Knowledge Engineering*, 2015.

[12] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, 2017, pp. 78–89.

[13] S. Kumar and S. Kumar, *A Decision Tree Regression based Approach for the Number of Software Faults Prediction*. ACM Sigsoft Software Engineering Notes, 2016, vol. 40, no. 1.

[14] S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Computing*, vol. 21, no. 24, pp. 7417–7434, 2017.

[15] ——, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems," *Knowledge-Based Systems*, vol. 119, pp. 232–256, 2017.

[16] ——, *Towards an ensemble based system for predicting the number of software faults*. Pergamon Press, Inc., 2017, vol. 82, no. 1.

[17] Y. Zhang and Q. Yang, "A survey on multi-task learning," *CoRR*, vol. abs/1707.08114, 2017.

[18] A. Jalali, P. D. Ravikumar, S. Sanghavi, and R. Chao, "A dirty model for multi-task learning," in *Proceedings of Neural Information Processing Systems Conference*, 2010, pp. 964–972.

[19] K. Lounici, M. Pontil, A. B. Tsybakov, and S. V. D. Geer, "Taking advantage of sparsity in multi-task learning," in *Proceeding of The Conference on Learning Theory*, 2009.

[20] V. Torczon, "On the convergence of pattern search algorithms," *SIAM Journal on optimization*, vol. 7, no. 1, pp. 1–25, 1997.

[21] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang, "File-level defect prediction: Unsupervised vs. supervised models," in *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2017, pp. 344–353.

[22] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information & Software Technology*, vol. 59, no. C, pp. 170–190, 2015.

[23] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software engineering metrics and models*. Benjamin/Cummings Publishing Company, Inc, 1986.

[24] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.