Revisting the Impact of Regression Models for Predicting the Number of Defects

Man Wu[1,2], Sizhe Ye[4], Chunhua Li[1*], Ziyi Ma[3], Zhongwang Fu[2]
[1]Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China
[2]School of Computer Science and Information Engineering, Hubei University, Wuhan, China
[3]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[4]State Key Lab. of Software Engineering, Computer School, Wuhan University, Wuhan, China
*Corresponding author email: li.chunhua@hust.edu.cn

***Abstract*— Predicting the number of faults in software modules can be more helpful instead of predicting the modules being faulty or non-faulty. Chen et al. (SEKE 397-402, 2015) and Rathore et al. (Soft Computing 21: 7417-7434, 2017) empirically investigate the feasibility of some regression algorithms for predicting the number of defects. The experimental results showed that the decision tree regression algorithm performed best in terms of average absolute error (AAE), average relative error (ARE) and root mean square error (RMSE). However, they did not consider the imbalanced data distribution problem in defect datasets and employed improper performance measures for evaluating the regression models to evaluate the performance of models for predicting the number of defects. Hence, we revisit the impact of different regression algorithms for predicting the number of defects using Fault-Percentile-Average (FPA) as the performance measure. The experiments on 31 datasets from PROMISE repository show that the prediction performance of models for predicting the number of defects built by different regression algorithms are various, and the gradient boosting regression algorithm and the Bayesian ridge regression algorithm can achieve better performance.**

Keywords—predicting the number of defects; regression algorithm; data imbalance; Fault-Percentile-Average;

I. INTRODUCTION

Software defect prediction is one of the most important software quality assurance techniques. Based on the investigation of historical metrics, defect prediction aims to detect the defect proneness of new software modules. Therefore, defect prediction is often used to help to reasonably allocate limited development and maintenance resources [1]. So far, many efficient software defect prediction methods using statistical methods or machine learning techniques have been proposed [2-4], but they are usually confined to predicting a given software module being faulty or non-faulty by means of some binary classification techniques.

However, predicting the defect-prone of a given software module does not provide enough logistics to software testing in practice [5-6]. Some of the faulty software modules may have comparatively vast quantities of faults compared to other modules and hence require some additional maintenance resources to fix them. So, it may result in a waste of limited maintenance resources if simply predicting the defect-prone of a given software module and allocating the limited maintenance resources solely based on faulty and non-faulty information. If we are able to predict the accurate number of faults, software testers will pay particular attention to those software modules that have more number of faults, which makes testing processes more efficient in the case of limited development and maintenance resources. Thus, predicting the number of faults in software modules can be more helpful instead of predicting the modules being faulty or non-faulty [6].

A. Motivation

A number of prior studies have investigated regression models on predicting the number of faults. Some researchers [7-12] have investigated genetic programming, decision tree regression, and multilayer perceptron in the context of predicting the number of defect, and found that these models achieved good performance. Chen et al. [11] performed an empirical study on predicting the number of faults using six regression algorithms and found that the prediction model built with decision tree regression had the highest prediction accuracy in terms of precision and root mean square error (RMSE). In another similar study, Rathore et al. [12] presented an experimental study to evaluate and compare six regression algorithms for predicting the number of defects. The results found that decision tree regression, multilayer perceptron, and linear regression achieved better performance in terms of average absolute error (AAE) and average relative error (ARE), measure of completeness, and prediction at level *l* measures.

However, the software defect datasets are imbalanced. In other words, the number of defects in the majority of modules is zero, and the minority of modules have one or more defects. Using imbalanced defect data to derive a regression model and then estimate the error value of the resulting learned model can result in misleading over-optimistic estimates due to over-specialization of the learning algorithm to the imbalanced defect data [6]. Suppose that we have trained a regression model to predict the number of defects in the project Ant 1.3 (see Table I), which contains 124 instances and 33 defects. An AAE value of, say, 0.264 (=33/124) may make the regression model seem quite accurate. But, an AAE value of 0.264 may

DOI reference number: 10.18293/SEKE2018-068

not be acceptable—the regression model could predict the number of defects of all instances to be zero.

In addition, Yang et al. [13] pointed out that predicting the precise number of defects of a module is hard to do due to the lack of good quality data in practice. Actually, for those existing approaches that tried to predict explicitly the number of defects in a software module, they used these predicted numbers to rank the modules anyway, to direct the software quality assurance team in targeting the most faulty modules first [14], [15]. Actually, for software defect prediction for the ranking task, models with higher prediction accuracy (smaller AAE or ARE) might give a worse ranking. For example, assuming that there are three software modules M_1, M_2, and M_3, which have 2, 3, and 4 defects respectively, model A predicts that M_1, M_2, and M_3 have 2, 1, and 4 defects respectively, while model B predicts that M_1, M_2, and M_3 have 0, 1, and 2 defects respectively. Although model A has a better prediction accuracy (according to AAE and ARE), model B gives tester better ranking of these three software modules and can guide the assignment of testing resources correctly. Hence, Weyuker et al. [15] proposed FPA to reflect the effectiveness of the different prediction models for predicting the number of defects.

B. Our work

Considering on the issue that the existing studies [11-12] employed improper performance measures (e.g., RMSE, AAE and ARE) to evaluate the performance of models for predicting the number of defects, thus resulting in wrong conclusion, we revisit the impact of seven regression algorithms for predicting the number of defects by using FPA as the performance measure. These seven regression algorithms are Bayesian Ridge Regression (BRR), Decision Tree Regression (DTR), Gradient Boosting Regression (GBR), Linear Regression (LR), Nearest Neighbors Regression (NNR), Multilayer Perceptron Regression (MPR), and Support Vector Regression (SVR). The experimental study is performed on 9 software projects collected from the PROMISE repository. The FPA performances of the seven algorithms are analyzed by the one-way ANOVA test and the multiple comparison test. The experimental results show that the performance difference of seven regression algorithms for predicting the number of defects are statistically significant, and the model for predicting the number of defects built by Gradient Boosting Regression algorithm and Bayesian Ridge Regression algorithm achieve better performance.

C. Organization

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the seven regression algorithms for predicting the number of defects. Section 4 introduces the experiment setup. Section 5 demonstrates the experimental results. Finally, Section 6 addresses the conclusion.

II. Related Work

Many researchers have proposed various models for predicting the module being faulty or non-faulty. Support vector machine [16-17], neural networks [18], decision trees [19] and Bayesian methods [20] paved the way for classification-based methods in the flied of defect prediction. These methods used software metrics to properly predict whether a module is defect-prone or not. However, these methods are confined with predicting a given software module being faulty or non-faulty.

A number of prior studies have investigated regression models on predicting the number of software faults. Graves et al. [21] presented a generalized linear regression based method for predicting the number of defects by using various change metrics datasets collected from a large telecommunication system, and found that modules age, changes made to module and the age of the changes were significantly correlated with the defect-prone. Afzal et al. [8] used genetic programming for modeling software reliability growth to help in deciding project release time and managing project resources. Rathore et al. [7] proposed an approach to predict the number of faults in the software system and developed defect prediction model using neural network and genetic programming. In the subsequent study [9], they proposed an approach to predict the number of faults in the given software system using the Genetic Programming (GP). The results showed that GP based models could produce the significant results for the number of faults prediction. In another paper [10], they explored the capability of decision tree regression (DTR) for the number of faults prediction in two different scenarios, intra-release prediction and inter-releases prediction for the given software system. The experimental results indicated that intra-project prediction produced better accuracy.

Chen et al. [11] performed an empirical study on predicting the number of faults using six regression algorithms and found that the prediction model built with decision tree regression had the highest prediction accuracy in terms of precision and RMSE in most cases. In another similar study, Rathore et al. [12] presented an experimental study to evaluate and compare six regression algorithms for the number of faults prediction. The results found that decision tree regression, genetic programming, multilayer perceptron, and linear regression achieved better performance in terms of AAE, ARE, measure of completeness, and prediction at level *l* measures in many cases. However, the two empirical studies employed improper performance measures (e.g., RMSE, AAE and ARE) to evaluate the performance of models for predicting the number of defects, thus resulting in wrong conclusion.

III. Regression Algorithms

Figure 1 shows a typical process of predicting the number of defects, which can be divided into four stages. In the first stage, different from labeling a module defective or not for predicting a given software module being faulty or non-faulty, the number of defects in software modules is extracted. In the second stage, the features for each software module are extracted. Common features include complexity metrics, changes, or structural dependencies. Therefore, we can construct a training dataset for predicting the number of defects. In the third stage, we can build a model for predicting the number of defects with the help of some regression algorithms. In the last stage, after extracting the same features from a new

software module, we can use the model to predict the number of defects in the module.

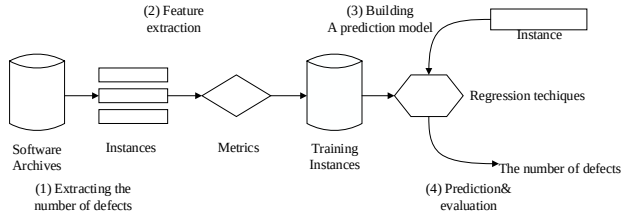

Fig. 1. The flow chart of predicting the number of defects

In this paper, we select seven regression algorithms, i.e., Bayesian Ridge Regression (BRR), Decision Tree Regression (DTR), Gradient Boosting Regression (GBR), Linear Regression (LR), Nearest Neighbors Regression (NNR), Multilayer Perceptron Regression (MPR), and Support Vector Regression (SVR). We implement these regression algorithms 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 introductions of the seven regression algorithms are described as follows:

(1) Bayesian Ridge Regression (BRR). It estimates a probabilistic model of the regression problem by introducing uninformative priors over the hyper parameters of the model. The prior for the parameter ω is given by a spherical Gaussian:

$$p(\omega|\lambda) = N(\omega|0, \lambda^{-1} I_p) \tag{1}$$

The priors over α and λ are chosen to be gamma distributions, the conjugate prior for the precision of the Gaussian. The resulting model is called Bayesian Ridge Regression, and is similar to the classical Ridge. The parameters ω, α and λ are estimated jointly during the fit of the model. The remaining hyperparameters are the parameters of the gamma priors over α and λ. These are usually chosen to be non-informative. The parameters are estimated by maximizing the marginal log likelihood.

(2) Decision Tree Regression (DTR). It predicts the value of a target variable by learning simple decision trees inferred from the data features. A decision tree is built top-down from a root node and uses a splitting criterion to partition the data into subsets that contain instances with similar values. The attribute which maximizes the expected error reduction is chosen as the root node. The process is run recursively on the non-leaf branches, until all data is processed.

(3) Gradient Boosting Regression (GBR). It produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do. It allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

(4) Linear Regression (LR). It is a statistical approach for modeling the linear relationship between a dependent variable *y* and one or more independent variables. A linear regression model can be described according to Eq.(2).

$$Y=b_0+b_1 x_1+b_2 x_2+\ldots+ b_n x_n \tag{2}$$

where Y is the dependent variable, x_1, x_2,…,and x_n are the independent variables, b_1,b_2,…,and b_n are the regression coefficients of the independent variables and b_0 is the error term.

(5) Nearest Neighbors Regression (NNR). It is based on the *k*-nearest neighbors algorithm, and the regression value of an instance is computed based the mean of the labels of its nearest neighbors. The basic nearest neighbors regression uses uniform weights: that is, each point in the local neighborhood contributes uniformly to the classification of a query point. Under some circumstances, it can be advantageous to weight points such that nearby points contribute more to the regression than faraway points.

(6) Multilayer Perceptron Regression (MPR). It consists of a series of processing elements interconnected through the connection weights in the form of layers. A multilayer perceptron regression model can be described according to Eq. (3) and Eq. (4).

$$net_k = w_{1k}x_1 + w_{2k}x_2 + \cdots + w_{nk}x_n + b_k \tag{3}$$

$$O_k = f(net_k) \tag{4}$$

where O_k is the dependent variable, x_1, x_2,…,x_n are the independent variables, w_{1k},w_{2k},…,w_{nk} are the weights associated with each input layer, and function $f(.)$ is a activation function.

(7) Support Vector Regression (SVR). It uses the same principles as the SVM for classification, with only a few minor differences. The main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

Training the original SVR means solving:

$$\text{minimize} \frac{1}{2}\|w\|^2 \tag{5}$$

$$\text{subject to} \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases} \tag{6}$$

where x_i is a training sample with target value y_i. The inner product plus intercept $\langle w, x_i \rangle + b$ is the prediction for that sample, and ε is a free parameter that serves as a threshold: all predictions have to be within an ε range of the true predictions. Slack variables are usually added into the above to allow for errors and to allow approximation in the case the above problem is infeasible.

IV. Experiment Setup

A. Data set

In this experiment, we employ 31 available and commonly used datasets which can be obtained from PROMISE [22]. Table I tabulates the details about the datasets. The 31 datasets have the same 20 features. For the complete details of the software features, please refer to [14].

B. Performance measures

Considering k modules listed in increasing order of predicted defect number as f_1, f_2, f_3,…, f_k, and assuming that n_i is the actual defect number in the module i, $n=n_1+n_2+\ldots+n_k$ is

the total number of defects, and the top predicted modules should have $\sum_{i=k-m+1}^{k} n_i$ defects. The proportion of the actual defects in the top *m* predicted modules to the whole defects is

$$\frac{1}{n}\sum_{i=k-m+1}^{k} n_i. \quad (7)$$

Then the FPA is define as

$$\frac{1}{k}\sum_{m=1}^{k}\frac{1}{n}\sum_{i=k-m+1}^{k} n_i. \quad (8)$$

FPA is actually the average of the proportions of actual defects in the top modules to the whole defects. A higher FPA means a better ranking, where the modules with most defects come first.

TABLE I. DETAILS OF EXPERIMENT DATASET

Project	Release	#Instance	#Defects	%Defects	Max	Avg
Ant	1.3	125	33	16	3	1.65
	1.4	178	47	22.5	3	1.18
	1.5	293	35	10.9	2	1.09
	1.6	351	184	26.2	10	2.00
	1.7	745	338	22.3	10	2.04
Camel	1.0	339	14	3.8	2	1.08
	1.2	608	522	35.5	28	2.42
	1.4	872	335	16.6	17	2.31
	1.6	965	500	19.5	28	2.66
Forest	0.7	29	15	17.2	8	3.0
	0.8	32	6	6.2	4	3.0
Jedit	3.2	272	382	33.1	45	4.24
	4.0	306	226	24.5	23	3.01
	4.1	312	217	25.3	17	2.75
	4.2	367	106	13.1	10	2.21
	4.3	492	12	2.2	2	1.09
Log4j	1.0	135	61	25.2	9	1.79
	1.1	109	86	33.9	9	2.32
Prop	1	18471	5293	14.8	37	2.01
	2	23014	4096	10.6	27	1.68
	3	10274	1640	11.5	11	1.39
	4	8718	1362	9.6	22	1.62
	5	8516	1930	15.3	19	1.49
	6	660	79	10	4	1.2
Synaps	1.0	157	21	10.2	4	1.31
	1.1	222	99	27	7	1.65
	1.2	256	145	33.6	9	1.69
Xalan	2.4	723	156	15.2	7	1.42
	2.5	803	531	48.2	9	1.37
Xerces	1.3	453	193	15.2	30	2.8
	1.4	588	1596	74.3	62	3.65

C. Experimental procdure

Tantithamthavorn et al. [23] investigated some model validation techniques in the domain of defect prediction and recommended out-of-sample bootstrap validation. Therefore, we employ out-of-sample bootstrap validation to perform the experiment. The out-of-sample bootstrap process is made up of two steps:

(Step 1) A bootstrap sample of size *N* is randomly drawn with replacement from an original dataset that is also of size *N*.

(Step 2) A model is trained using the bootstrap sample and tested using the rows of the original sample that do not appear in the bootstrap sample.

We repeat the out-of-sample bootstrap 10 times and the average out-of-sample performance is reported as the performance estimate.

D. Reasearch questions

In order to provide the guidance for the choice of various regression algorithms for predicting the number of defects, we address the following two research questions.

RQ1：Which regression algorithm can achieve the best performance for predicting the number of defects among the seven algorithms?

RQ2：Are the performances of models for predicting the number of defects built with different regression algorithms different?

V. EXPERIMENT RESULTS

A. Results for RQ1

The results (in terms of FPA) of the seven regression algorithms for each dataset are reported in Table II. Experimental results show that BRR and GBR produce the higher FPA values on most of the datasets compared to other considered regression algorithms for predicting the number of defects. GBR outperforms other regression algorithms on 9 datasets and achieves the highest average FPA value (0.656). LR, NNR and DTR are the third, fourth and fifth best defect prediction algorithms in terms of the average FPA value, respectively, while MPR and SVR produce relatively lower FPA values compared to other regression algorithms.

TABLE II. FPA VALUES ON 31 DATASETS USING THE SEVEN REGRESSION ALGORITHMS

Release	BRR	DTR	GBR	NNR	LR	MPR	SVR
Ant-1.3	0.708	0.698	0.675	0.639	0.618	**0.733**	0.680
Ant-1.4	0.551	0.610	**0.673**	0.590	0.601	0.479	0.551
Ant-1.5	**0.742**	0.543	0.580	0.656	0.684	0.366	0.440
Ant-1.6	0.721	0.695	**0.768**	0.721	0.715	0.207	0.449
Ant-1.7	**0.811**	0.711	0.798	0.771	0.809	0.385	0.475
Camel-1.0	**0.608**	0.603	0.600	0.597	**0.608**	0.351	**0.608**
Camel-1.2	**0.645**	0.571	0.602	0.574	0.642	0.533	0.570
Camel-1.4	0.691	0.620	0.714	0.652	**0.736**	0.615	0.453
Camel-1.6	0.704	0.635	0.693	0.486	**0.728**	0.660	0.513
Forrest-0.7	0.801	0.468	0.718	**0.827**	0.801	0.641	0.494
Forrest-0.8	0.700	0.800	**0.900**	0.700	0.700	0.500	0.800
Jedit-3.2	0.846	0.562	**0.853**	0.789	0.835	0.383	0.503
Jedit-4.0	0.787	0.710	**0.803**	0.750	0.795	0.723	0.608
Jedit-4.1	**0.738**	0.619	0.689	0.628	**0.738**	0.469	0.463
Jedit-4.2	0.787	0.690	0.787	0.743	**0.795**	0.633	0.716
Jedit-4.3	0.286	0.267	0.267	0.286	0.286	**0.955**	0.286
Log4j-1.0	0.456	0.715	0.703	0.784	0.457	**0.891**	0.695
Log4j-1.1	**0.735**	0.659	0.640	0.642	0.472	0.722	0.547
Prop-1.0	0.588	0.539	**0.595**	0.555	0.584	0.498	0.392
Prop-2.0	0.542	0.506	0.533	0.497	**0.543**	0.461	0.447
Prop-3.0	0.488	0.451	0.472	0.466	0.486	**0.539**	0.426
Prop-4.0	0.631	0.566	0.618	0.597	**0.632**	0.609	0.508
Prop-5.0	0.552	0.444	0.540	0.508	0.550	**0.561**	0.430
Prop-6.0	0.588	0.479	0.547	0.567	0.584	**0.607**	0.501
Synaps-1.0	0.282	0.517	**0.562**	0.311	0.266	0.271	0.282
Synaps-1.1	**0.755**	0.753	0.702	0.538	0.740	0.750	0.589
Synaps-1.2	0.648	0.535	0.596	**0.650**	0.623	0.428	0.567
Xalan-2.4	0.545	0.543	0.541	**0.595**	0.584	0.537	0.533
Xalan-2.5	0.631	0.597	**0.658**	0.615	0.616	0.442	0.556
Xerces1-.3	0.769	0.740	0.754	0.762	**0.804**	0.230	0.442
Xerces-1.4	0.724	0.655	**0.752**	0.680	0.720	0.689	0.632
Average	0.647	0.597	**0.656**	0.619	0.637	0.544	0.521

Figure 2 shows the box-plot diagrams of FPA measure. X-axis indicates the regression algorithms under consideration, and Y-axis indicates the values of FPA measure produced by the used regression algorithms. The different parts of box-plot show the minimum, maximum, median, first quartile, and third quartile of the samples. The line in the middle of the box shows the median of the samples. SVR and MPR produce the lowest median value, DTR, LR and NNR produce moderate median FPA values, whereas BRR and GBR produce the highest median FPA values. MPR produces the lowest minimum FPA values, DTR, LR, BRR and SVR produce moderate minimum FPA values, GBR and NNR produce the highest minimum FPA values. For maximum value, SVR produces the lowest FPA values, DTR, NNR, LR and MPR produce moderate FPA values, and BRR and GBR produce the highest FPA values. For first quartile value, SVR produces the lowest value, DTR, NNR, LR and MPR produce moderate values, and BRR and GBR produce the highest values. For the third quartile, MPR and SVR produce the lowest value, DTR and NNR produce moderate values, and BRR, GBR and LR produce the highest values.

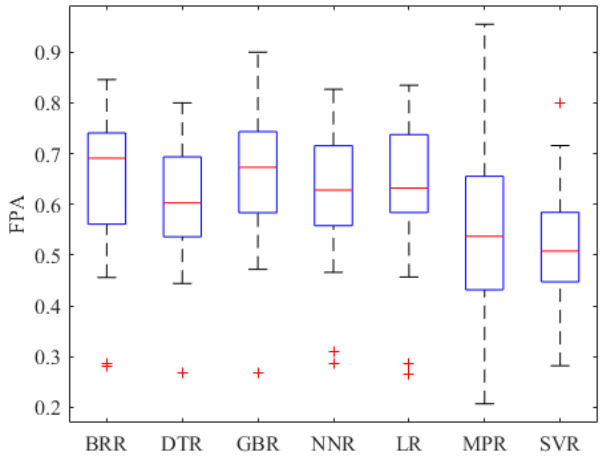

Fig. 2. Box-plot analysis for FPA measure for all the datasets

In total, Table II and Figure 2 illustrate that BRR and GBR outperform other considered regression algorithms. DTR, NNR and LR perform moderately, while MPR and SVR perform relatively poor in compared to other used regression algorithms. To answer the first question, Bayesian Ridge Regression and Gradient Boosting Regression algorithms can achieve the best performance for predicting the number of defects among the seven algorithms.

B. Results for RQ2

To answer the second question, we carry out one-way ANOVA test on the seven regression algorithms to examine if the algorithms are statistically different or not. Analysis of variance (ANOVA) is a collection of statistical models and their associated procedures used to analyze the differences among group means. Since the two-group case can be covered by a t-test, the main idea of one-way ANOVA is to compare means of two or more groups by using the F distribution. It is more conservative than the t-test and performs well in terms of comparing groups for statistical significance. The null hypothesis for the one-way ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different. Table III shows the one-way ANOVA results.

The first row of the Table III lists five parameters, i.e., sums of squares, degrees of freedom, mean square, F and p-value. Since F=4.59> 3.68, the results are significant at the 5% significance level. In addition, it is clearly that the p-value (0.0002) for this test is less than the typical cutoff 0.05. We would reject the null hypothesis, concluding that there is strong evidence that at least two regression algorithms are significantly different from each other.

TABLE III. ANOVA FOR THE DATASETS

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Approach	0.507	6	0.0845	4.59	0.0002
Error	3.865	210	0.0184		
Total	4.37	216			

In the study, we further performed the multiple comparison test using Tukey's honestly significant difference criterion. Figure 3 shows the multiple comparison result for the seven regression algorithms. The figure displays graphs with each group mean represented by a symbol (◦) and 95% confidence interval as a line around the symbol. There are two situations in the figure: two means are significantly different, if their intervals disjoint. On the contrary, two means are not significantly different, if their intervals overlap. From Figure 3, we can summarize the following points:

(1) MPR and SVR algorithms have significantly worse prediction performance than other algorithms.

(2) Although the other five algorithms show similar performance (no significant difference), BRR and GBR performs slightly better than DTR, NNR and LR.

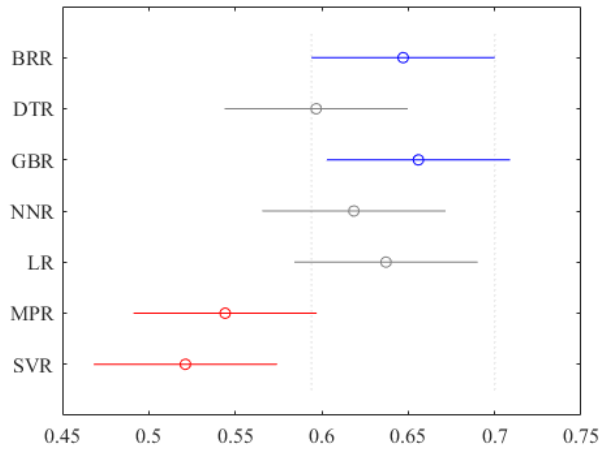

Fig. 3. Multiple comparison for seven algorithms

As seen in Table III and Figure 3, we can conclude that the models built by different regression algorithms for predicting the number of defects have different performance.

C. Threats to Validity

In this subsection, we discuss several validity threats that may have an impact on the results of our studies. (1) Although the 31 datasets in our experiment have been widely used in many software defect prediction studies, we still cannot claim that our conclusion can be generalized to other datasets. (2) We only study the seven regression algorithms without additional optimization for a given dataset. (3) We only employ FPA as the evaluation measure. Nonetheless, other evaluation measures such as cost effectiveness graph [24] can also be considered.

VI. Conclusion

The existing studies [11-12] employed improper performance measures (e.g., RMSE, AAE and ARE) to evaluate the performance of models for predicting the number of defects. Therefore, in this paper, we evaluated and compared the performance of seven regression algorithms (i.e., BRR, DTR, GBR, LR, NNR, MPR, and SVR) for predicting the number of defects in given software modules by using FPA as the performance measure. The experiments were performed on 31 datasets from the PROMISE repository. In addition, the one-way ANOVA test and the multiple comparison test are performed to assess the relative performance of the seven algorithms. The experimental results show that the performance difference of seven regression algorithms for predicting the number of defects are statistically significant, and Gradient Boosting Regression algorithm and Bayesian Ridge Regression algorithm achieve better performance for predicting the number of defects.

Acknowledgment

This work is supported by the National Key R&D Program of China (2016YFB0800402), partially supported by the National Natural Science Foundation of China under Grant No.61232004 and the Fundamental Research Funds for the Central Universities(2016YXMS020).

References

[1] F. Rahman, D. Posnett, P. Devanbu , Recalling the imprecision of cross-project defect prediction, Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering,2012, 61.

[2] X. Yu, M. Wu, Y. Jian, et al, Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning, Soft Computing, 2018: 1-12.

[3] X. Yu, J. Liu, W. Peng, et al, Improving Cross-Company Defect Prediction with Data Filtering, International Journal of Software Engineering and Knowledge Engineering, 2017, 27(09n10): 1427-1438.

[4] X. Yu, J. Liu, M. Fu, et al, A Multi-Source TrAdaBoost Approach for Cross-Company Defect Prediction, SEKE, 2016: 237-242.

[5] X. Yu, J. Liu, Z. Yang, et al, The Bayesian Network based program dependence graph and its application to fault localization, Journal of Systems and Software, 2017, 134: 44-53.

[6] Yu X, Liu J, Yang Z, et al. Learning from Imbalanced Data for Predicting the Number of Software Defects, 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2017: 78-89.

[7] Rathore S S, Kuamr S, Comparative analysis of neural network and genetic programming for number of software faults prediction, 2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), 2015: 328-332.

[8] W. Afzal, R. Torkar, R.Feldt, Prediction of fault count data using genetic programming, Multitopic Conference, 2008. INMIC 2008. IEEE International. IEEE, 2008: 349-356.

[9] S. S. Rathore, S. Kumar , Predicting number of faults in software system using genetic programming, Procedia Computer Science, 2015, 62: 303-311.

[10] S. S. Rathore, S.Kumar, A Decision Tree Regression based Approach for the Number of Software Faults Prediction, ACM SIGSOFT Software Engineering Notes, 2016, 41(1): 1-6.

[11] M. Chen, Y. Ma, An empirical study on predicting defect numbers, 28th International Conference on Software Engineering and Knowledge Engineering, 2015: 397-402.

[12] S. S. Rathore, S. Kumar, An empirical study of some software fault prediction techniques for the number of faults prediction, Soft Computing, 2016: 1-18.

[13] Yang, Xiaoxing, K. Tang, and X. Yao, A learning-to-rank approach to software defect prediction, IEEE Transactions on Reliability, 2015, 64(1): 234-246.

[14] K. Gao and T. M. Khoshgoftaar, A comprehensive empirical study of count models for software fault prediction, IEEE Transactions on Reliability, 2007, 56(2): 223-236.

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

[16] D. Gray, D. Bowes, N. Davey, et al, Using the support vector machine as a classification method for software defect prediction with static code metrics, International Conference on Engineering Applications of Neural Networks, Springer Berlin Heidelberg, 2009: 223-234.

[17] Z. Yan, X. Chen, P. Guo, Software defect prediction using fuzzy support vector regression, International Symposium on Neural Networks, Springer Berlin Heidelberg, 2010: 17-24.

[18] M. M. T. Thwin, T. S.Quah, Application of neural networks for software quality prediction using object-oriented metrics, Journal of systems and software, 2005, 76(2): 147-156.

[19] J. Wang, B. Shen, Y.Chen, Compressed C4. 5 models for software defect prediction, 2012 12th International Conference on Quality Software. IEEE, 2012: 13-16.

[20] T. Wang, W. Li, Naive bayes software defect prediction model, Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on IEEE, 2010: 1-4.

[21] T. L. Graves, A. F. Karr, J. S. Marron, et al, Predicting fault incidence using software change history, IEEE Transactions on software engineering, 2000, 26(7): 653-661.

[22] G. Boetticher, T. Menzies, T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, 2007 <http://promisedata.org/repository>.

[23] C. Tantithamthavorn, S. Mcintosh, A. E. Hassan, et al, An Empirical Comparison of Model Validation Techniques for Defect Prediction Models, IEEE Transactions on Software Engineering, 2017, 43(1):1-18.

[24] T. Jiang, L. Tan, S. Kim, Personalized defect prediction, Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. IEEE, 2013.