Information and Software Technology 146 (2022) 106847

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Predicting the precise number of software defects: Are we there yet?

Xiao Yu [a,b], Jacky Keung [c], Yan Xiao [d], Shuo Feng [e,*], Fuyang Li [a,b,*], Heng Dai [f]

[a] *School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China*
[b] *Wuhan University of Technology Chongqing Research Institute, Chongqing, China*
[c] *Department of Computer Science, City University of Hong Kong, Hong Kong, China*
[d] *School of Computing, National University of Singapore, Singapore*
[e] *Nanjing Institute of Intelligent Technology, Nanjing, China*
[f] *School of Mechanical and Electrical Engineering, Wuhan Qingchuan University, Wuhan, China*

ARTICLE INFO

Keywords:
Software Defect Prediction
Number of defects
Regression algorithms
Empirical software engineering

ABSTRACT

Context: Defect Number Prediction (DNP) models can offer more benefits than classification-based defect prediction. Recently, many researchers proposed to employ regression algorithms for DNP, and found that the algorithms achieve low Average Absolute Error (AAE) and high Pred(0.3) values. However, since the defect datasets generally contain many non-defective modules, even if a DNP model predicts the number of defects in all modules as zero, the AAE value of the model will be low and Pred(0.3) value will be high. Therefore, the good performance of the regression algorithms in terms of AAE and Pred(0.3) may be questioned due to the imbalanced distribution of the number of defects.
Objective: To revisit the impact of regression algorithms for predicting the precise number of defects.
Method: We examine the practical effects of 12 widely-used regression algorithms, two data resampling algorithm (SmoteR and ROS), and three ensemble learning algorithms (gradient boosting regression, AdaBoost.R2, and Bagging), one feature selection method (information gain) and one parameter optimization method (grid search) for predicting the precise number of defects on the 18 PROMISE datasets. We propose to evaluate the AAE and Pred(0.3) values for the modules with different numbers of defects separately.
Results: The AAE values for defective modules are very high and the Pred(0.3) values are very low, i.e., the regression algorithms are very inaccurate for predicting the precise number of defects in defective modules.
Conclusion: The problem of predicting the precise number of defects via regression algorithms is far from being solved. We recommend that software testers use regression algorithms to rank modules for testing resource allocation, rather than predict the precise number of defects to evaluate the software reliability and maintenance effort. In addition, most existing DNP studies employing the whole AAE and Pred(0.3) values of all modules as the evaluation metrics for the proposed DNP algorithms should be revisited.

1. Introduction

Software Defect Prediction (SDP) plays a very important role in predicting whether software modules are defective based on some extracted software features [1–5]. Accurate prediction results can help software testers conserve limited testing resource by focusing on those predicted defective modules [6–9] and guide the fault localization [10–13]. However, classification-based SDP that predicts the defect-proneness of the software modules is not enough in the real situation. We take an example to explain the reason in Fig. 1.

Example 1. There are 200 software modules (M_1, M_2, M_3, …, M_{200}) that need to be tested in a new software project shown in Fig. 1. Software testers want to not only know which software modules should be inspected first, but also evaluate the reliability and maintenance effort of each module. Therefore, they can first employ the historical data to construct a classification-based SDP model or a Defect Number Prediction (DNP) model, then use the two trained models to predict the defective-proneness or the number of defects.

(1) Assuming that the classification-based SDP model predicts 25% software modules are defective (e.g., M_1 and M_{200} are predicted to be defective), but software testers can only inspect few software modules (e.g., 15% modules) because of the limited testing resources. Therefore, they face down a challenge to choose which modules among the predicted defective modules to be inspected based on the classification

* Corresponding authors.
E-mail addresses: xiaoyu@whut.edu.cn (X. Yu), jacky.keung@cityu.edu.hk (J. Keung), dcsxan@nus.edu.sg (Y. Xiao), shuofeng5-c@my.cityu.edu.hk (S. Feng), fyli@whut.edu.cn (F. Li), daiheng726@163.com (H. Dai).

https://doi.org/10.1016/j.infsof.2022.106847
Received 17 June 2021; Received in revised form 3 December 2021; Accepted 11 January 2022
Available online 22 January 2022

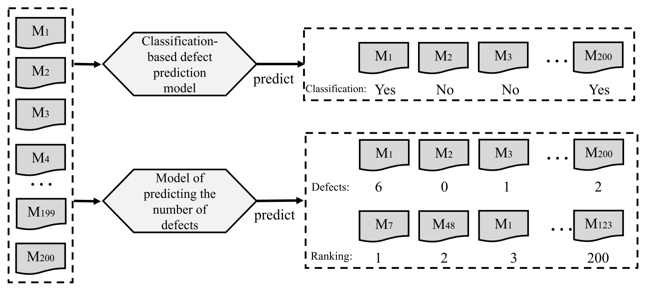

M1
M2
M3
M4
M199
M200
Classification-based defect prediction model
predict
Classification: Yes No No Yes
M200
Model of predicting the number of defects
predict
Defects: 6 0 1 2
M7
M48
M123
Ranking: 1 2 3 200

Fig. 1. The difference between the classification-based model and the DNP model.

model. But they can inspect the top-ranked 15% modules (i.e., M_7, M_{48}, M_1, and etc.) to find more defects based on the DNP model, which sorts the 200 modules based on the predicted number of defects.

(2) Generally, the software unreliability and maintenance effort is proportional to the number of defects [14,15]. Based on the prediction results of the classification model, the software unreliability and maintenance effort of M_1 and M_{200} are equal, since M_1 and M_{200} are both predicted to be defective. But the DNP model predicts that M_1 contains 6 defects, and M_{200} contains 2 defects. Therefore, M_{200} is more reliable than M_1, and the maintenance effort of M_{200} is less than that of M_1. In other words, the classification model cannot distinguish between a module with more defects and a module with less defects when evaluating the reliability and maintenance effort of each module.

To sum up, DNP models can offer more benefits of scarce testing resources allocation, reliability evaluation, and maintenance effort estimation than classification-based defect prediction in software testing process [14–17].

1.1. Motivation

Recently, many researchers proposed to employ regression algorithms for DNP, and found that the algorithms performed well. Generally, there are two purposes to predict the defect number of software modules.

(1) The DNP model is used to sort software modules and guide software testers to allocate the scarce testing resources more efficiently. In this situation, some ranking performance measures are employed to evaluate the DNP model, such as Fault Percentage Average (FPA) and Percentage of the found Bugs when inspecting the top 20% modules (PofB).

(2) The DNP model is used to evaluate the reliability and maintenance effort of each module. For example, Lyu et al. [39] and Afzal et al. [21] stated that the DNP model can assist to estimate reliability or predict future reliability by different forms of extrapolation. Zhang et al. [14,40] pointed out that the DNP model can help software developers and software users from the following aspects. First, it provides software developers with an evaluation of software reliability, which is vital to the determination of delivery time to the market of a developing software. Second, it guides the effort estimation involved in software maintenance after delivery, because the maintenance effort is positively related to the defect number of a module. Third, knowing the possible defect number of a software can help software users to evaluate the quality of the software and determine when or whether they can install the software. In these situations, researchers employ some regression performance measures, such as Root Mean Square Error (RMSE), Average Absolute Error (AAE), Average Relative Error (ARE), Mean Magnitude of Relative Error (MMRE), Pred(l), and Completeness.

However, the number of defects in a software project is extremely imbalanced. There are many non-defective modules (i.e., the number of defects is zero), followed by the software modules with one defect, and very few modules with more than one defect. For example, only 11.1% modules in the project Ant 1.6 (See Table 1) contain two or three defects, and 13.1% modules contain one defects, while 73.8% modules are non-defective. Using the dataset with the imbalanced number of defects to construct a DNP model and then evaluating the error value of the constructed DNP model may lead to the over-optimistic estimation because of the over-specialization of the regression algorithm to the imbalanced dataset.

Example 2. Assuming that we have trained a model using the Ant 1.6 dataset, and then use it to predict the number of defects in the Ant 1.7 dataset, which has 745 modules and 338 defects. If the model predicts the number of defects in all modules to be 0, the AAE value of all the modules is 0.454 (=338/745), which may make the model appear to be quite accurate. The Rathore et al.'s paper [33] published in IEEE Transactions on Reliability 2019 (abbreviated as "Rathore 2019 TR paper" in the following) proposed a dynamic selection algorithm, and the results showed the AAE value of the dynamic selection algorithm trained on the Ant 1.6 dataset and tested on the Ant 1.7 dataset was 0.52 (See Table V in Rathore 2019 TR paper). In other words, if a model predicts the number of defects in all modules to be zero, the model outperforms the proposed algorithm by Rathore 2019 TR paper in terms of AAE. But the model is quite inaccurate and unacceptable to predict the number of defects in all defective modules to be zero. The AAE value for the modules with one defect is 1, the AAE value for the modules with two defects is 2, and the AAE value for the modules with three defects is 3. Since the non-defective modules occupy a large portion of the whole defect dataset, the AAE value (i.e., 0.454) of the all modules is small.

Therefore, the findings that some regression algorithms achieved good performance in terms of AAE, RMSE, ARE, Pred(l), and Completeness may be susceptible to the imbalanced defect datasets.

1.2. Our work and contributions

Considering this issue and some papers [16–18,38] are published in some top and high impact journals, we want to revisit the impact of regression algorithms for DNP. First, we conduct a literature review to identify and analyze a set of 23 primary DNP studies published until 2021.7 from different perspectives, including the used datasets, regression algorithms, and evaluation measures. Then, we find that 17 studies among all 23 studies employ the performance measures that evaluate regression models, such as RMSE, AAE, ARE, Pred(l), Completeness, etc. In other words, the main purpose of most existing DNP studies is still to evaluate the reliability and maintenance effort of each module according to the predicted precise number of defects. Then, we conduct the empirical study by raising the following Research Questions (RQs):

RQ1: How effective are the regression algorithms for predicting the precise number of defects?

We investigate the impact of some common used regression algorithms for DNP, including Poisson Regression (PR), Zero-Inflated Poisson Regression (ZIPR), Negative Binomial Regression (NBR), Zero-Inflated Negative Binomial Regression (ZINBR), Hurdle Poisson Regression (HPR), Genetic Programming (GP), Neural Network Regression (NNR), Decision Tree Regression (DTR), Linear Regression (LR), Bayesian Ridge Regression (BRR), Support Vector Regression (SVR), and K-nearest Neighbors Regression (KNR). We calculate the AAE and Pred(0.3) values for the modules with different numbers of defects separately. Specifically, we calculate the AAE and Pred(0.3) values for the non-defective modules, the AAE and Pred(0.3) values for the modules with one defect, the AAE and Pred(0.3) values for the modules with two defects, the AAE and Pred(0.3) values for the modules with three defects, the AAE and Pred(0.3) values for the modules with four, five, and six defects, and the AAE and Pred(0.3) values for the modules with more than six defects. The experimental results show the AAE

Table 1
The literature overview of the studies for predicting the numbers of defect.

Study	Corpus/Number	Regression algorithms[a]	Performance measures
Ostrand [18] 2005	ISS/12	Negative Binomial Regression (NBR)	PofB
Janes [19] 2006	ISS/5	Poisson Regression (PR), NBR, Zero-Inflated Negative Binomial Regression (ZINBR)	Alberg diagrams
Gao [20] 2007	ISS/1	PR, Zero-Inflated Poisson Regression (ZIPR), NBR, ZINBR, Hurdle Poisson Regression (HPR)	AAE, ARE
Afzal [21] 2008	ISS/3	Genetic Programming (GP)	Pred(*l*), MMRE, Spearman
Yu [22] 2012	PROMISE/5	NBR	Accuracy, Precision, Recall
Wang [15] 2012	Bugzilla and Jira/6	BugStates	Absolute Error (AE), Mean Absolute Error (MAE)
Rathore [23] 2015	PROMISE/10	Neural Network Regression (NNR), Genetic Programming (GP)	ARE, Recall, Completeness
Rathore [24] 2015	PROMISE/10	GP	ARE, Recall, Completeness
Chen [25] 2015	PROMISE/26	Linear Regression (LR), Bayesian Ridge Regression (BRR), Support Vector Regression (SVR), Nearest Neighbors Regression (NNR), Decision Tree Regression (DTR), Gradient Boosting Regression (GBR)	Precision, RMSE
Rathore [26] 2016	PROMISE/18	DTR	AAE, ARE, Pred(*l*)
Rathore [27] 2016	Eclipse/3	(Bagging/Boosting/Random subspace/Rotation Forest/Stacking)+(LR/Multilayer Perceptron Regression (MPR)/DTR)	AAE, ARE
Rathore [28] 2017	Firefox/3	NBR, ZIPR, MPR, GP, DTR, LR	AAE, ARE, Pred(*l*), Completeness
Rathore [29] 2017	PROMISE/11	Linear Regression based Combination Rule (LRCR), Gradient Boosting based Combination Rule (GRCR), MPR, GP, LR, NBR, ZIPR	AAE, ARE, Pred(l), Completeness
Rathore [30] 2017	PROMISE and Eclipse/17	Error Rate based Weighted Average (ERWA) combination rule, Linear Regression based Weighted Average (LRWA) combination rule, Decision Tree Forest based (DTF) ensemble method, Gradient Boosting Regression (GBR) based ensemble method, LR, MPR, DTR, GP, NBR, ZIPR	AAE, ARE, Pred(*l*), Completeness
Yu [31] 2017	PROMISE/22	(SMOTER/RUS/AdaBoost.R2)+(DTR/BRR/LR), SmoteNDBoost, RusNDBoost	FPA, Kendall
Zhang [14] 2018	Firefox/7	Sample entropy-Support Vector Regression (SSVR), Auto-Regressive Integrated Moving Average (ARIMA) model, X12-ARIMA model, NNR	Magnitude of Relative Error (MRE), MMRE
Wu [32] 2018	PROMISE/31	BRR, DTR, GBR, LR, NNR, MPR, and SVR	FPA
Rathore [33] 2019	PROMISE and Eclipse/19	A dynamic selection algorithm (DynSelection), LR, MPR, DTR, GP, NBR, ZIPR	AAE, ARE, Pred(*l*), Precision, Recall, F-measure
Chen [34] 2019	PROMISE/24	(SMOTER/SMOTUNED/AdaBoost.R2)+(DTR/BRR/LR)	FPA, Kendall
Huang [35] 2019	PROMISE/30	Multi-Project Regression (MPR), LR, NNR, SVR, DTR, BRR, GBR	AAE, ARE
Nevendra [36] 2019	PROMISE/15	AdaBoost.R2+(Extra Tree Regression (ETR)/Random Forest Regression (RFR)/Extreme Gradient Boosting Regression (EGBR)/GBR)	MAE, MRE
Qiao [17] 2020	PROMISE and ISS/2	Deep Learning Neural Network (DPNN), SVR, DTR, Fuzzy Support Vector Regression (FSVR), RFR	Mean Squared Error (MSE), R^2
Bal [37] 2020	PROMISE/26	Weighted Regularization Extreme Learning Machine (WR-ELM), Weighted Extreme Learning Machine (WELM), ELM, SmoteR+(ELM/SVR/NNR)	AAE, ARE, Pred(*l*),
Tong [38] 2021	PROMISE/27	Subspace Hybrid Sampling Ensemble (SHSE), SmoteR, SmoteRDE, DynSelection, SmoteNDBoost, RusNDBoost	FPA, Kendall, RMSE

[a](Bagging/Boosting/Random subspace/Rotation Forest/Stacking)+(LR/MPR/DTR) represents that the five ensemble learning methods (Bagging, Boosting, Random subspace, Rotation Forest, and Stacking) use LR, MPR, and DTR as the base learners. It is the same below.

value for the non-defective modules is low and the Pred(0.3) value for the non-defective modules is high, but the AAE value for the defective modules is very high and the Pred(0.3) value for the defective modules is low. It indicates that the regression algorithms tend to predict the modules non-defective, and have very poor capability to predict the number of defects in defective modules accurately.

RQ2: Are the performances of DNP models improved when applying data resampling and ensemble learning methods?

Since the imbalanced distribution of the number of defects affects the performance of DNP models, we apply Smote for Regression (SmoteR) and Random Over-Sampling (ROS) to alleviate the data imbalance problem. In addition, most studies [27,31,36] indicate that the ensemble learning method also can improve the performance of regression algorithms. Therefore, we also investigate whether applying AdaBoost.R2, Gradient Boosting Regression (GBR), and Bagging can boost the performance of regression algorithms. The experimental results show that AdaBoost.R2, SmoteR, and ROS can improve the performance of LR and BRR in terms of some performance measures, but it is still difficult for the algorithms to predict the precise number of defects in defective modules accurately.

RQ3: Are the performances of DNP models improved when applying feature selection and parameter optimization methods?

Yang et al. [41] pointed out that the severe multicollinearity problem (i.e., the existence of strong correlations among the software features) in the defect datasets may cause the poor performance of regression algorithms. Therefore, we apply the information gain feature selection method to investigate the effectiveness of the most relevant software features for DNP. In addition, Tantithamthavorn et al. [42] found that the auto parameter optimization of classification algorithms can boost the performance of defect prediction models. Therefore, we also investigate whether the grid search parameter optimization method can improve the performance of DNP models. The experimental results show that the information gain and grid search methods cannot improve the performance of DTR, LR, and BRR in terms of most performance measures.

The contributions of this paper are as follows:

•We conduct a meta-study and revisit the impact of 12 regression algorithms, two data resampling algorithms, three ensemble learning algorithms for DNP. We propose to calculate the AAE and Pred(0.3) values for the modules with different numbers of defects separately instead of the whole AAE and Pred(0.3) values of all modules, and find that predicting the precise number of defects in defective modules is very hard.

•We identify and analyze a set of 23 DNP studies until 2021 from different perspectives, including the used datasets, the regression algorithms, and the evaluation measures. Researchers can use the set as a starting point to conduct further DNP studies. To the best of our knowledge, this is the first work to conduct a comprehensive literature review of DNP studies.

•We provide some suggestions to practitioners and researchers in this domain, including (1) researchers should evaluate the AAE and Pred(0.3) values for the modules with different numbers of defects separately when predicting the precise number of defects. Therefore, some DNP papers using regression performance measures to evaluate the proposed method should be revisited. (2) the title of the paper should be “Ranking-Oriented Defect Prediction” rather than “Predict the number of defects/bugs” to avoid ambiguity when the purpose of DNP is to rank software modules.

•We open-source our source code[1] and datasets to facilitate the replication of our study and conduct further works.

1.3. Organization

The remainder of this paper is organized as follows. Section 2 introduces the literature review. Section 3 introduces the investigated regression algorithms, the data resampling algorithm, the ensemble learning algorithms, the feature selection method, and the parameter optimization method. Sections 4 and 5 present the experimental setup and results. Section 6 presents the implications of our findings and the potential threats of validity. Finally, we conclude the paper in Section 7.

2. Literature review

2.1. Literature research

To know about the progress of the DNP studies, we conducted a literature research to find all related DNP papers that are published between 2005 and 2021.7.[2] To the best of our knowledge, the first paper with the title containing the terms “number of faults/defects/bugs” was published by Ostrand et al. [18] in IEEE Transactions on Software Engineering 2005 (abbreviated as “Ostrand 2005 TSE paper” in the following), therefore, we set the starting year of the literature research to 2005. The searched papers should meet the following criteria:

• The paper should be written using English.

• The full text of the paper should be available.

• We only choose the journal version, if the paper had the journal and conference versions.

We searched related papers using Google Scholar and DBLP. We followed the approach of Zhou et al. [43] to conduct a forward snowballing search by recursively inspecting the papers which cited the “Ostrand 2005 TSE paper”. To be more specific, we first searched and found the relevant papers that cited the paper. Then, we repeated the process on all the relevant papers. Consequently, 23 related papers were identified in the literature. Table 1 presents an overview of the DNP studies. For each study, the first column in the table lists the authors and the published year; the second column lists the experimental datasets; the third column lists the investigated regression algorithms; and the last column lists the evaluation measures used in the study.

2.2. DNP

As shown in Table 1, several studies [18,19,21,31,32,34] used some ranking performance measures, such as PofB, Alberg diagrams, FPA, Spearman rank correlation coefficient, and Kendall rank correlation coefficient. In other words, the studies first used the regression algorithms to predict the number of defects in software modules, and then used the predicted value to sort the software modules. PofB evaluates how many bugs can be found when inspecting the top 20% Lines Of Code (LOC), Alberg diagrams and FPA evaluates the global ranking of modules according to the number of defects, and Spearman and Kendall rank correlation coefficients measure the ordinal association between two modules. In addition, some studies [22–24] investigated accuracy, precision, and recall as the performance measures. These studies first used the regression algorithms to predict the number of defects in software modules, and then used the predicted value to classify the modules (i.e., modules whose predicted number of defects is larger than zero are predicted as defective; otherwise, non-defective).

Most DNP studies [14–17,20,21,23–26,29,30,33,35–37,44] employed AAE, ARE, MMRE, AE, MAE, RMSE, MRE, MSE, Pred(l), and Completeness as the performance measures. In these studies, Wang et al. [15] proposed the BugStates method based on the state transition model. Zhang et al. [14] proposed SamEN-SVR using time series analysis. Huang et al. [35] proposed a multi-project regression method to take full advantage of relatedness among projects for DNP. Rathore et al. [36] proposed a dynamic selection method to dynamically choose regression algorithms to build DNP models according to the similarity between testing modules and training modules. Qiao et al. [17] investigated the performance of deep neural network for DNP. To improve the performance, Rathore et al. [27–30] investigated some ensemble learning methods for DNP, such as Bagging, Boosting, random subspace, rotation forest, stacking, linear regression based combination rule, gradient boosting based combination rule, error rate based weighted average combination rule, linear regression based weighted average combination rule, and decision tree forest based ensemble method. Bal et al. [37] proposed a weighted extreme learning machine method to alleviate the data imbalance problem.

The other studies [20,21,23–26,28] performed the empirical investigation of several regression algorithms for DNP, and found that DTR, LR, and RR achieved better RMSE, AAE, Pred(l), and Completeness values. However, the findings may be susceptible due to the imbalanced distribution of the number of defects. Even the DNP model predicts the number of defects in all modules to be 0, the performance measure values will be good. Therefore, we revisit the impact of these regression algorithms for predicting the precise number of defects by calculating the AAE and Pred(0.3) values for modules with the different numbers of defects separately.

2.3. Ranking-oriented defect prediction

Except for the regression algorithm, some researchers employ the pairwise learning to rank algorithm and listwise learning to rank algorithm to sort software modules according to the number of defects, and call it Ranking-Oriented Defect Prediction (RODP) [45,46]. Dissimilar to the regression algorithm that first predicts the number of defects and then ranks software modules according to the predicted value, the pairwise and listwise learning to rank algorithms directly predict the ranking of software modules.

[1] https://github.com/xiaoyu-whu/Revisitng-IST.

[2] The search was conducted in 2021.7.

The pairwise algorithm predicts the relationship between any two modules. For example, there are the three modules, i.e., M_1, M_2, and M_3. The pairwise algorithm predicts that M_1 has more defects than M_2, M_2 has more defects than M_3, and M_1 has more defects than M_3. Then, the final ranking of M_1, M_2, and M_3 is $M_1 > M_2 > M_3$. Nguyen et al. [47] investigated the performance of two pairwise learning to rank algorithms (i.e., Ranking SVM and RankBoost) for RODP, and found that these algorithms outperformed the linear regression algorithm 4% to 21% in terms of the Spearman rank correlation coefficient. In our previous study [46], we proposed a cost-sensitive ranking SVM (CSRankSVM) method, which considers the cost of the wrong ranking of the modules with more defects.

The listwise algorithm directly optimizes the performance measures to obtain a ranking model. Yang et al. [48] proposed a listwise learning to rank approach called LTR to rank software modules according to their predicted number of defects. You et al. [45] proposed a ranking-oriented cross-project defect prediction model to rank cross-project modules based on the predicted number of defects.

Recently, Yang et al. [41] investigated the performance of five regression algorithms and LTR for RODP on 41 PROMISE datasets, and we [49] investigated 6 classification algorithms, 9 regression algorithms, 4 pairwise learning to rank algorithms, and 4 listwise learning to rank algorithms for ranking software modules according to the number of defects on 41 PROMISE datasets. The experimental results showed that Bayesian ridge regression achieved better performance in terms of FPA.

3. Investigated algorithms for DNP

Let $\boldsymbol{m}_i = (\boldsymbol{x}_i, y_i)$ represent a software module, where $\boldsymbol{x}_i = (x_1, x_2, \ldots, x_d)$ is a d-dimensional feature vector of the ith software module, and y_i is the number of defects in the ith software module. A defect dataset S is represented as:

$$S = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_n\}, \tag{1}$$

where n is the number of modules in S. The purpose of DNP is to train a prediction model from S:

$$y = F(\boldsymbol{x}), \tag{2}$$

where $\boldsymbol{x}$ is feature vector, and y is the predicted number of defects. It is worth noting that the predicted number of defects by the regression algorithms will be rounded to an integer, and be set as zero if the predicted number of defects is negative.

3.1. Regression algorithms

We investigate the performance of all base algorithms investigated in the 23 previous DNP studies shown in Table 1, except BugStates, Multilayer Perceptron Regression (MPR), Sample entropy–Support Vector Regression (SSVR), Auto-Regressive Integrated Moving Average (ARIMA) model, X12-ARIMA model, Deep Learning Neural Network (DPNN), and Fuzzy Support Vector Regression (FSVR). The reasons are as follows: (a) BugStates uses the Markovian method to predict the number of defects at the current state based on the state transition model. SSVR, ARIMA, and X12-ARIMA use time series analysis to predict the number of defects. Since our defect datasets do not contain the information of states and time series, we do not investigate the performance of BugStates, SSVR, ARIMA, and X12-ARIMA. (b) MPR and DPNN are the same types of algorithms as NNR, FSVR is the same type of algorithm of SVR, so we only investigate NNR and SVR. Therefore, we investigate the 12 regression algorithms.

(1) Poisson Regression (PR) [50]: It takes the assumption that the response variable follows the Poisson distribution, and the mean and the variance should be equal. It is normally used to investigate the relationship between the response variable and a count of events.

(2) Zero-Inflated Poisson Regression (ZIPR) [51]: It is used to model count data that has an excess of zero counts. The excess zeros are generated by two processes independently. The first process generates structural zeros by a binary distribution. The second process generates counts by a Poisson distribution.

(3) Negative Binomial Regression (NBR) [52]: It is an extension of the Poisson regression model. It accommodates the overdispersion and loosens the assumption that the Poisson regression model requires the mean and the variance be equal.

(4) Zero-Inflated Negative Binomial Regression (ZINBR) [53]: It is usually used to model over-dispersed count outcome variables that have an excess of zero counts. The excess zeros are generated by two processes independently. The first process generates structural zeros by a binary distribution. The second process generates counts by a negative binomial distribution.

(5) Hurdle Poisson Regression (HPR) [54]: Hurdle count models are two-component models with a truncated count component for positive counts, and a hurdle component that models the zero counts. When the Poisson distribution is used to represent the two-component models, the resulting model is a HPR model.

(6) Genetic Programming (GP) [55]: It trains a linear model:

$$y = f(\mathbf{b}, \mathbf{x}) = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_d x_d, \tag{3}$$

where $\boldsymbol{b} = (b_0, b_1, \ldots, b_d)$ is a $(d+1)$-dimensional vector of regression coefficients, $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ is a d-dimensional feature vector of the ith software module, and y is the predicted number of defects in the module. Then, it minimizes the following loss function by employing the genetic algorithm to find the optimal $\boldsymbol{b}$ value:

$$\sum_{i=1}^{n} (y_i - f(\mathbf{b}, \boldsymbol{x}_i))^2. \tag{4}$$

We set the feasible solution space to [−2, 2], and the population size and maximal generation were set to 100, since higher population size and maximal generation values do not improve the performance of GP significantly.

(7) Neural Network Regression (NNR) [56]: It trains a non-linear function approximator by employing the backpropagation method with no activation function in the output layer.

(8) Decision Tree Regression (DTR) [57]: It trains a regression model with the decision tree structure based on the training software modules.

(9) Linear Regression (LR) [58]. It trains the same linear model as GP, then minimizes the same loss function as GP by using the least square method to find the optimal $\boldsymbol{b}$ value.

(10) Bayesian Ridge Regression (BRR) [59]: It trains the same linear model as LR, but minimizes the following loss function by using the Bayes theorem to find the optimal $\boldsymbol{b}$ value:

$$\sum_{i=1}^{n} (y_i - f(\mathbf{b}, \boldsymbol{x}_i))^2 + \lambda |\mathbf{b}|^2, \tag{5}$$

where λ is a small regularization parameter.

(11) Support Vector Regression (SVR) [60]: It employs the same principles as SVM with a minuscule difference. Because the output of SVR is a real number, it is hard to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance is set in approximation to the SVM which would have already requested from the problem.

(12) K-nearest Neighbors Regression (KNR) [61]: It first selects k software modules that are the nearest to the new module, and then predicts the number of defects of the new module according to the mean of those of these nearest neighbors.

We implement PR, ZIPR, NBR, ZINBR, and HPR using the R package, and other algorithms using the Sklearn package. We use the default parameters for NNR, DTR, BRR, SVR, and KNR.

3.2. Data resampling algorithms

(1) Smote for Regression (SmoteR) [62]: It first randomly chooses m defective modules from the training dataset (m is calculated according to the desired ratio value of the defective modules). For each chosen defective module M_A, it randomly chooses one of its k nearest neighbors, i.e., M_B. The feature of the synthetic module $M_{synthetic}$ is

$$\boldsymbol{x}_{synthetic} = \boldsymbol{x}_A + Random(0,1) \times (\boldsymbol{x}_B - \boldsymbol{x}_A), \tag{6}$$

where $\boldsymbol{x}_{synthetic}$, $\boldsymbol{x}_A$, and $\boldsymbol{x}_B$ are the feature of $M_{synthetic}$, M_A, and M_B, respectively. Since SmoteR probably chooses two modules with different numbers of defects to generate a synthetic module, it uses the following approach to decide the number of defects:

$$y_{synthetic} = (d_2 \times y_A + d_1 \times y_B)/(d_1 + d_2), \tag{7}$$

where $y_{synthetic}$, y_A, and y_B are the numbers of defects in $M_{synthetic}$, M_A, and M_B respectively, d_1 is the distance between $M_{synthetic}$ and M_A, and d_2 is the distance between $M_{synthetic}$ and M_B. In SmoteR, the default desired ratio value of the defective modules is 0.5.

(2) Random Over-Sampling (ROS) randomly chooses m defective modules from the original dataset (m is calculated according to the desired ratio value of the defective modules), and then adds them to the original dataset to form the resampled dataset. In ROS, the default desired ratio value of the defective modules is 0.5.

3.3. Ensemble learning algorithms

(1) AdaBoost.R2 [63]: It is a Boosting algorithm for regression problem. Similar to other Boosting algorithms, it first assigns each module an equal weight. Then, it iteratively trains t weak regression models based on the training dataset, and the weights of all training modules are updated according to the loss function. Finally, the t weak regression models are combined to construct the final strong regression model, and the output of the strong regression model is the weighted median of that of the t weak regression models.

(2) Gradient Boosting Regression (GBR) [64] : It builds the model in a forward stage-wise fashion like other Boosting algorithms. The difference is that GBR trains a DTR model based on the negative gradient of the loss function in each iteration, then combines t DTR models to a strong regression model.

(3) Bagging [65]: Given a training dataset which contains n software modules, it first generates t new training datasets, each of which also contains n software modules that are sampled from the original dataset with replacement. Then, it trains t weak regression models on the t new training datasets. The output of the final strong regression model is the average of that of the t weak regression models.

For SmoteR, ROS, AdaBoost.R2, and Bagging, we employ DTR, LR, and BRR as the base learners, since previous studies [25,28,31,34] pointed out that the three regression algorithms achieved better performance for DNP, and our experimental results also show that the three regression algorithms perform better than other investigated regression algorithms. We denote SmoteR, ROS, AdaBoost.R2, and Bagging that employ DTR, LR, and BRR as the base learners as SmoteR+DTR, SmoteR+LR, SmoteR+BRR, ROS+DTR, ROS+LR, ROS+BRR, AdaBoost.R2+DTR, AdaBoost.R2+LR, AdaBoost.R2+BRR, Bagging+DTR, Bagging+LR, and Bagging+BRR. For GBR, we only use DTR as the base learner, since GBR cannot use other algorithms as the base learners. We utilize the Sklearn package to implement AdaBoost.R2, Bagging, and GBR, and use the default parameters for the three algorithms.

Table 2
The optimized parameter of DTR, LR, and BRR.

Algorithm	Parameter description	Candidate parameter values (default value is in bold)
DTR	The minimum number of samples required to split an internal node	{**2**,3,4,5,6}
LR	Whether the regressor will be normalized before	{true, **false**}
BRR	Precision of the solution	{0.1, 0.01, **0.001**, 0.0001, 0.00001}

3.4. Information gain

Information Gain (IG) measures the reduction of uncertainty about the number of defects after observing the software features. The bias of information gain is that it tends to select the software features with more values [66]. We employ information gain as the feature selection method for two reasons. First, some previous studies [67,68] have shown its effectiveness for classification-based defect prediction models. Second, the research on feature selection for DNP is limited. Only Yang et al. [48] and Yu et al. [46] employed information gain as the feature selection method for ranking software modules based on the predicted number of defects. We select the top $log_2 m$ features, where m is the number of software features, following the parameter setting in Khoshgoftaar et al.'s work [69,70] that suggested various prediction models for imbalanced defect datasets are appropriate to the setup. We denote DTR, LR, and BRR trained on the selected features by information gain as IG+DTR, IG+LR, and IG+BRR.

3.5. Grid search

Grid Search (GS) is one of the most commonly used parameter optimization methods. It divides the domain of the parameters of regression algorithms into a discrete grid. Then, it tries every combination of values of the grid, calculates the performance measure using cross-validation, and chooses the parameter that achieves the best performance as the recommended one. In this study, we use the sum value of AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} as the performance measure. The optimized parameter of DTR, LR, and BRR are shown in Table 2. We denote DTR, LR, and BRR that use grid search to optimize the parameter as GS+DTR, GS+LR, and GS+BRR.

4. Experimental setup

4.1. Datasets

Some researchers have published some defect data repositories, such as NASA [71], SOFTLAB [72], the just-in-time data repository [73] that only contain the information of the class label. However, we aim to predict the number of defect in software modules, so we only choose the datasets that contain the information of the number of defects as the experiment datasets. Similar to Rathore [44] and Bal [37], we choose the defect datasets from the PROMISE repository [74]. Since it is more practical to use the previous version for training and the current version for testing, we only choose the projects with multi versions from the PROMISE repository. In addition, we choose the versions that contain more than 300 modules, since we want to train the DNP model using more training data.

Table 3 provides the details about the 18 chosen defect datasets from the PROMISE repository, including the number of software modules in the version ($\#M$), the percentage of defective software modules(%D), the percentage of modules with one defect (%1), the percentage of modules with two defects (%2), the percentage of modules with three defects (%3), the percentage of modules with four, five, and six defects (%4–6), the percentage of modules with more than six defects (%>6), the average number of defects in the version (Avg), and the maximum number of defects in the version (Max). The datasets have the 20 software features, for details of the features, please refer to [74].

Table 3
The details of the 18 chosen defect datasets.

Dataset	#M	%D	%1	%2	%3	%4–6	%>6	Avg	Max
Ant 1.6	351	26.2	13.11	6.27	4.84	1.43	0.57	2.00	10
Ant 1.7	745	22.3	12.48	4.03	2.15	3.09	0.54	2.04	10
Camel 1.2	608	35.5	16.28	8.22	4.77	4.28	1.97	2.42	28
Camel 1.4	872	16.6	8.14	3.90	2.06	1.72	0.80	2.31	17
Camel 1.6	965	19.5	10.47	3.32	1.66	2.28	1.76	2.66	28
Jedit 4.0	306	24.5	11.77	5.23	2.94	1.96	2.62	3.01	23
Jedit 4.1	312	25.3	11.86	4.49	3.21	3.53	2.24	2.75	17
Jedit 4.2	367	13.1	7.36	2.45	1.36	1.09	0.82	2.21	10
Poi 2.0	314	11.8	11.15	0.64	0.00	0.00	0.00	1.05	2
Poi 2.5	385	64.4	20.00	36.88	2.86	3.38	1.30	2.00	11
Poi 3.0	442	63.6	45.48	9.73	3.17	3.39	1.81	1.78	19
Xalan 2.4	723	15.2	10.93	3.04	0.83	0.28	0.14	1.42	7
Xalan 2.5	803	48.2	36.99	8.10	1.25	1.62	0.25	1.37	9
Xalan 2.6	885	46.4	30.62	11.30	2.60	1.81	0.11	1.52	9
Xalan 2.7	909	98.8	72.61	20.79	3.85	1.21	0.33	1.35	8
Xerces 1.2	440	16.1	7.05	8.64	0.00	0.46	0.00	1.62	4
Xerces 1.3	453	15.2	5.96	4.86	2.21	1.33	0.88	2.80	30
Xerces 1.4	588	74.3	30.78	21.94	4.42	6.29	10.89	3.65	62
Average	581.56	35.39	20.41	5.87	2.08	2.77	4.08	2.11	16.89

4.2. Performance measures

The above mentioned AAE, ARE, MMRE, AE, MAE, RMSE, MRE, and MSE performance measures in Table 1 all evaluate the difference between the predicted number of defects and the actual number of defects. Due to the space limit, we only use AAE as the performance measure, since it can present the difference more intuitively.

The definition of AAE is as follows:

$$AAE = (1/n)\sum_{i=1}^{n}(|\hat{y}_i - y_i|), \tag{8}$$

where n is the number of software modules in the testing dataset, y_i is the predicted number of defects in the ith software module, and $\hat{y}_i$ is the actual number of defects.

In addition, we use AAE$_j$ as the performance measure to calculate the AAE value for the modules with different numbers of defects separately:

$$AAE_j = (1/n_j)\sum_{i=1}^{n_j}(|\hat{y}_i - y_i|), \tag{9}$$

where n_j is the number of software modules with j defects, and j can be 0, 1, 2, 3, 4–6, and 6+, respectively. In other words, AAE$_0$ evaluates the AAE value of all non-defective modules, AAE$_1$ evaluates the AAE value of all modules with one defect, AAE$_2$ evaluates the AAE value of all modules with two defects, AAE$_3$ evaluates the AAE value of all modules with three defects, AAE$_{4-6}$ evaluates the AAE value of all modules with four, five, and six defects, and AAE$_{6+}$ evaluates all AAE value of the modules with more than six defects, respectively. We regard the modules with four, five, and six defects as a group, and the modules with more than six defects as a group to calculate the AAE value.

Pred(l) is the ratio of the modules whose relative errors are within l% of the actual number of defects to all modules in the testing dataset:

$$Pred(l) = k/n, \tag{10}$$

where n is the number of software modules in the testing dataset, and k is the number of modules whose relative errors are within l% of the actual number of defects. Following the previous DNP studies [33,37], we set l as 30. In addition, we use Pred(0.3)$_j$ as the performance measure to calculate the Pred(0.3) value for the modules with different numbers of defects separately:

$$Pred(0.3)_j = k_j/n_j, \tag{11}$$

where n_j is the number of software modules with j defects, k_j is the number of software modules with j defects whose relative errors are within 0.3×j, and j can be 0, 1, 2, 3, 4–6, and 6+, respectively.

Completeness is the ratio of the total predicted number of defects to the total actual number of defects of all modules in the testing dataset. We can also use Completeness$_j$ to calculate the Completeness value for the modules with different numbers of defects separately, which is the ratio of the total predicted number of defects to the total actual number of defects of all modules with j bugs in the testing dataset. For example, Completeness$_1$ is the ratio of the total predicted number of defects to the total actual number of defects of all modules with one defect in the testing dataset. Rathore et al. [30] pointed out that the DNP model with Completeness equal to 1 is preferred. However, we find that Completeness$_1$ is not appropriate to evaluate DNP models. For example, there are five software modules with one defect, a DNP model predicts a module contains five defects, and other modules are non-defective. Therefore, the Completeness$_1$ is equal to 1. However, the DNP model predict the five modules very inaccurately. Therefore, we do not employ Completeness to evaluate DNP models.

4.3. Experimental procedure

We employ the cross-version validation to evaluate the performance of DNP models, since it is more practical in an actual testing scenario than cross-validation. We use the previous version for training, and the current version for testing. For example, we use Ant 1.6 for training, and Ant 1.7 for testing. We repeat the cross-version validation procedure 50 times to avoid the bias when using the data imbalance learning methods. Therefore, we can get 50 results for each regression algorithm on each pair of training and testing dataset. We record the median value of the 50 results for each algorithm on each dataset as the experiment results.

Then, we employ the Scott–Knott with Cohen's d effect size awareness (Scott–Knott ESD) [75] test to divide the regression algorithms into different ranks at the significance level of 0.05 using the hierarchical clustering algorithm. The test could group the regression algorithms distinctly without any overlapping compared with other statistical tests such as the Wilcoxon test [76]. The regression algorithms in the same rank have no statistically significant difference, while the regression algorithms in different ranks have statistically significant difference.

5. Experimental results

5.1. RQ1: How effective are the regression algorithms for predicting the precise number of defects?

Methods: To answer the RQ1, we calculate the AAE and Pred(0.3) values for the 12 regression algorithms. In addition, we calculate the AAE$_0$, AAE$_1$, AAE$_2$, AAE$_3$, AAE$_{4-6}$, AAE$_{6+}$, Pred(0.3)$_0$, Pred(0.3)$_1$, Pred(0.3)$_2$, Pred(0.3)$_3$, Pred(0.3)$_{4-6}$, and Pred(0.3)$_{6+}$ to evaluate the AAE and Pred(0.3) values for the non-defective modules, the modules with one defect, the modules with two defects, the modules with three defects, the modules with four, five, and six defects, and the modules with more than six defects, respectively. Figs. 2 and 4 show the distribution of the AEE, AAE$_0$, AAE$_1$, AAE$_2$, AAE$_3$, AAE$_{4-6}$, and AAE$_{6+}$ values with the Scott–Knott ESD test over all testing datasets. Figs. 3 and 5 shows the distribution of the Pred(0.3), Pred(0.3)$_0$, Pred(0.3)$_1$, Pred(0.3)$_2$, Pred(0.3)$_3$, Pred(0.3)$_{4-6}$, and Pred(0.3)$_{6+}$ values with the Scott–Knott ESD test over all testing datasets. Different colors of the boxplot indicate different Scott–Knott ESD test ranks. From top down, the order is red, green, blue, purple, orange, and black. Each box-plot presents the 75th percentile (the upper side of the box), the median (the horizontal line within the box), and the 25th percentile (the lower side of the box). Table 4 lists the average AEE, AAE$_0$, AAE$_1$, AAE$_2$, AAE$_3$, AAE$_{4-6}$, and AAE$_{6+}$ values of the 12 regression algorithms over all testing datasets. Table 5 lists the average Pred(0.3), Pred(0.3)$_0$,

Table 4
The average AAE values of the 12 regression algorithms on the testing datasets. (The best value in each row is bold.)

Measures	PR	ZIPR	NBR	ZINBR	HPR	GP	NNR	DTR	LR	BRR	SVR	KNR
AAE_0	0.58	0.66	**0.08**	0.75	0.82	2.06	0.48	0.26	0.33	0.33	0.13	0.29
AAE_1	0.90	0.96	0.96	1.13	1.03	2.88	0.92	0.96	0.74	**0.71**	0.82	0.77
AAE_2	1.76	1.79	1.94	1.79	1.99	3.01	1.54	1.65	**1.35**	**1.35**	1.60	1.43
AAE_3	2.64	2.60	2.92	2.55	3.12	4.68	2.21	2.27	**2.12**	2.13	2.40	2.08
AAE_{4-6}	4.56	4.58	4.55	4.56	4.45	7.38	3.88	3.55	3.45	3.51	3.87	**3.33**
AAE_{6+}	9.88	9.85	10.31	10.06	9.85	13.70	9.81	**7.32**	7.65	7.67	9.27	7.77
AAE	1.13	1.19	0.92	1.27	1.38	2.65	1.05	0.90	0.85	0.85	**0.81**	0.87

Table 5
The average Pred(0.3) values of the 12 regression algorithms on the testing datasets. (The best value in each row is bold.)

Measures	PR	ZIPR	NBR	ZINBR	HR	GP	NNR	DTR	LR	BRR	SVR	KNR
$Pred(0.3)_0$	0.68	0.65	**0.95**	0.66	0.66	0.49	0.72	0.83	0.73	0.73	0.88	0.77
$Pred(0.3)_1$	0.27	0.27	0.05	0.27	0.23	0.10	0.27	0.20	0.34	**0.35**	0.18	0.29
$Pred(0.3)_2$	0.04	0.04	0.01	0.04	0.05	0.09	0.10	0.08	**0.14**	0.13	0.09	0.12
$Pred(0.3)_3$	0.04	0.03	0.02	0.04	0.04	0.08	**0.30**	0.26	0.25	0.25	0.14	0.26
$Pred(0.3)_{4-6}$	0.02	0.01	0.00	0.02	0.01	0.09	0.07	**0.14**	0.09	0.09	0.03	0.13
$Pred(0.3)_{6+}$	0.00	0.00	0.00	0.00	0.00	0.05	0.18	**0.23**	0.07	0.08	0.00	0.09
Pred(0.3)	0.49	0.48	0.55	0.48	0.48	0.35	0.49	0.54	0.53	0.53	**0.59**	0.53

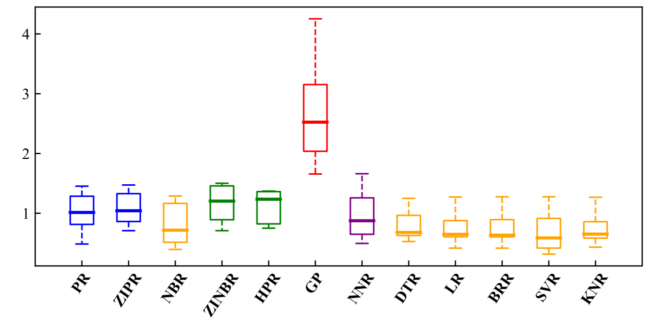


Fig. 2. The boxplot of the AAE values of the regression algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

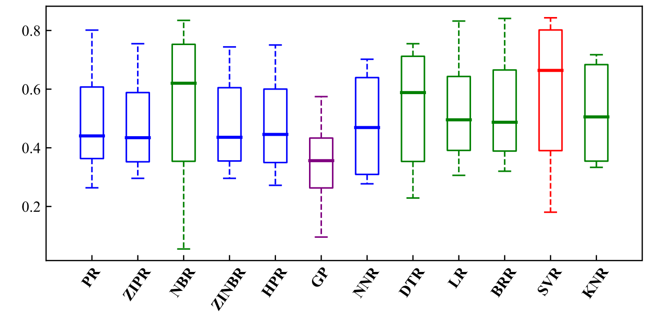


Fig. 3. The boxplot of the Pred(0.3) values of the regression algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$Pred(0.3)_1$, $Pred(0.3)_2$, $Pred(0.3)_3$, $Pred(0.3)_{4-6}$, and $Pred(0.3)_{6+}$ values of the 12 regression algorithms over all testing datasets.

Results: From these tables and figures, we have the following observations:

(1) The median and average AAE values of NBR, DTR, LR, BRR, SVR, and KNR are very low, which are less than one. The Scott–Knott ESD test results also show NBR, DTR, LR, BRR, SVR, and KNR significantly perform better than other regression algorithms in terms of AAE. The average Pred(0.3) value of SVR is 0.59, which is significantly higher than that of other 11 regression algorithms according to the Scott–Knott ESD test. In addition, the average Pred(0.3) values of other algorithms are also high, and GP achieves the lowest Pred(0.3) value (0.35).

(2) The median and average AAE_0 values of NBR, DTR, LR, BRR, SVR, and KNR are very low, which are less than 0.34. The median and average $Pred(0.3)_0$ values of the six algorithm are very high, which are larger than or equal to 0.73. Especially, the average AAE_0 values of NBR and SVR are 0.08 and 0.13, respectively, and the average $Pred(0.3)_0$ values of NBR and SVR are 0.95 and 0.88, respectively. The Scott–Knott ESD test results also show NBR and SVR outperform other regression algorithms in terms of AAE_0 and $Pred(0.3)_0$. The results indicate that NBR, DTR, LR, BRR, SVR, and KNR can predict the non-defective modules accurately.

(3) The average AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values of the 12 regression algorithms are very high. LR achieves the lowest average AAE_2 and AAE_3 values, which are 1.35 and 2.12. BRR achieves the lowest average AAE_1 and AAE_2 values, which are 0.71 and 1.35. DTR achieves the lowest average AAE_{6+} value (7.32). KNR achieves the lowest average AAE_{4-6} value (3.33). The Scott–Knott ESD test results also show that DTR, KNR, LR, and BRR significantly outperform other regression algorithms in terms of AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+}.

(4) The average $Pred(0.3)_1$, $Pred(0.3)_2$, $Pred(0.3)_3$, $Pred(0.3)_{4-6}$, and $Pred(0.3)_{6+}$ values of the 12 regression algorithms are very low. BRR achieves the highest average $Pred(0.3)_1$ value (0.35), which indicates that average 35% modules with one defect are predicted to contain one defect accurately. LR and NNR achieve the highest average $Pred(0.3)_2$ and $Pred(0.3)_3$ values (0.14 and 0.30, respectively). DTR achieves the highest average $Pred(0.3)_{4-6}$ and $Pred(0.3)_{6+}$ values (0.14 and 0.23, respectively). The Scott–Knott ESD test results also show that LR and BRR significantly outperform other algorithms in terms of $Pred(0.3)_1$ and $Pred(0.3)_2$; NNR significantly performs the best in terms of $Pred(0.3)_3$; DTR significantly outperforms other algorithms in terms of $Pred(0.3)_{4-6}$ and $Pred(0.3)_{6+}$.

(5) The overall results show that the 12 regression algorithms tend to predict the number of defects of the modules as zero. Therefore, the AAE_0 values of the algorithms are low and the $Pred(0.3)_0$ are high. But the algorithms have very poor capability to predict the number of defects in defective modules accurately. However, the main purpose of defect prediction is to predict the defective modules rather than the non-defective ones accurately. Since the non-defective modules occupy a large proportion, the AAE values of the regression algorithms are low, and the Pred(0.3) values are high.

Answer to RQ1: The regression algorithms are very inaccurate for predicting the precise number of defects in defective modules.

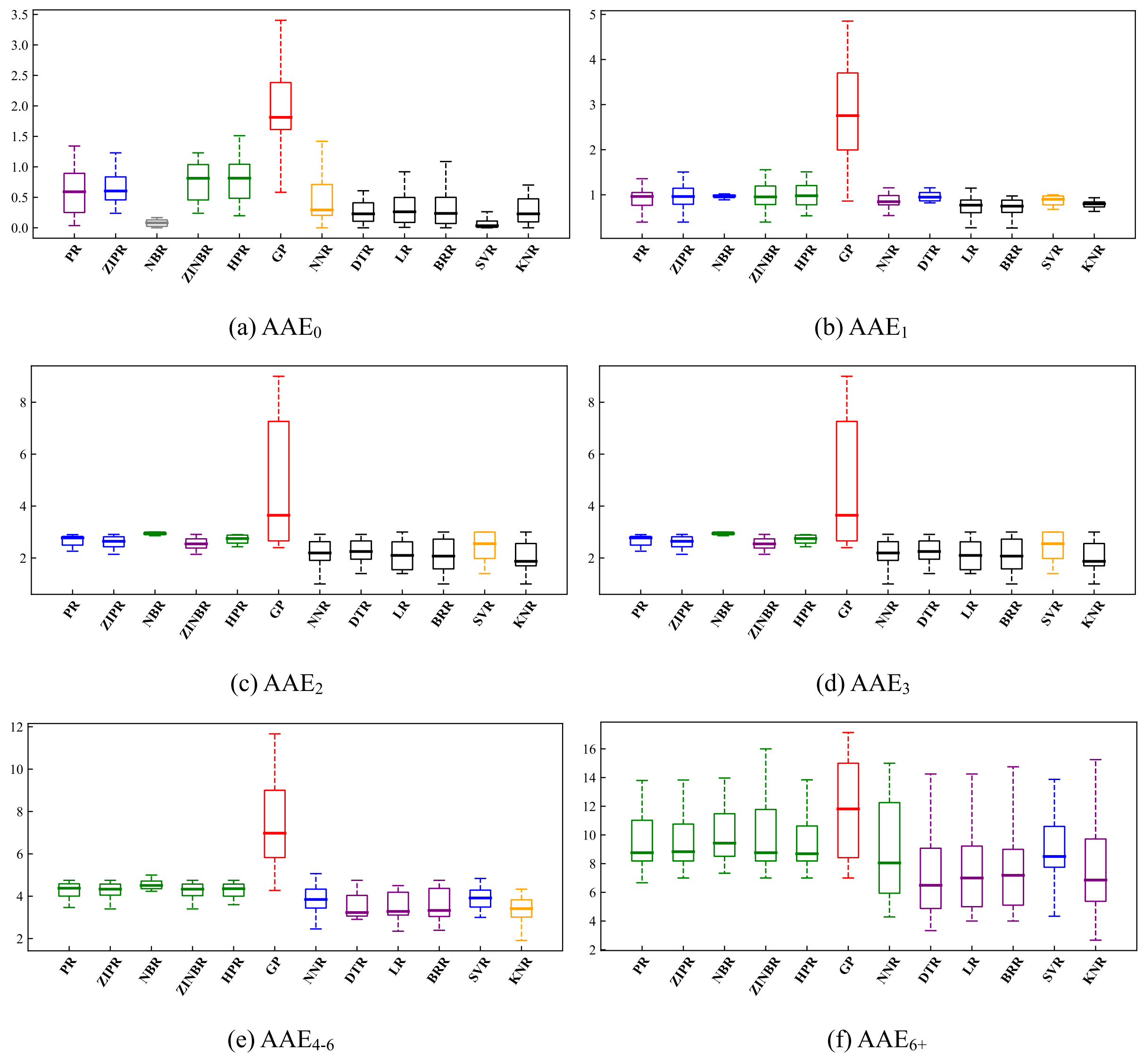


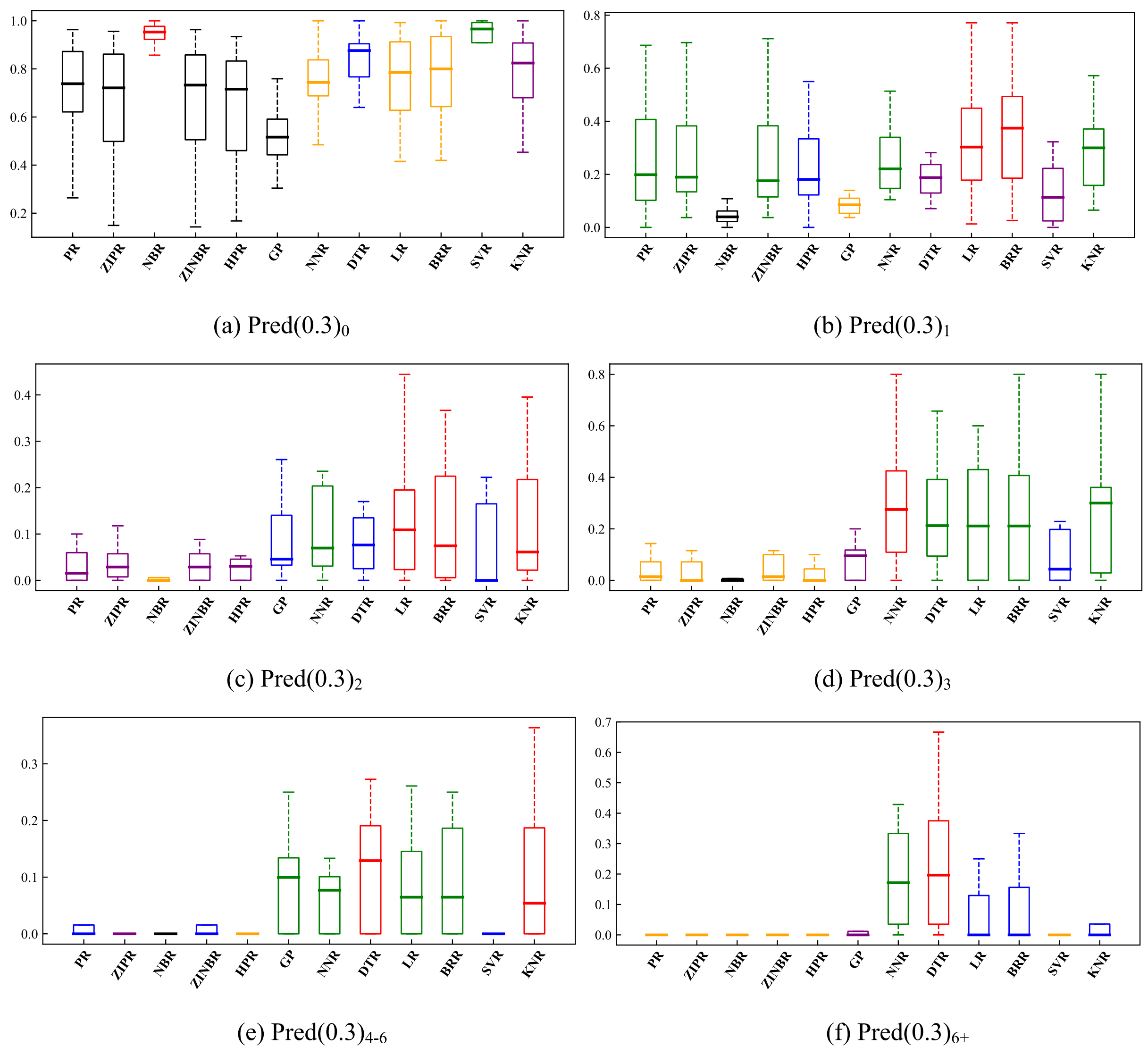
Fig. 4. The boxplot of the AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+} values of the 12 regression algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 6
The average AAE values of the 13 regression algorithms on the testing datasets. (The bold value means the improved performance after applying data resampling and ensemble learning algorithms.)

Measures	GBR	AR+DTR	AR+LR	AR+BRR	BG+DTR	BG+LR	BG+BRR	SR+DTR	SR+LR	SR+BRR	ROS+DTR	ROS+LR	ROS+BRR
AAE_0	**0.25**	**0.24**	0.74	0.71	0.27	**0.32**	**0.31**	0.28	0.60	0.63	0.27	0.61	0.63
AAE_1	**0.79**	**0.85**	0.75	0.73	**0.74**	**0.72**	0.72	0.97	**0.59**	**0.56**	**0.93**	**0.61**	**0.60**
AAE_2	**1.44**	**1.50**	**1.22**	**1.20**	**1.40**	**1.34**	1.35	**1.62**	**1.13**	**1.08**	**1.57**	**1.11**	**1.08**
AAE_3	**2.25**	**2.24**	**1.95**	**1.89**	**2.09**	2.12	2.14	**2.20**	**1.84**	**1.79**	**2.24**	**1.84**	**1.79**
AAE_{4-6}	**3.41**	**3.39**	**3.18**	**3.08**	**3.25**	**3.35**	**3.40**	**3.42**	**3.09**	**3.12**	**3.46**	**3.12**	**3.13**
AAE_{6+}	**7.01**	**6.94**	**7.11**	**7.17**	**6.69**	**7.38**	**7.49**	**7.02**	**7.23**	**7.16**	**7.08**	**7.18**	**7.14**
AAE	**0.82**	**0.84**	1.06	1.02	**0.82**	**0.84**	0.85	0.91	0.96	0.97	**0.89**	0.96	0.96

5.2. RQ2: Are the performances of DNP models improved when applying data resampling and ensemble learning methods?

Methods: To answer the RQ2, we calculate the AAE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+}, Pred(0.3), $Pred(0.3)_0$, $Pred(0.3)_1$, $Pred(0.3)_2$, $Pred(0.3)_3$, $Pred(0.3)_{4-6}$, and $Pred(0.3)_{6+}$ for GBR, AdaBoost.R2+DTR, AdaBoost.R2+LR, AdaBoost.R2+BRR, Bagging +DTR, Bagging+LR, Bagging+BRR, SmoteR+DTR, SmoteR+LR, SmoteR +BRR, ROS+DTR, ROS+LR, and ROS+BRR. Figs. 6 and 8 show the distribution of the AEE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values with the Scott–Knott ESD test over all testing datasets. Figs. 7 and 9 show the distribution of the Pred(0.3), $Pred(0.3)_0$, $Pred(0.3)_1$, $Pred(0.3)_2$, $Pred(0.3)_3$, $Pred(0.3)_{4-6}$, and $Pred(0.3)_{6+}$ values with the Scott–Knott ESD test over all testing datasets. Table 6 lists the average AEE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values of the 13 regression algorithms over all testing datasets. Table 7 lists the average Pred(0.3), $Pred(0.3)_0$, $Pred(0.3)_1$, $Pred(0.3)_2$, $Pred(0.3)_3$, $Pred(0.3)_{4-6}$, and $Pred(0.3)_{6+}$ values of the 13 regression algorithms over all testing datasets. To ease the demonstration, we abbreviate AdaBoost.R2, Bagging, SmoteR to AR, BG, and SR in the figures and tables, respectively.



(a) Pred(0.3)$_0$ (b) Pred(0.3)$_1$ (c) Pred(0.3)$_2$ (d) Pred(0.3)$_3$ (e) Pred(0.3)$_{4\text{-}6}$ (f) Pred(0.3)$_{6+}$

Fig. 5. The boxplot of the Pred(0.3)$_0$, Pred(0.3)$_1$, Pred(0.3)$_2$, Pred(0.3)$_3$, Pred(0.3)$_{4-6}$, Pred(0.3)$_{6+}$ values of the 12 regression algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 7
The average Pred(0.3) values of the 13 regression algorithms on the testing datasets. (The bold value means the improved performance after applying data resampling and ensemble learning algorithms.)

Measures	GBR	AR+DTR	AR+LR	AR+BRR	BG+DTR	BG+LR	BG+BRR	SR+DTR	SR+LR	SR+BRR	ROS+DTR	ROS+LR	ROS+BRR
Pred(0.3)$_0$	0.80	0.81	0.46	0.44	0.79	**0.74**	**0.74**	0.82	0.49	0.46	0.83	0.50	0.47
Pred(0.3)$_1$	**0.30**	**0.26**	**0.41**	**0.43**	**0.34**	**0.35**	0.34	0.19	**0.52**	**0.54**	**0.21**	**0.52**	**0.52**
Pred(0.3)$_2$	0.07	**0.09**	**0.19**	**0.19**	**0.11**	0.14	0.12	**0.12**	**0.18**	**0.17**	**0.12**	**0.17**	**0.19**
Pred(0.3)$_3$	0.21	0.26	**0.29**	**0.33**	0.25	0.25	0.25	0.25	**0.29**	**0.32**	0.23	**0.31**	**0.33**
Pred(0.3)$_{4-6}$	0.12	**0.17**	**0.17**	**0.21**	**0.16**	**0.10**	**0.10**	**0.16**	**0.12**	**0.12**	0.12	**0.15**	**0.13**
Pred(0.3)$_{6+}$	0.19	0.20	**0.20**	**0.15**	0.12	**0.08**	0.08	**0.24**	**0.13**	**0.11**	**0.25**	**0.10**	**0.10**
Pred(0.3)	**0.55**	**0.55**	0.41	0.42	**0.55**	0.53	0.53	0.53	0.42	0.40	0.54	0.42	0.41

Results: From these tables and figures, we have the following observations:

(1) The median and average AAE values of the 13 regression algorithms are also very low, which are less than 1.07. GBR and Bagging+DTR achieve the lowest average AAE values (0.82). The Scott–Knott ESD test results show that there is no significant difference between DTR and GBR, Bagging+DTR, SmoteR+DTR, and ROS+DTR; AdaBoost.R2+LR, AdaBoost.R2+BRR, SmoteR+LR, SmoteR+BRR, ROS+LR, and ROS+BRR significantly perform worse than LR and BRR. GBR, AdaBoost.R2+DTR, and BG+DTR achieve the highest average Pred(0.3) value (0.55). The Scott–Knott ESD test results show applying AdaBoost.R2, SmoteR, and ROS significantly degrades the performance of LR and BRR in terms of Pred(0.3).

(2) The median and average AAE$_0$ values of GBR, AdaBoost.R2+DTR, Bagging+DTR, Bagging+LR, and Bagging+BRR, and ROS+DTR are low, which are less than or equal to 0.32. The Scott–Knott ESD test results show that GBR and AdaBoost.R2 can significantly improve the performance of DTR, and applying Bagging, SmoteR, and ROS cannot significantly improve the performance of DTR, LR, and BRR in terms of AAE$_0$. GBR and AdaBoost.R2+DTR achieve the highest average Pred(0.3)$_0$ values (0.8 and 0.81, respectively). The Scott–Knott ESD test results show that applying GBR, AdaBoost.R2, Bagging,

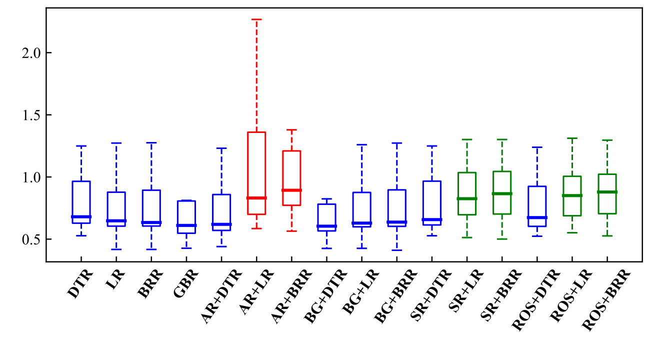


Fig. 6. The boxplot of the AAE values of the 16 regression algorithms.

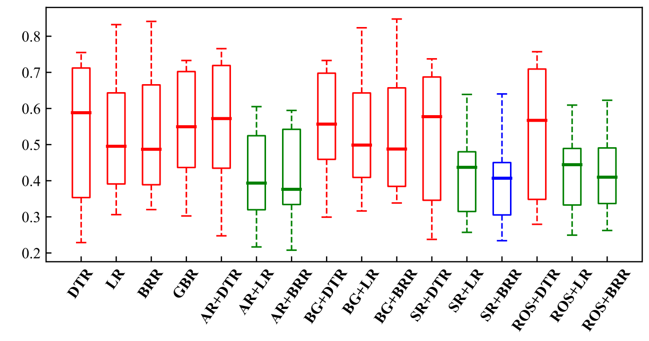


Fig. 7. The boxplot of the Pred(0.3) values of the 16 regression algorithms.

SmoteR, and ROS cannot significantly improve the performance of DTR, LR, and BRR in terms of $\text{Pred}(0.3)_0$.

(3) The Scott–Knott ESD test results show that SmoteR+LR, SmoteR+BRR, ROS+LR, and ROS+BRR significantly outperform other algorithms in terms of AAE_1, AAE_2, AAE_3, AAE_4, and AAE_{4-6}. SmoteR+BRR achieves the lowest average AAE_1, AAE_2, AAE_3, and AAE_{4-6} values, which are 0.56, 1.08, 1.79, and 3.12, respectively. SmoteR+LR achieves the lowest average AAE_2 value (1.08). Bagging+DTR achieves the lowest average AAE_{6+} value (6.69). However, after applying GBR, AdaBoost.R2, Bagging, SmoteR, and ROS to DTR, LR, and BRR, the average AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values of the algorithms are still very high.

(4) The Scott–Knott ESD test results show that AdaBoost.R2+LR, AdaBoost.R2+BRR, SmoteR+LR, SmoteR+BRR, ROS+LR, and ROS+BRR significantly perform better than other algorithms in terms of $\text{Pred}(0.3)_1$, $\text{Pred}(0.3)_2$, and $\text{Pred}(0.3)_3$; AdaBoost.R2+BRR significantly outperforms other algorithms in terms of $\text{Pred}(0.3)_{4-6}$; SmoteR+LR and SmoteR+BRR perform the best in term of $\text{Pred}(0.3)_{6+}$. SmoteR+BRR achieves the highest average $\text{Pred}(0.3)_1$ value (0.54), AdaBoost.R2+LR, AdaBoost.R2+BRR, and ROS+BRR achieve the highest average $\text{Pred}(0.3)_2$ value (0.19), AdaBoost.R2+BRR and ROS+BRR achieve the highest average $\text{Pred}(0.3)_3$ value (0.33), AdaBoost.R2+BRR achieves the highest average $\text{Pred}(0.3)_{4-6}$ value (0.21), and ROS+DTR achieves the highest average $\text{Pred}(0.3)_{6+}$ value (0.25).

(5) The overall results show that applying SmoteR and ROS to LR and BRR can make models not tend to predict the number of defects as zero. Therefore, the AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+}, $\text{Pred}(0.3)_1$, $\text{Pred}(0.3)_2$, $\text{Pred}(0.3)_3$, $\text{Pred}(0.3)_{4-6}$, and $\text{Pred}(0.3)_{6+}$ can be improved. But the algorithms still have very poor capability to predict the number of defects of defective modules accurately.

Answer to RQ2: AdaBoost.R2, SmoteR, ROS can improve the performance of LR and BRR in terms of some performance measures, but it is still difficult for the algorithms to predict the precise number of defects in defective modules accurately.

Table 8
The average AAE values of the 6 regression algorithms on the testing datasets. (The bold value means the improved performance after applying information gain and grid search methods.)

Measures	IG+DTR	IG+LR	IG+BRR	GS+DTR	GS+LR	GS+BRR
AAE_0	**0.25**	0.39	0.39	0.27	0.33	0.33
AAE_1	0.97	**0.60**	**0.59**	0.97	0.74	0.72
AAE_2	**1.60**	1.37	1.37	**1.63**	1.35	1.35
AAE_3	2.40	2.20	2.22	2.34	2.12	2.13
AAE_{4-6}	3.73	3.53	3.53	**3.48**	3.45	3.51
AAE_{6+}	7.51	8.53	8.54	7.75	7.65	**7.62**
AAE	0.91	0.87	0.87	0.91	0.85	0.85

5.3. RQ3: Are the performances of DNP models improved when applying feature selection and parameter optimization methods?

Methods: To answer the RQ3, we calculate the AAE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+}, Pred(0.3), $\text{Pred}(0.3)_0$, $\text{Pred}(0.3)_1$, $\text{Pred}(0.3)_2$, $\text{Pred}(0.3)_3$, $\text{Pred}(0.3)_{4-6}$, and $\text{Pred}(0.3)_{6+}$ for IG+DTR, IG+LR, IG+BRR, GS+DTR, GS+LR, and GS+BRR. Figs. 10 and 12 show the distribution of the AEE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values with the Scott–Knott ESD test over all testing datasets. Figs. 11 and 13 show the distribution of the Pred(0.3), $\text{Pred}(0.3)_0$, $\text{Pred}(0.3)_1$, $\text{Pred}(0.3)_2$, $\text{Pred}(0.3)_3$, $\text{Pred}(0.3)_{4-6}$, and $\text{Pred}(0.3)_{6+}$ values with the Scott–Knott ESD test over all testing datasets. Table 8 lists the average AEE, AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, and AAE_{6+} values of the 6 regression algorithms over all testing datasets. Table 9 lists the average Pred(0.3), $\text{Pred}(0.3)_0$, $\text{Pred}(0.3)_1$, $\text{Pred}(0.3)_2$, $\text{Pred}(0.3)_3$, $\text{Pred}(0.3)_{4-6}$, and $\text{Pred}(0.3)_{6+}$ values of the 6 regression algorithms over all testing datasets.

Results: From these tables and figures, we have the following observations:

(1) The 6 regression algorithms also achieve the very low AAE values (less than 1). GS+LR and GS+BRR achieve the lowest average AAE values (0.85), and GS+DTR achieves the highest average Pred(0.3) value (0.54). The Scott–Knott ESD test results show that there is no significant difference between DTR, LR, BRR, IG+DTR, IG+LR, IG+BRR, GS+DTR, GS+LR, and GS+BRR in terms of AAE and Pred(0.3). In other words, applying IG and GS cannot improve the performance of DTR, LR, and BRR in terms of AAE and Pred(0.3).

(2) IG+LR and IG+BRR achieve the lowest average AAE_1 values (0.6 and 0.59, respectively) and the highest average $\text{Pred}(0.3)_1$ values (0.44). The Scott–Knott ESD test results show that applying information gain feature selection method can significantly improve the performance of LR and BRR in terms of AAE_1 and $\text{Pred}(0.3)_1$;

(3) Applying the grid search parameter optimization method can improve the performance of DTR in terms of AAE_2 and $\text{Pred}(0.3)_2$. However, the Scott–Knott ESD test results show IG+DTR only significantly outperforms DTR in terms of $\text{Pred}(0.3)_2$. In addition, applying information gain and grid search methods cannot significantly improve or even degrades the performance of DTR, LR, and BRR in terms of other evaluation metrics.

Answer to RQ3: The information gain feature selection method and grid search parameter optimization method cannot improve the performance of DTR, LR, and BRR in terms of most evaluation metrics.

6. Discussion

6.1. Implication

In this subsection, we analyze implications based on our experimental results and provide a few suggestions to practitioners and researchers in this domain.

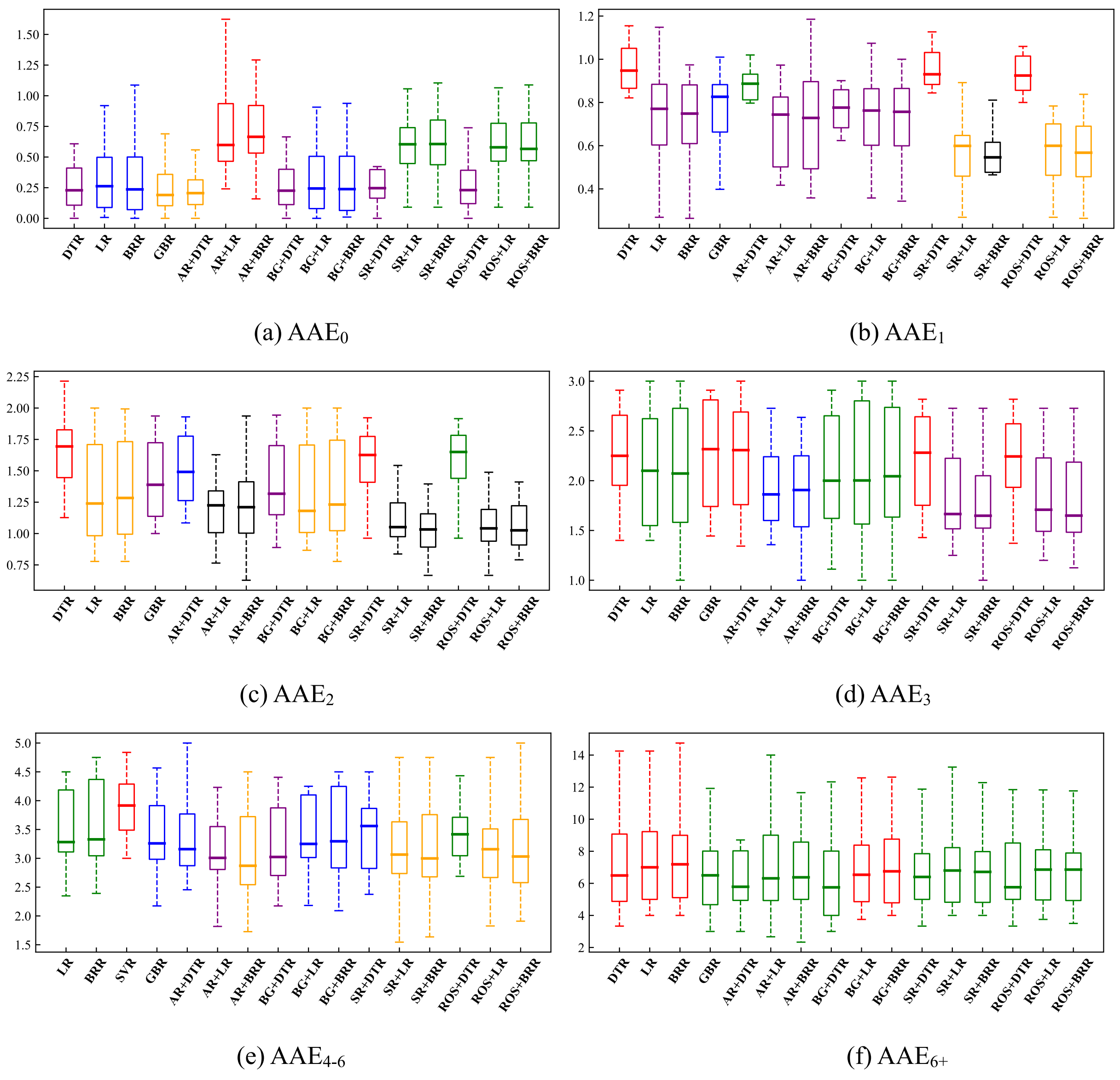


Fig. 8. The boxplot of the AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+} values of the 16 regression algorithms.

Table 9
The average Pred(0.3) values of the 6 regression algorithms on the testing datasets. (The bold value means the improved performance after applying information gain and grid search methods.)

Measures	IG+DTR	IG+LR	IG+BRR	GS+DTR	GS+LR	GS+BRR
$Pred(0.3)_0$	0.81	0.65	0.65	0.83	0.73	0.73
$Pred(0.3)_1$	**0.23**	**0.44**	**0.44**	0.18	0.34	0.35
$Pred(0.3)_2$	**0.10**	0.09	0.10	**0.10**	0.14	0.13
$Pred(0.3)_3$	0.18	0.14	0.14	0.21	0.25	0.25
$Pred(0.3)_{4-6}$	0.08	0.08	0.08	0.14	0.09	0.09
$Pred(0.3)_{6+}$	0.19	0.03	0.03	0.14	0.07	0.08
Pred(0.3)	0.53	0.51	0.50	0.54	0.53	0.53

(1) When the purpose of DNP models is to predict the precise number of defects, researchers should evaluate the AAE and Pred(0.3) values for the modules with different numbers of defects separately, because the number of defects is imbalanced and the non-defective modules occupy a large portion of the whole defect dataset. Therefore, some DNP studies in Table 1 that employ the whole AAE and Pred(0.3) values of all modules as the evaluation metrics for the proposed DNP algorithms should be revisited.

(2) The problem of predicting the precise number of defects through the adoption of regression algorithms is far from being solved, and therefore the research community has the opportunity to make more effort in proposing more accurate DNP algorithms that better assist practitioners in practice. Since it is very difficult to accurately predict the precise number of defects in defective modules at present, practitioners should predict the number of defects by the regression algorithms and use the predicted number to sort software modules, then employ some ranking performance measures to evaluate the prediction model, e.g., FPA and Kendall. Yang [41] and Yu [49] find that BRR performs the best in terms of FPA, so we recommend that practitioners employ BRR to rank software modules to allocate scarce testing resources more efficiently.

(3) If the main purpose of DNP models is to rank software modules based on the predicted number of defects, the title of the paper should ideally be “Ranking-Oriented Defect Prediction” rather than “Predict the number of defects/bugs” to avoid ambiguity. In addition, researchers can improve the performance of regression algorithms for ranking software modules accurately by modifying their loss function, because the loss function of regression algorithms is generally to minimize the total difference between the actual and predicted number

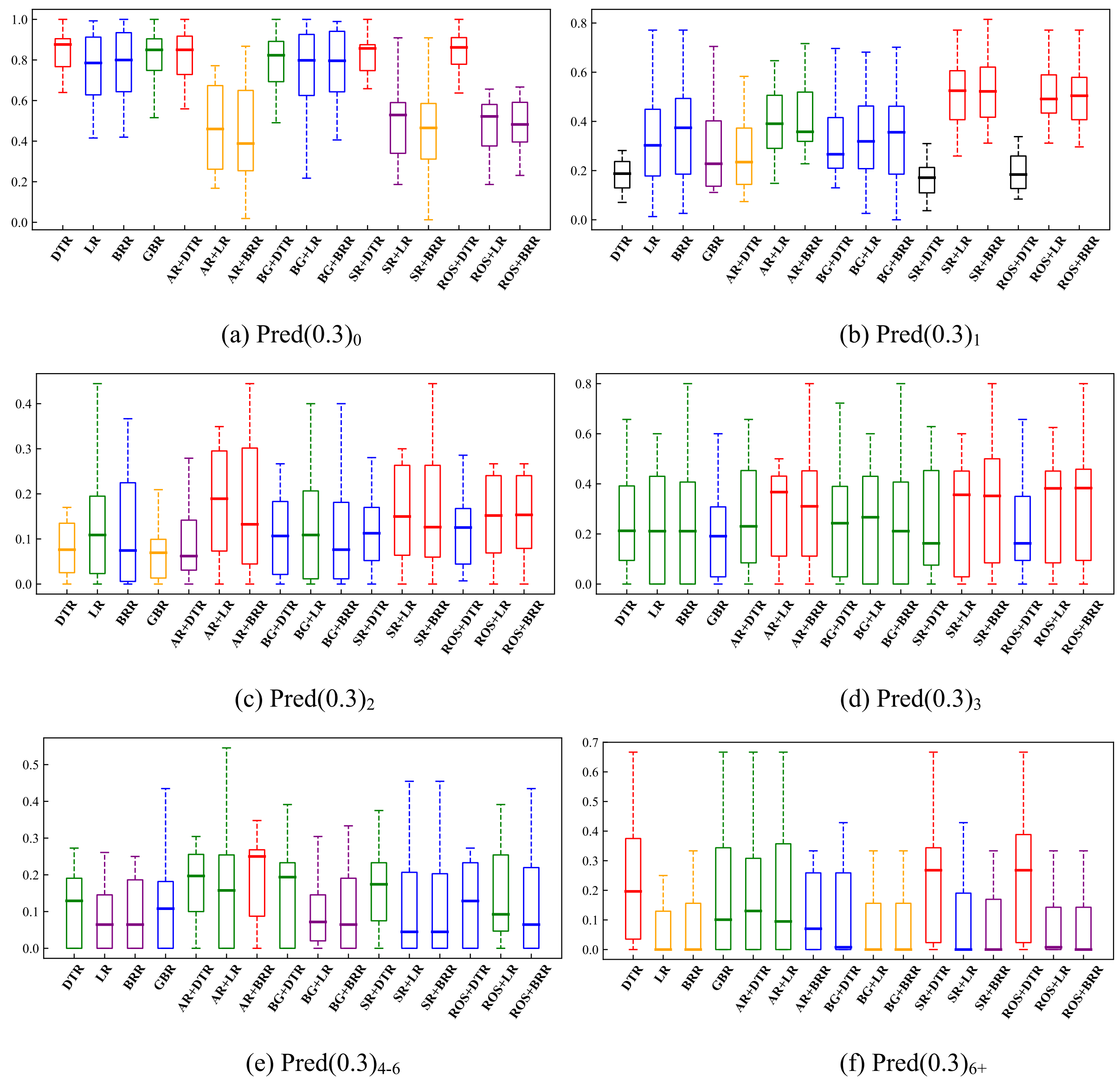


Fig. 9. The boxplot of the Pred(0.3)$_0$, Pred(0.3)$_1$, Pred(0.3)$_2$, Pred(0.3)$_3$, Pred(0.3)$_{4-6}$, Pred(0.3)$_{6+}$ values of the 16 regression algorithms.

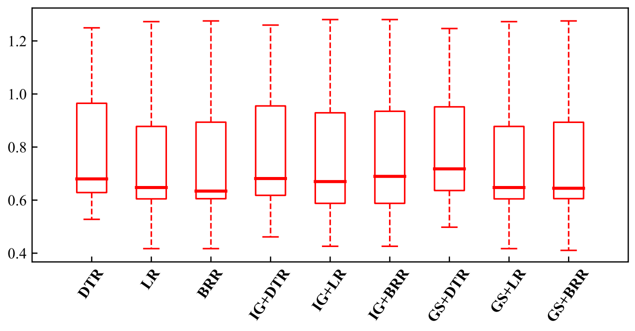


Fig. 10. The boxplot of the AAE values of the 9 regression algorithms.

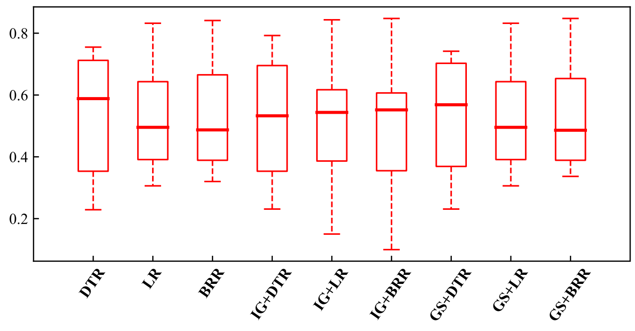


Fig. 11. The boxplot of the Pred(0.3) values of the 9 regression algorithms.

of defects rather than optimize the ranking performance measures (e.g., FPA and PofB). A good model with the higher accuracy may generate a worse ranking of software modules. For example, there are the three modules in the testing datasets, i.e., M$_1$, M$_2$, and M$_3$, which have 6, 5, and 4 defects, respectively. The DNP model A predicts the numbers of defects in M$_1$, M$_2$, and M$_3$ are 6, 3, and 4, whereas the DNP model B predicts the numbers of defects in the three modules are 3, 2, and 1. Although the model A has the lower AAE and higher Pred(0.3) values, the model B predicts the correct ranking of the three modules.

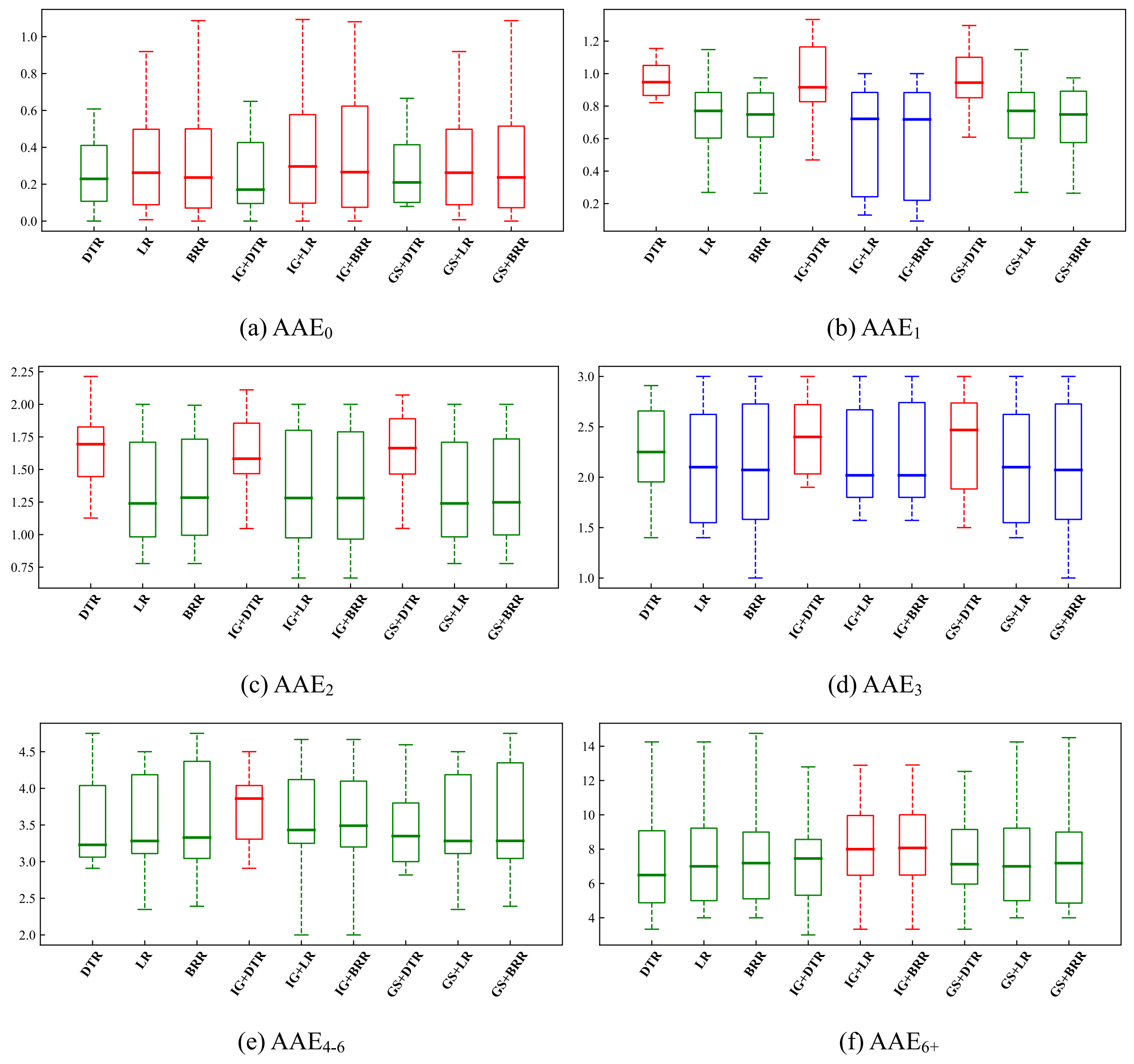


Fig. 12. The boxplot of the AAE_0, AAE_1, AAE_2, AAE_3, AAE_{4-6}, AAE_{6+} values of the 9 regression algorithms.

6.2. Threats of validity

(1) We use the defect datasets from the PROMISE repository, because the PROMISE datasets contain the information of the number of defects and have been widely used in many DNP studies. However, we still cannot claim that our conclusion can be generalized to other datasets, especially the proprietary datasets.

(2) We investigate the performance of all base algorithms investigated in the 23 previous DNP studies, except BugStates, Multilayer Perceptron Regression (MPR), Sample entropy–Support Vector Regression (SSVR), Auto-Regressive Integrated Moving Average (ARIMA) model, X12-ARIMA model, Deep Learning Neural Network (DPNN), and Fuzzy Support Vector Regression (FSVR). We only investigate the performance of SmoteR, ROS, GBR, AdaBoost.R2, and Bagging, which are widely investigated data imbalance learning algorithms in previous DNP studies. Additionally, we acknowledge the existence of several other regression and data imbalance learning algorithms. Our study employs the 17 algorithms which is sufficient for an empirical study. Adoption of other algorithms not used in this study is left for a future study.

(3) In this study, we sort software modules based on the number of defects. It assumes that the testing effort is only related to the inspected number of defects, and ignores that different software modules require different testing effort due to different module size (i.e., LOC). Some researchers [77–81] proposed to rank software modules based on the defect density, and assumed that the inspection effort is only linearly associated with LOC. However, Huang et al. [82] have pointed out that the frequent context switch between different software modules also increase the testing effort, if a software defect prediction model requires software developers to inspect many software modules. In other words, even two software developers inspect the same LOC and find the same number of defects, if one developer needs to inspect more software modules than the other one, he will cost the more testing effort. In addition, Ostrand et al. [83] have claimed that "discussions with software testers convinced us that it is often more valuable for testers to know which files had the largest numbers of faults rather than the largest fault densities since that would allow them to better identify most of the faults quickly". Therefore, we ignore the LOC of software modules in this study. In the future work, we will investigate which factors are related to the testing effort and how to determine the weights of different factors.

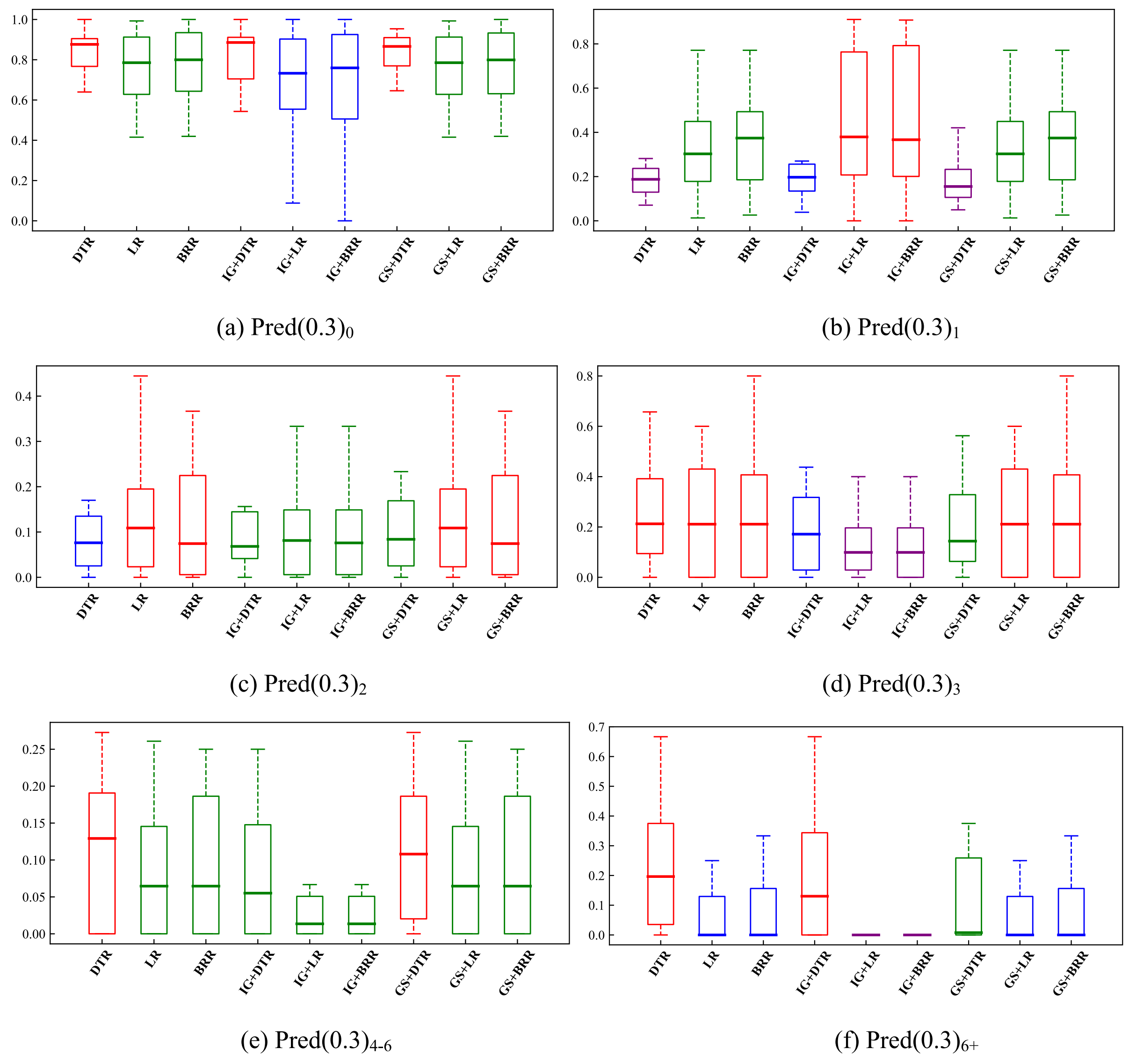


Fig. 13. The boxplot of the Pred(0.3)$_0$, Pred(0.3)$_1$, Pred(0.3)$_2$, Pred(0.3)$_3$, Pred(0.3)$_{4-6}$, Pred(0.3)$_{6+}$ values of the 9 regression algorithms.

7. Conclusion

For practical benefits, predicting the number of defects has been much suggested in literature than classification-based defect prediction. A number of regression algorithms have been used for building DNP models and shown to achieve the good performance in terms of AAE and Pred(0.3). However, the good performance is questionable due to the imbalanced distribution of the number of defects. Therefore, in this paper, we revisit the impact of 12 widely-used regression algorithms, two data resampling algorithms, three ensemble learning algorithms, one feature selection method, and one parameter optimization method for predicting the precise number of defects on the 18 PROMISE datasets. We propose to calculate the AAE and Pred(0.3) values for the modules with different numbers of defects separately rather than the whole AAE and Pred(0.3) values for all modules. The experimental results show that it is still difficult for the algorithms to predict the precise number of defects of defective modules accurately. Therefore, we recommend that the regression algorithms can be only used for ranking software modules based on the predicted number of defects rather than predicting the precise number of defects, and some DNP studies employing regression performance measures to evaluate the proposed method should be revisited.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported in part by the Natural Science Foundation of Chongqing City (No. cstc2021jcyj-msxmX1115), the Fundamental Research Funds for the Central Universities (WUT: 213110003), the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017) and the research funds of City University of Hong Kong (No. 7005028 and 7005217), and the Research Support Fund by Intel (No. 9220097), and funding supports from other industry partners (No. 9678149, 9440227, 9229029, 9440180, and 9220103). We thank Juan Li and Guoping Nie for their helpful comments about this work, Chenkai Shen, Weiqiang Peng, Peixin Yang, and Kuan Zou for helping us solve many coding problems.

References

[1] Z. Sun, J. Li, H. Sun, L. He, CFPS: Collaborative filtering based source projects selection for cross-project defect prediction, Appl. Soft Comput. 99 (2021) 106940.

[2] H. Tong, B. Liu, S. Wang, Kernel spectral embedding transfer ensemble for heterogeneous defect prediction, IEEE Trans. Softw. Eng. 47 (9) (2021) 1886–1906.

[3] X. Chen, Y. Mu, K. Liu, Z. Cui, C. Ni, Revisiting heterogeneous defect prediction methods: How far are we? Inf. Softw. Technol. 130 (2021) 106441.

[4] N. Li, M. Shepperd, Y. Guo, A systematic review of unsupervised learning techniques for software defect prediction, Inf. Softw. Technol. 122 (2020) 106287.

[5] Q. Zou, L. Lu, Z. Yang, X. Gu, S. Qiu, Joint feature representation learning and progressive distribution matching for cross-project defect prediction, Inf. Softw. Technol. 137 (2021) 106588.

[6] T. Chakraborty, A.K. Chakraborty, Hellinger net: A hybrid imbalance learning model to improve software defect prediction, IEEE Trans. Reliab. 70 (2) (2021) 481–494.

[7] J. Yao, M.J. Shepperd, The impact of using biased performance metrics on software defect prediction research, Inf. Softw. Technol. 139 (2021) 106664.

[8] J. Xu, F. Wang, J. Ai, Defect prediction with semantics and context features of codes based on graph representation learning, IEEE Trans. Reliab. 70 (2) (2021) 613–625.

[9] K. Zhao, Z. Xu, T. Zhang, Y. Tang, M. Yan, Simplified deep forest model based just-in-time defect prediction for android mobile apps, IEEE Trans. Reliab. 70 (2) (2021) 848–859.

[10] X. Yu, J. Liu, Z. Yang, X. Liu, The Bayesian network based program dependence graph and its application to fault localization, Inf. Softw. Technol. 134 (2017) 44–53.

[11] X. Xiao, Y. Pan, B. Zhang, G. Hu, Q. Li, R. Lu, ALBFL: A novel neural ranking model for software fault localization via combining static and dynamic features, Inf. Softw. Technol. 139 (2021) 106653.

[12] X. Yan, B. Liu, S. Wang, D. An, F. Zhu, Y. Yang, Efilter: An effective fault localization based on information entropy with unlabelled test cases, Inf. Softw. Technol. 134 (2021) 106543.

[13] D. Ghosh, J. Singh, Spectrum-based multi-fault localization using chaotic genetic algorithm, Inf. Softw. Technol. 133 (2021) 106512.

[14] W. Zhang, Y. Du, T. Yoshida, Q. Wang, X. Li, SamEn-SVR: using sample entropy and support vector regression for bug number prediction, IET Softw. 12 (3) (2018) 183–189.

[15] J. Wang, H. Zhang, Predicting defect numbers based on defect state transition models, in: Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, 2012, pp. 191–200.

[16] S.S. Rathore, S. Kumar, Homogeneous ensemble methods for the prediction of number of faults, in: Fault Prediction Modeling for the Prediction of Number of Software Faults, Springer, 2019, pp. 31–45.

[17] L. Qiao, X. Li, Q. Umer, P. Guo, Deep learning based software defect prediction, Neurocomputing 385 (2020) 100–110.

[18] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE Trans. Softw. Eng. 31 (4) (2005) 340–355.

[19] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, Identification of defect-prone classes in telecommunication software systems using design metrics, Inform. Sci. 176 (24) (2006) 3711–3734.

[20] K. Gao, T.M. Khoshgoftaar, A comprehensive empirical study of count models for software fault prediction, IEEE Trans. Reliab. 56 (2) (2007) 223–236.

[21] W. Afzal, R. Torkar, R. Feldt, Prediction of fault count data using genetic programming, in: 2008 IEEE International Multitopic Conference, IEEE, 2008, pp. 349–356.

[22] L. Yu, Using Negative Binomial Regression Analysis to Predict Software Faults: A Study of Apache Ant, Citeseer, 2012.

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

[24] S.S. Rathore, S. Kumar, Predicting number of faults in software system using genetic programming, in: SCSE, 2015, pp. 303–311.

[25] M. Chen, Y. Ma, An empirical study on predicting defect numbers, in: SEKE, 2015, pp. 397–402.

[26] S.S. Rathore, S. Kumar, A decision tree regression based approach for the number of software faults prediction, ACM SIGSOFT Softw. Eng. Notes 41 (1) (2016) 1–6.

[27] S.S. Rathore, S. Kumar, Ensemble methods for the prediction of number of faults: a study on eclipse project, in: 2016 11th International Conference on Industrial and Information Systems (ICIIS), IEEE, 2016, pp. 540–545.

[28] S.S. Rathore, S. Kumar, An empirical study of some software fault prediction techniques for the number of faults prediction, Soft Comput. 21 (24) (2017) 7417–7434.

[29] S.S. Rathore, S. Kumar, Towards an ensemble based system for predicting the number of software faults, Expert Syst. Appl. 82 (2017) 357–382.

[30] S.S. Rathore, S. Kumar, Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems, Knowl.-Based Syst. 119 (2017) 232–256.

[31] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, S. Ye, Learning from imbalanced data for predicting the number of software defects, in: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2017, pp. 78–89.

[32] M. Wu, S. Ye, C. Li, Z. Ma, Z. Fu, Revisting the impact of regression models for predicting the number of defects, in: SEKE, 2018, 427–426.

[33] S.S. Rathore, S. Kumar, An approach for the prediction of number of software faults based on the dynamic selection of learning techniques, IEEE Trans. Reliab. 68 (1) (2019) 216–236.

[34] X. Chen, D. Zhang, Y. Zhao, Z. Cui, C. Ni, Software defect number prediction: Unsupervised vs supervised methods, Inf. Softw. Technol. 106 (2019) 161–181.

[35] Q. Huang, C. Ni, X. Chen, Q. Gu, K. Cao, Multi-project regression based approach for software defect number prediction, in: SEKE, 2019, pp. 425–546.

[36] M. Nevendra, P. Singh, Software bug count prediction via AdaBoost. R-ET, in: 2019 IEEE 9th International Conference on Advanced Computing (IACC), IEEE, 2019, pp. 7–12.

[37] P.R. Bal, S. Kumar, WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction, IEEE Trans. Reliab. 69 (4) (2020) 1355–1375.

[38] H. Tong, W. Lu, W. Xing, B. Liu, S. Wang, SHSE: A subspace hybrid sampling ensemble method for software defect number prediction, Inf. Softw. Technol. (2021) 106747.

[39] M.R. Lyu, Software reliability engineering: A roadmap, in: L.C. Briand, A.L. Wolf (Eds.), International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA, IEEE Computer Society, 2007, pp. 153–170.

[40] H. Zhang, An initial study of the growth of eclipse defects, in: A.E. Hassan, M. Lanza, M.W. Godfrey (Eds.), Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-Located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings, ACM, 2008, pp. 141–144.

[41] X. Yang, W. Wen, Ridge and lasso regression models for cross-version defect prediction, IEEE Trans. Reliab. 67 (3) (2018) 885–896.

[42] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, IEEE Trans. Softw. Eng. 46 (11) (2020) 1200–1219.

[43] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, B. Xu, How far we have progressed in the journey? an examination of cross-project defect prediction, ACM Trans. Softw. Eng. Methodol. (TOSEM) 27 (1) (2018) 1.

[44] S.S. Rathore, S. Kumar, Techniques used for the prediction of number of faults, in: Fault Prediction Modeling for the Prediction of Number of Software Faults, Springer, 2019, pp. 11–29.

[45] G. You, F. Wang, Y. Ma, An empirical study of ranking-oriented cross-project software defect prediction, Int. J. Softw. Eng. Knowl. Eng. 26 (09n10) (2016) 1511–1538.

[46] X. Yu, J. Liu, J.W. Keung, Q. Li, K.E. Bennin, Z. Xu, J. Wang, X. Cui, Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM, IEEE Trans. Reliab. 69 (1) (2019) 139–153.

[47] T.T. Nguyen, T.Q. An, V.T. Hai, T.M. Phuong, Similarity-based and rank-based defect prediction, in: 2014 International Conference on Advanced Technologies for Communications (ATC 2014), IEEE, 2014, pp. 321–325.

[48] X. Yang, K. Tang, X. Yao, A learning-to-rank approach to software defect prediction, IEEE Trans. Reliab. 64 (1) (2014) 234–246.

[49] X. Yu, K.E. Bennin, J. Liu, J.W. Keung, X. Yin, Z. Xu, An empirical study of learning to rank techniques for effort-aware defect prediction, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2019, pp. 298–309.

[50] M.J. Hayat, M. Higgins, Understanding poisson regression, J. Nurs. Educ. 53 (4) (2014) 207–215.

[51] D. Lambert, Zero-inflated Poisson regression, with an application to defects in manufacturing, Technometrics 34 (1) (1992) 1–14.

[52] J.M. Hilbe, Negative Binomial Regression, Cambridge University Press, 2011.

[53] M. Minami, C.E. Lennert-Cody, W. Gao, M. Román-Verdesoto, Modeling shark bycatch: the zero-inflated negative binomial regression model with smoothing, Fish. Res. 84 (2) (2007) 210–221.

[54] S.E. Saffari, R. Adnan, W. Greene, Investigating the impact of excess zeros on hurdle-generalized Poisson regression model with right censored count data, Stat. Neerl. 67 (1) (2013) 67–80.

[55] J.R. Koza, Genetic Programming, Citeseer, 1997.

[56] D.F. Specht, A general regression neural network, IEEE Trans. Neural Netw. 2 (6) (1991) 568–576.

[57] M. Xu, P. Watanachaturaporn, P.K. Varshney, M.K. Arora, Decision tree regression for soft classification of remote sensing data, Remote Sens. Environ. 97 (3) (2005) 322–336.

[58] G.A. Seber, A.J. Lee, Linear Regression Analysis, Vol. 329, John Wiley & Sons, 2012.

[59] Q. Shi, M. Abdel-Aty, J. Lee, A Bayesian ridge regression analysis of congestion's impact on urban expressway safety, Accid. Anal. Prev. 88 (2016) 124–137.

[60] H. Drucker, C.J. Burges, L. Kaufman, A.J. Smola, V. Vapnik, Support vector regression machines, in: Advances in Neural Information Processing Systems, 1997, pp. 155–161.

[61] M. Maltamo, A. Kangas, Methods based on k-nearest neighbor regression in the prediction of basal area diameter distribution, Can. J. Forest Res. 28 (8) (1998) 1107–1115.

[62] L. Torgo, P. Branco, R.P. Ribeiro, B. Pfahringer, Resampling strategies for regression, Expert Syst. J. Knowl. Eng. 32 (3) (2015) 465–476.

[63] H. Drucker, Improving regressors using boosting techniques, in: D.H. Fisher (Ed.), Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997, Morgan Kaufmann, 1997, pp. 107–115.

[64] D. Steinberg, M. Golovnya, N.S. Cardell, Stochastic gradient boosting: An introduction to TreeNet^{TM}, in: S.J. Simoff, G.J. Williams, M. Hegland (Eds.), The 15th Australian Joint Conference on Artificial Intelligence 2002, Proceedings Australasian Data Mining Workshop, Canberra, Australia, 2002, pp. 1–12.

[65] Q. Sun, B. Pfahringer, Bagging ensemble selection for regression, in: M. Thielscher, D. Zhang (Eds.), AI 2012: Advances in Artificial Intelligence - 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7691, Springer, 2012, pp. 695–706.

[66] Z. Xu, J. Liu, Z. Yang, G. An, X. Jia, The impact of feature selection on defect prediction performance: An empirical comparison, in: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2016, pp. 309–320.

[67] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2006) 2–13.

[68] T.M. Khoshgoftaar, K. Gao, A. Napolitano, An empirical study of feature ranking techniques for software quality prediction, Int. J. Softw. Eng. Knowl. Eng. 22 (02) (2012) 161–183.

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

[70] T.M. Khoshgoftaar, M. Golawala, J. Van Hulse, An empirical study of learning from imbalanced data using random forest, in: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), Vol. 2, IEEE, 2007, pp. 310–317.

[71] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: Some comments on the nasa software defect datasets, IEEE Trans. Softw. Eng. 39 (9) (2013) 1208–1215.

[72] Q. Song, Y. Guo, M. Shepperd, A comprehensive investigation of the role of imbalanced learning for software defect prediction, IEEE Trans. Softw. Eng. PP (99) (2017) 1.

[73] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, IEEE Trans. Softw. Eng. 39 (6) (2012) 757–773.

[74] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, 2012.

[75] C. Tantithamthavorn, ScottKnottESD: The scott-knott effect size difference (ESD) test, 2016, R Package Version 2.

[76] B. Ghotra, S. McIntosh, A.E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: Proceedings of the 37th International Conference on Software Engineering-Vol. 1, IEEE Press, 2015, pp. 789–800.

[77] T. Mende, R. Koschke, Effort-aware defect prediction models, in: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 107–116.

[78] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, A.E. Hassan, Revisiting common bug prediction findings using effort-aware models, in: 2010 IEEE International Conference on Software Maintenance, IEEE, 2010, pp. 1–10.

[79] W. Ma, L. Chen, Y. Yang, Y. Zhou, B. Xu, Empirical analysis of network measures for effort-aware fault-proneness prediction, Inf. Softw. Technol. 69 (2016) 50–70.

[80] Y. Yang, M. Harman, J. Krinke, S. Islam, D. Binkley, Y. Zhou, B. Xu, An empirical study on dependence clusters for effort-aware fault-proneness prediction, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 296–307.

[81] Y. Qu, J. Chi, H. Yin, Leveraging developer information for efficient effort-aware bug prediction, Inf. Softw. Technol. 137 (2021) 106605.

[82] Q. Huang, X. Xia, D. Lo, Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction, Empir. Softw. Eng. 24 (5) (2019) 2823–2862.

[83] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE Trans. Softw. Eng. 31 (4) (2005) 340–355.