Software Defect Prediction Method Based on Cost-Sensitive Random Forest

Wei-Dong Zhao, Sheng-Dong Zhang, Ming Wang

Software Defect Prediction Method Based on Cost-Sensitive Random Forest

Wei-Dong Zhao[1], Sheng-Dong Zhang[1] and Ming Wang[2*]

[1] Shandong University of Science and Technology, Qingdao 266590, China
[2] Qingdao Customs Technology Center, Qingdao 266590, China
zwdslj@163.com
zsdsuccess@163.com
wangm415@163.com

Abstract. In this paper, a new method was proposed to reduce the misclassification cost of software defect prediction under the condition of imbalanced classes. The effectiveness of the method was evaluated by the correct rate of sample classification, precision rate, recall rate, and F-Measure index. The main results of this research were as follows: (1) The proposed method can maintain a high accuracy rate while maintaining a relatively low cost of misclassification; (2) In the data preprocessing stage, a median assignment method is proposed, Used to deal with the field missing value problem of a reasonable sample in the data set; (3) In the classification stage of the decision tree and the voting classification stage of the formed random forest, the cost-sensitive factors defined according to different objects are introduced respectively, and the training is based on the cost-sensitive The improved random forest model. Experimental results show that this method can reduce the cost of misclassification while maintaining a high accuracy rate.

Keywords: Software Defect Prediction, Class Imbalance, Cost-sensitive, Random Forest.

1 Introduction

With the rapid change and development of software projects, software development technology continues to improve and improve. However, in the development process of the entire software project, due to the increasing scale of the software and the increasing complexity of internal logic, it leads to it becomes more difficult to manually judge software defects. If these problems cannot be discovered in time, it will increase the cost of software development, cause waste of human and material resources. Software defect prediction technology is an effective means to find defects in software projects. Therefore, the research of software defect prediction technology is of great significance.

Software defect prediction (SDP) has always been one of the most active areas in software engineering research. According to the different granularity of prediction, it mainly includes module-level, file-level, and change-level defect prediction [1]. This article is research on defect prediction technology for module-level software entities. This technology uses the existing software defect public data set to train a defect prediction model to predict whether there are defects in the software modules.

In the early stage of the development of software defect prediction technology, Nagappan N, Compton BT, T.Menzies and others generated defect prediction models based on statistical methods [2][3][4]. However, due to the serious class imbalance and high feature dimension in the acquired data set, the performance of the model based on statistics may not always achieve the expected effect. F. Xing, Kechao Wang et al. used support vector machine (SVM) [5][6], Haijin Ji et al. used Naive Bayes (NB) method [7], K. Ayan, J. Zheng used neural network [8][9], T.M. Khoshgoftaar used decision tree [10], Zhou L, Sun Z et al. used ensemble learning [11][12] to build a software defect prediction model, although they improved the prediction accuracy of unknown category modules, however, most of these constructed classifiers did not consider the impact of the serious class imbalance in the data set, which increased the cost of misclassification. In addition, Michael J, M. Liu, Ling Xu and others used cost-sensitive learning [13][14][15] to create software defect prediction models. Although it reduced the cost of misclassification, it also reduced the prediction accuracy. However, in the field of software defect prediction (SDP), the construction of classifiers is often aimed at minimizing the classification cost, which is the cost associated with the classification [13].

To construct a software defect prediction model that can minimize the cost of misclassification while ensuring high prediction accuracy, this paper proposes a software defect prediction method based on cost-sensitive and improved random forest. This method mainly improves the forecasting work in three aspects. First, in the data preprocessing stage, for the field missing value problem of a reasonable sample in the data set, use the median assignment processing of the column where the missing value is located; second, divide the sample in the ID3 tree when classifying, introduce cost-sensitive factors, and use the best model discriminant index that is customized according to the error rate, recall rate, and precision rate indicators to filter the generated ID3 tree; third, in the constructed random forest, the performance and number of the selected ID3 trees are secondarily limited, and in the stage of voting and classifica-

tion, the cost-sensitive factor is introduced again. Overall, reduce the impact on prediction accuracy and increase the tendency of predictions to be defective.

2 Related Work

2.1 Classification Error Type

According to current research, there are two types of errors in the environment of software defect prediction (SDP) [15]. When the classification model predicts a non-defective module as a defective module, Type I misclassification occurs. Similarly, when a defective module is incorrectly classified as non-defective, Type II misclassification occurs. In practical applications, Type II will cause more serious prediction errors, because software defects have not been discovered, causing more serious damage to the software after it is put into use. Therefore, the cost of Type II is much higher than that of Type I. Therefore, to avoid only Type II misclassification, the frequency of Type I misclassification can be appropriately increased. This cost conversion is considered worthwhile in the field of software defect prediction and can effectively reduce the cost of misclassification [14].

2.2 ID3 Tree

The ID3 algorithm [16] proposed by Quinlan J R has a simple structure, strong learning ability, and is easy to understand. It is a classic decision tree algorithm. The algorithm uses information gain as the basis for division and selects the attribute with the largest information gain as the split node to generate a decision tree. At present, there are many optimizations on the ID3 algorithm [17][18][19].

The ID3 algorithm selects the current best feature to segment the data set each time. There are two ways of segmentation: First, segment according to all possible values of the feature; Secondly, use the binary segmentation method to divide the data set into two parts each time.

In software defect prediction, ID3 trees based on binary segmentation are mostly used. Generally, two aspects need to be considered, on the one hand, select the best segmentation attribute and attribute value. First, calculate the expected information. The expected information indicates the degree of fluctuation of the attribute's influence on the result. The smaller the expected information, the smaller the fluctuation, and the greater the information gain, assuming that there are k categories in the data set D, and the occupancy rate of the i-th category in the total data is pi, the calculation formula for the expected information is as (1); secondly, find the information entropy of each attribute and attribute value. It is a measure of the uncertainty of random variables. The greater the uncertainty, the greater the entropy. Assuming that feature A has m categories or values, and Dj is a subset of D, the information entropy is calculated as formula (2); then, solve for information gain according to formula (3), which is determined by the expected information and information of the attribute Entropy is jointly determined, and it is also the criterion for selecting partition attributes, generally, the attribute and attribute value with the largest information gain is selected as

the split node. Finally, under this attribute, put the value less than or equal to this attribute in the left subtree, and vice versa put it in the right subtree.

On the other hand, set the conditions for the ID3 tree to stop splitting. In the process of constructing the ID3 tree, to reduce the impact of model overfitting and the complexity of tree growth, the minimum number of nodes and a single type of child nodes are used as general conditions for the ID3 tree nodes to stop splitting. When the data volume of the node is less than a specified amount or when the data in the node is all of the same types, the split will not continue. Finally, complete the construction of the ID3 tree.

$$Info(D) = -\sum_{i=1}^{k} pi \ log_2 pi \tag{1}$$

$$Ent(D, A) = -\sum_{j=1}^{m} Info(Dj) \tag{2}$$

$$Gain(A) = Info(D) - Ent(D, A) \tag{3}$$

2.3 Random Forest

The random forest algorithm proposed by Breiman [20] is an extended variant of Bagging, which is composed of multiple independent decision trees. When judging the category of a new sample, each tree in the random forest will judge the classification result of the sample, and finally vote for judgment. The category with more votes is the final result of the prediction. The formula is as (4):

$$H(x) = arg \max_{j} \sum_{i=1}^{T} h_j^i(x) \tag{4}$$

Specific experimental steps: assuming that the original data set has n samples, use data disturbance sampling to form the training set; from the d attributes of the sample, use attribute perturbation to randomly select a subset containing k (k < d) attributes, and use the attribute subset to construct a decision tree model; input the sample, each tree gets its result, vote on the result, and choose the classification result of the sample with more votes.

Owing to the random forest [21] being simple to implement and easy to understand, it has good advantages in processing high-dimensional data, detection feature importance, anti-noise ability, and classification accuracy, making this model a widely used software defect prediction method. However, because the data set used for software defect prediction is usually unbalanced, the random forest has defects in the prediction and classification process.

3 Method Of This Article

3.1 Method Flow

Figure 1 is a flowchart of the proposed software defect prediction model. The process of the method is mainly divided into two parts. The first part builds a prediction model, including data preprocessing and training cost-sensitive ID3 trees. The second part

is the prediction algorithm, a cost-sensitive random forest is constructed from the trained ID3 tree, for each test module entered, the proposed prediction algorithm divides it into defective or non-defective. The specific process is as follows:

(1) Data preprocessing

Step 1. For the software defect prediction data set Data, delete the duplicate and contradictory data; use the median of the missing value column to fill in the missing values of a reasonable piece of data to obtain the preprocessed data set Data';

Step 2. Use the minimum-maximum normalization method to normalize Data′, and map the value of each attribute to the interval [0,1] to obtain Data";

Step 3. According to the size of Data", randomly divide it into a training set TrainD and test TestD according to a custom ratio.

(2) Training cost-sensitive ID3 tree

Step 1. Use Bagging technology to perturb the data of TrainD, randomly extract samples from TrainD with replacement, use the extracted samples as the training set TrainD_train of the training ID3 tree, and use the remaining samples as the test set TrainD_test in the training set;

Step 2. Introduce attribute disturbance, perform attribute disturbance on TrainD_train, and generate TrainD_trainM after attribute disturbance;

Step 3. Use TrainD_trainM, adopt the binary segmentation method, introduce the cost-sensitive factor Acost, and build a cost-sensitive ID3 tree;

Step 4. Use the constructed ID3 tree to predict the samples in TrainD_test and determine whether each sample is a defective module;

Step 5. Calculate and analyze the performance of the improved ID3 tree;

Step 6. Set the best model discriminating index value Ψ, and select the ID3 tree with better overall performance.

(3) Build a cost-sensitive random forest

Step 1. Set the conditions for the secondary screening of ID3 trees and the limit on the number of ID3 trees in the forest, introduce the cost-sensitive factor Bcost, and construct a cost-sensitive random forest (RF);

Step 2. Use the constructed RF to predict the samples in TestD, and each test sample is predicted to be one of two categories (i.e., defective or non-defective);

Step 3. Calculate and analyze the predictive performance of RF;

Step 4. Output the final prediction result of the sample.

3.2 Cost-sensitive ID3 Tree

Use binary segmentation method to construct ID3 tree. In the process of building an ID3 tree, a single type of child node and the minimum number of nodes are used as the general conditions for the ID3 tree node to stop splitting. When it is found that the number of remaining samples SN is the same category or SN is less than or equal to the minimum number of nodes, that is, the split threshold ε, stop splitting and classify all of them as a leaf node to reduce the impact of overfitting.

When selecting the category of the leaf node, the cost-sensitive factor Acost related to the number of different categories in the data set is introduced as the weight of the defect module in the leaf node to increase the tendency of predicting the defect cate-

gory, as in formula (5). The general denominator num_y is the number of true defective samples in the data set, the numerator num_x is the number of true non-defective samples, and η and λ represent auxiliary parameters. Without loss of generality, add 1 to both the numerator and denominator.

$$A\text{cost} = (num_x + 1)/(num_y + 1) \times \eta \pm \lambda \tag{5}$$

To filter the ID3 tree with better overall performance, for the constructed ID3 tree, the pros and cons model threshold Ψ is introduced, which is set according to the custom values of the error rate, recall rate, and precision rate indicators. Use the test set to test the trained ID3 tree to obtain test indicators such as error rate. If the test index is lower than the defined Ψ, the decision tree is considered unqualified and cannot be included in the number of decision trees specified by the user.

3.3 Cost-sensitive Random Forest

The cost-sensitive ID3 tree is used as the decision tree in the random forest. Before adding the constructed ID3 tree to the random forest, the conditions for the secondary screening of the ID3 tree are introduced. The numTrees refers to the total number of ID3 trees trained. When the performance of the ID3 tree satisfies: error $\leq \sum_{i=1}^{\text{numTrees}} \text{error}_i/numTrees$, recall $\geq \sum_{i=1}^{numTrees} recall_i/numTrees$, precision $\geq \sum_{i=1}^{numTrees} precision_i/numTrees$, the ID3 trees will be added to the forest.

Secondly, the random forest will limit the number of ID3 trees added. The number of ID3 trees in the random forest needs to meet: $(numTrees \times 1/2) \leq numF \leq (\text{num}Trees \times 4/5)$, numF is the number of ID3 trees in the random forest.

Finally, in the stage of voting to determine the sample category, the cost-sensitive factor Bcost is introduced, which is limited by the number of decision trees in the random forest with different sample prediction results and the number of samples in different categories in the data set. Let it be the weight of the number of decision trees that predict the sample category as a defect to increase the tendency to predict the defect category. The solution of Bcost is as formula (6), where, $t\text{rees}_x$ is the number of ID3 trees that predict the sample as a non-defect category, $t\text{rees}_v$ is the number of ID3 trees that predict the sample as a defect category, t_x is the number of real non-defective samples in the data set, t_v is the number of real defective samples, η, λ, γ (γ>0, generally take 1) are auxiliary parameters.

$$B\text{cost} = \frac{trees_x}{trees_y+\gamma} \times \sqrt{(t_x + 1)/(t_y + 1)} \times \eta \pm \lambda \tag{6}$$

4 Experiment

4.1 Experimental Data Set And Processing

The experimental data comes from the KC3 and CM1 data sets of the NASA MDP data warehouse. They are a collection of modules after software measurement, which can be downloaded from the website.

In the data set, faced with the problem of missing values of reasonable data, this

paper proposes to use the median of the column where the missing value is located to fill the missing value of this data. For example, there are n samples in a certain data set Data, and each sample is represented as d_i (i = 0,1,2,…, n − 1), There are m attributes (including label attributes), and each attribute is represented as p_j(j = 0,1,2,…, m − 1), The j-th attribute of the i-th sample is expressed as dp_{ij}, When the dp_{ij} of a piece of data is missing, after sorting this column of data, use formula (7) to assign value to dp_{ij}.

$$value = \begin{cases} dp^{n/2^j}, \text{n为偶数} \\ dp^{n+1/2^j}, n\text{为奇数} \end{cases} \tag{7}$$

4.2 Method Evaluation Index

The binary classification results of software defect prediction are represented by a confusion matrix, as shown in Table 1. In Table 1, TP means that positive cases are correctly predicted as positive cases; FN means that positive cases are incorrectly predicted as negative cases (Type II misclassification); FP means that negative cases are incorrectly predicted as positive cases (Type I misclassification); TN means that a negative case is correctly predicted as a negative case. The actual number of positive (defective) samples is P=TP+FN; the actual number of negative (non-defective) samples is N=FP+TN; the total number of all samples is C=P+N.

This paper uses correct rate, recall rate, precision rate, F-Measure index to measure the predictive ability of the software defect prediction model and verify the effectiveness of the proposed method. The calculation methods of these performance indicators are described in detail as follows. And, on any given data set, each performance index takes the average of 15 running results as the final performance index value of the method.

(1) Accuracy refers to the proportion of modules that are correctly classified in all modules.

$$Accuracy = \frac{TP+TN}{C} \tag{8}$$

(2) Recall refers to the proportion of all correctly classified defective modules to all truly defective modules.

$$Recall = \frac{TP}{P} \tag{9}$$

(3) Precision refers to the proportion of all correctly classified defective modules to all predicted defective modules.

$$Precision = \frac{TP}{TP+FP} \tag{10}$$

(4) F1-Measure is a weighted harmonic average of precision and recall and is often used to evaluate the quality of a classification model.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{11}$$

Table 1. Confusion matrix for software defect prediction

		Forecast category	
		defective	non-defective
Instance category	defective	TP (True Positive)	FN (False Negative)
	non-defective	FP (False Positive)	TN (True Negative)

4.3 Verification Experiment

This paper mainly conducts two sets of experiments. Through verification experiments, the feasibility of the cost-sensitive random forest method is verified, and the effectiveness of the proposed method is verified through comparative experiments. In the two data sets of the experiment, take the KC3 data set as an example, and repeat similar operations on the other data sets.

Input: public data set KC3, public data set division ratio Ex=7:3, number of decision trees constructed by the user numTrees=200

Output: predicted defect label for each test module

Step 1. Data preprocessing is performed on the public data set KC3, and the data volume of TrainD in the data set KC3 is 135 and that of TestD is 59;

Step 2. Perform data perturbation on trained to obtain the training set TrainD_train and the test set TrainD_test of the training ID3 tree;

Step 3. Calculate the number of attributes ms=7 after attribute disturbance by formula (12), randomly select the sample containing only 7 attributes from TrainD_train to generate a training set TrainD_trainM;

$$\text{ms} = log(\sqrt{m}) * 10 - 1 (ms < m, m: \text{Total number of attributes}) \tag{12}$$

Step 4. Set ε=5, η=0.5, λ=0.4, and use formula (5) to get Acost≈1.75 is used as the weight of the defect category to construct a cost-sensitive ID3 tree;

Step 5. Use the constructed ID3 tree to predict the samples in TrainD_test and determine whether each sample is defective;

Step 6. Calculate various performance values to obtain the predictive performance of the ID3 tree;

Step 7: Set the best model identification index value Ψ (that is, Ψ_{error}=34%, Ψ_{recall}=25%, $\Psi_{accuracy}$=22%), and select ID3 trees with better overall performance;

Step 8: Screen ID3 trees again, and limit the number of ID3 trees in the random forest. In the random forest, the number of ID3 trees is 100≤numF≤160, and the constructed random forest is obtained.

Step 9. Set γ=1, η=2.5, λ=0.35, and use the formula (6) to obtain Bcost≈1.62 as the voting weight of the defect category to construct a cost-sensitive random forest(cost-RF);

Step 10: Use the constructed improved random forest model to vote and predict the sample categories in TestD;

Step 11: Calculate various performance values to obtain the predicted performance of RF, as shown in Table 2;

Step 12. Output the final prediction result of the sample.

Table 2. Average experimental results of cost-RF 15 runs.

Data Set	Accuracy (%)	Precision (%)	Recall (%)	F-Measure (%)
KC3	88.20	65.49	73.51	69.27
CM1	88.62	67.85	73.63	70.62

4.4 Comparative Experiment

To verify the prediction performance of the cost-sensitive random forest method, this paper compares the experimental results of this method with the traditional ID3-based random forest, SVM, and LASSO-SVM. To avoid losing generality, all experiments are compared under the same experimental environment, the data set, and the data preprocessing steps to verify the effectiveness of the cost-sensitive random forest method.

To make the experimental results more reliable, the average value of all performance indicators of 15 runs is taken as the final performance indicator value of each prediction model. Table 3 shows the accuracy (ACC), the precision (PRE), the recall (REC), and the F-Measure values of the traditional random forest model (t-RF), SVM, and LASSO-SVM on the two test sets (KC3, CM1).

Table 3. Average experimental results of 15 runs of other models.

Data Set	KC3				CM1			
	ACC (%)	PRE (%)	REC (%)	F-Measure (%)	ACC (%)	PRE (%)	REC (%)	F-Measure (%)
t-RF	69.65	60.31	52.18	55.95	74.60	62.10	56.02	58.90
SVM	74.28	64.67	60.18	62.34	74.32	62.25	59.34	60.76
LASSO-SVM	88.22	65.79	73.01	69.21	90.02	65.23	76.35	70.35

5 Experimental Results And Discussion

5.1 Analysis Of Verification Experiment Results

According to the verification experiment results in Table 3, it can be found that the cost-sensitive improved random forest model (cost-RF) is in the above data set. As the imbalance rate increases, the number of data increases. It can ensure that the precision rate will not change significantly, and the recall rate will be improved to a certain extent. Figure 2 shows the comparison of experimental results. Among them, in the KC3 data set, due to the low imbalance rate, this model increases the occurrence of Type I in the prediction process, that is, by reducing the precision, to improve the recall rate index. But the overall results also confirm the effectiveness of this model from the side. Under the CM1 data set, it can be seen that under a certain imbalance

rate and data volume, the precision rate is relatively stable, and the recall rate has been improved. At the same time, the F-Measure indicator has exceeded 70%. Further, the feasibility of this model is verified.

5.2 Comparative Test Results Analysis

According to the comparative experimental results in Table 3 and the comparison results of all models in the correct, precision, recall, and F-Measure, the results are shown in Figure 2. It can be found that cost-RF is superior to the random forest model and SVM composed of traditional ID3 trees in any index, especially in the index of recall rate, cost-RF is much higher than them. Compared with the LASSO-SVM model, it can be seen from Figure 3 that due to the cost-sensitive factor introduced in the cost-RF twice, the accuracy index is weaker than that of the LASSO-SVM. However, F-Measure is better than LASSO-SVM, and cost-RF is less than 8% in the difference between precision and recall. According to the above experimental data, cost-RF can effectively increase the weight of predicting defect categories while maintaining a high accuracy rate, reduce the frequency of Type II misclassification, and minimize the cost of misclassification.

Fig. 1. Flow chart of software defect prediction model.

6 Conclusion

This paper proposes a software defect prediction method based on cost-sensitive and improved random forest. Specifically, this method makes full use of the advantages that the random forest is not affected by high-dimensional data, can extract important features, and has stable performance and the ID3 tree with binary segmentation is used as the base tree of the random forest. At the same time, by improving the data set

and setting the filter conditions and quantity limits of ID3 trees in the forest, introduced cost-sensitive factors twice, and constructed a software defect prediction model based on cost-sensitive and improved random forests. It alleviates the high frequency of Type II misclassification caused by the serious class imbalance in the data set. Experimental results show that the proposed method can better overcome the impact of the imbalance of the data set class, while maintaining a high accuracy rate, effectively reducing the cost of misclassification.

However, in the face of the severe class imbalance under a large amount of data, the cost-sensitive factor introduced in the model proposed in this paper has a reduced effect, and it cannot minimize the cost of misclassification and maintain a high accuracy rate. So, the future work is to improve the introduced cost-sensitive factors to make them more applicable.

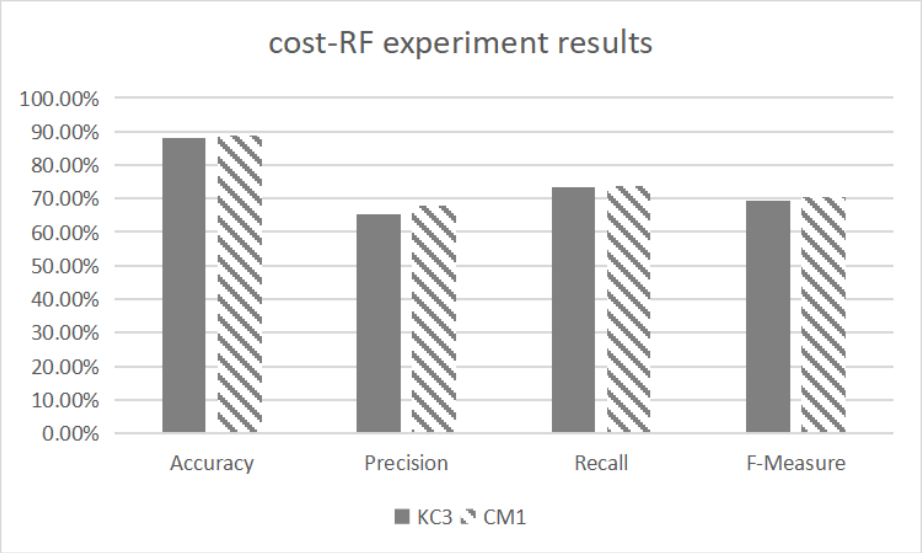

Fig. 2. Average experimental results of 15 runs of Cost-RF on different data sets.

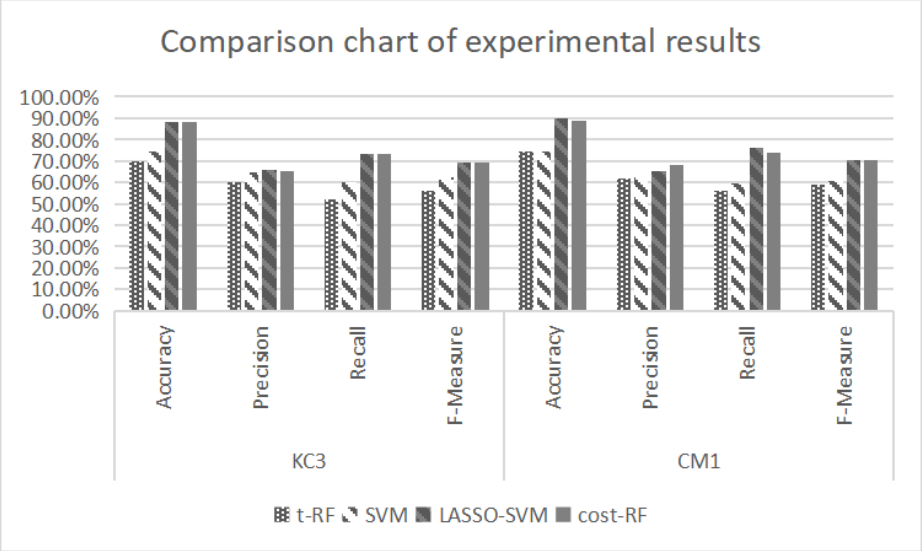

Fig. 2. Average experimental results of 15 runs of Cost-RF on different data sets.

References

1. Liang Cai, Yuanrui Fan, Meng Yan and Xin Xia., "Research Progress in Real-time SoftwareDefect Prediction."Journal of Software 30.05(2019):1288-1307.
2. Nagappan, Nachiappan and Thomas Ball., “Use of relative code churn measures to predict system defect density.” Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. (2005): 284-292.
3. Compton, B. Terry and Carol Withrow., “Prediction and control of ADA software defects.” J. Syst. Softw. 12 (1990): 199-207.
4. Menzies, Tim, Justin S DiStefano and Andres S. Orrego., “Assessing Predictors of Software Defects.” (2004).
5. Xing, Fei, Ping Guo and Michael R. Lyu., “A novel method for early software quality prediction based on support vector machine.” 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05) (2005): 10 pp.-222.
6. Wang, Kechao, Lin Liu, Cheng-jun Yuan and Zhifei Wang., “Software defect prediction model based on LASSO–SVM.” Neural Computing and Applications (2020): 1 - 11.
7. Ji, Haijin, Song Huang, Yaning Wu, Zhan-wei Hui and Changyou Zheng., “A new weighted naive Bayes method based on information diffusion for software defect prediction.” Software Quality Journal (2019): 1-46.
8. Arar, Ömer Faruk and Kürsat Ayan., “Software defect prediction using cost-sensitive neural network.” Appl. Soft Comput. 33 (2015): 263-277.
9. Zheng, Jun., “Cost-sensitive boosting neural networks for software defect prediction.” Expert Syst. Appl. 37 (2010): 4537-4543.
10. Khoshgoftaar, Taghi M., Edward B. Allen, Wendell D. Jones and John P. Hudepohl., “Classification-tree models of software-quality over multiple releases.” IEEE Trans. Reliab. 49 (2000): 4-11.
11. Zhou, Lijuan, Ran Li, Shudong Zhang and Hua Wang., “Imbalanced Data Processing Model for Software Defect Prediction.” Wireless Personal Communications 102 (2018): 937-950.
12. Sun, Zhongbin, Qinbao Song and Xiaoyan Zhu., “Using Coding-Based Ensemble Learning to Improve Software Defect Prediction.” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42 (2012): 1806-1817.
13. Siers, Michael J. and Md. Zahidul Islam., “Software defect prediction using a cost-sensitive decision forest and voting, and a potential solution to the class imbalance problem.” Inf. Syst. 51 (2015): 62-71.
14. Liu, Mingxia, Linsong Miao and Daoqiang Zhang., “Two-Stage Cost-Sensitive Learning for Software Defect Prediction.” IEEE Transactions on Reliability 63 (2014): 676-686.
15. Xu, Ling, Bei Wang, Ling Liu, Mo Zhou, Shengping Liao and Meng Yan., “Misclassification Cost-Sensitive Software Defect Prediction.” 2018 IEEE International Conference on Information Reuse and Integration (IRI) (2018): 256-263.
16. Quinlan, J. Ross., “Induction of Decision Trees.” Machine Learning 1 (2004): 81-106.
17. Zhu, Lin and Yang Yang. “Improvement of Decision Tree ID3 Algorithm.” CollaborateCom (2016).
18. Kraidech, Suratchanan and K. Jearanaitanakij. “Improving ID3 Algorithm by Combining Values from Equally Important Attributes.” 2017 21st International Computer Science and Engineering Conference (ICSEC) (2017): 1-5.
19. Zhang, He and Runjing Zhou. “The analysis and optimization of decision tree based on ID3 algorithm.” 2017 9th International Conference on Modelling, Identification and Control (ICMIC) (2017): 924-928.

20. Breiman, Leo., “Bagging predictors.” Machine Learning 24 (2004): 123-140.
21. Belgiu, Mariana and Lucian Daniel Drăguţ. “Random forest in remote sensing: A review of applications and future directions.” Isprs Journal of Photogrammetry and Remote Sensing 114 (2016): 24-31.