A Three-Level Training Data Filter for Cross-project Defect Prediction

Cangzhou Yuan[1(✉)], Xiaowei Wang[1], Xinxin Ke[1], and Panpan Zhan[2]

[1] School of Software, Beihang University, Beijing 100191, China
{yuancz,xiaowei_wang,kexinxin}@buaa.edu.cn
[2] Beijing Institute of Spacecraft System Engineering, Beijing 100094, China
panpan3210@qq.com

Abstract. The purpose of cross-project defect prediction is to predict whether there are defects in this project module by using a prediction model trained by the data of other projects. For the divergence of the data distribution between different projects, the performance of cross-project defect prediction is not as good as within-project defect prediction. To reduce the difference as much as possible, researchers have proposed a variety of methods to filter training data from the perspective of transfer learning. In this paper, we introduce a "project-instance-metric" hierarchical filtering strategy to select training data for the defect prediction model. Using the three-level filtering method, the candidate projects that are most similar to the target project, the instances that are most similar to the target instance, and the metrics with the highest correlation to the prediction result are filtered out respectively. We compared three-level filtering with project-level filtering, instance-level filtering, and the combination of project-level and instance-level filtering methods in four classification algorithms using NASA open source data sets. Our experiments show that the three-level filtering method achieves more significant f-measure and AUC values than the single level training data filtering method.

Keywords: Machine learning · Cross-project defect prediction · Transfer learning

1 Introduction

Software defect refers to a kind of problem, error, or hidden functional defect in the computer software or program that destroys the normal operation ability. In the process of software development, the generation of software defects is inevitable. Incorrect understanding of software requirements, unreasonable development process, and immature development technology might lead to defects. Once the defect is produced, the later it is discovered in the software development process, the greater the cost of fixing the defect. At present, software testing, static inspection, and other methods are mainly used to find defects to ensure the quality of software.

Q. Wu et al. (Eds.): WiSATS 2020, LNICST 357, pp. 109–123, 2021.
https://doi.org/10.1007/978-3-030-69069-4_10

Software defect prediction can use machine learning to predict defect-prone modules based on metrics such as software code characteristics. Software testers can reasonably allocate test resources based on the predicted results to detect defects more efficiently and accurately. Most of the researches on software defect prediction is based on within-project defect prediction (WPDP), that is, the historical data used for training prediction models come from the same project. However, in the actual development process, often a new project lacks of historical data. Collecting and labeling training data take a lot of manpower and resources. So it may face the problem of insufficient data to support model training. However, in the actual development process, there is often a new project to be developed, which may cause the problem that data is too insufficient to support model training.

To solve this problem, researchers consider cross-project defect prediction (CPDP) [1–3]. Cross-project defect prediction uses historical data from other projects to train prediction models to predict whether current projects have defects. Due to factors such as project type, development language, developer habits, and so on, the distribution characteristics of training dataset and test dataset are divergent among different projects, which causes dataset shift [4] and seriously affects the performance of prediction model. Currently, researchers mainly use transfer learning to solve this problem. Transfer learning applies the models trained based on the data of a certain project to other different but related projects. Within the research scope of transfer learning, researchers mainly propose the methods to improve the performance of cross-project defect prediction in the aspects of instance and metric, including instance selection, instance weight setting, metric selection and metric mapping, etc.

The method based on instance selection refers to the selection of appropriate instances from the source project to form the model training data according to the software module instances of the target project to realize the cross-project model transferring. The methods based on instance selection mainly include Burak filter [5], Peters filter [6] and riTDS filer [7] which proposed by Peng et al. to select training data on project level and instance level. The above methods are all based on fixed metrics for instance selection, without considering the correlation between metrics and classes. The metric selection method can reduce the impact of irrelevant and redundant metrics on the performance of the defect prediction model. An empirical study is conducted by Qiao et al. [8] on NASA and PROMISE datasets which showed that both the metric subset selection and metric ranking approaches can improve the performance of CPDP.

Both methods are used in CPDP to improve model prediction performance, but few researchers combine the two methods to select training data of prediction model. Therefore, considering the selection of the data for the training defect prediction model from the two aspects of instance selection and metric selection, a three-level filtering strategy is proposed to find the training data that can be used to train the most suitable defect prediction model for the target data. The three hierarchies in the three-level filtering method refer to the project-level, the instance-level and the metric-level respectively. At the project-level filtering, source projects with

high similarity to target projects are selected based on distribution characteristics. At the instance-level filtering, the source instances with high similarity to the target instances are selected based on the instances data. At the metric-level filtering, a metric with a high correlation with the predicted result is selected. To evaluate this strategy, we will focus on the following research issues:

RQ1: How does our strategy perform on each classifier compared to other strategies?
RQ2: Which classifier performs best by our strategy in cross-project defect prediction?
RQ3: Is our strategy practical in specific datasets?

The rest of the paper is organized as follows: Sect. 2 discusses the related work. Afterward, a detailed description of the three level filtering approach are presented in Sect. 3. We evaluate these strategies in a case study in Sect. 4. Section 5 discusses our strategies based on results. Finally, a summary of our strategies is persented and the directions for future work are discussed in Sect. 6.

2 Related Work

To our knowledge, the first research to conduct a feasibility analysis of cross-project defect prediction was completed by BriandL et al. [9]. Based on two midsize Java systems in the same environment, they conducted cross-project defect prediction experiments using logistic regression and MARS (Multivariate Adaptive Regression Splines). The results show that the accuracy of cross-project defect prediction is not as good as that of within-project defect prediction, but the performance of cross-project defect prediction is better than chance.

The method of instance selection mainly selects training data from project level and instance level. Herbold [10] proposed two strategies for project filtering based on the distribution characteristics of metrics, namely EM clustering and the nearest neighbor algorithm. The authors performed experiments on 44 datasets from 14 open-source projects. The results show that the two strategies can effectively improve the performance of cross-project defect prediction, but there is still some gap with within-project defect prediction. Kawata et al. [11] considered the noisy instances in cross-projects improve performance by using density-based spatial clustering of applications with noise (DBSCAN) filter approach. Can Cui et al. [12] proposed iForest filter which is an unsupervised machine learning approach to simplify the training data for CPDP to improve the model performance. This method uses path length to judge abnormal and normal instances, not distance or density. iForest filter uses a fixed size of sub-instances to conduct a small number of trees.

He et al. [13] studies from the perspective of metric selection and selects the subset of minimized metric elements through iterative selection. Amasaki et al. [14] used unsupervised learning to remove irrelevant and redundant metrics from the target project.

From the above related studies, the selection methods of training data mostly focus on only one related aspect, and few studies combine the two aspects of instance and metric. In our work, we combined them to implement source project data selection at the project-level and instance-level, and to remove redundant metrics at the metric-level to reduce the complexity of the model. Thus, the three-level Filtering approach is proposed to improve the performance of the defect prediction model.

3 Methodology

In the actual software development process, due to the lack of sufficient training data for newly developed projects or the high cost of data labeling, the economic benefits of software defect prediction are low and the feasibility is poor. Therefore, transfer learning is used to solve the problem of insufficient data for the target project with the help of the model trained based on the label data of the source project. Due to the divergence of data distribution between source and target projects, the performance of target projects in the prediction model is dramatic. To mitigate the impact of this problem, three level training data filtering method is proposed. First, at the project level, source projects which are the most similar to the target project from multiple candidate projects are filtered. Second, at the instance level, instances that have high distribution similarity to the target project instance from the source project instances are filtered. Third, at metrics level, metrics using feature selection which can identify and selects metrics having high correlation with classes are filtered. The training data of multiple candidate projects are filtered by the three level filtering to form training data. The prediction model is trained with the training data, and the target data is input to the trained model and the prediction result is output. The process of CPDP using three-level filtering are presented in Fig. 1.

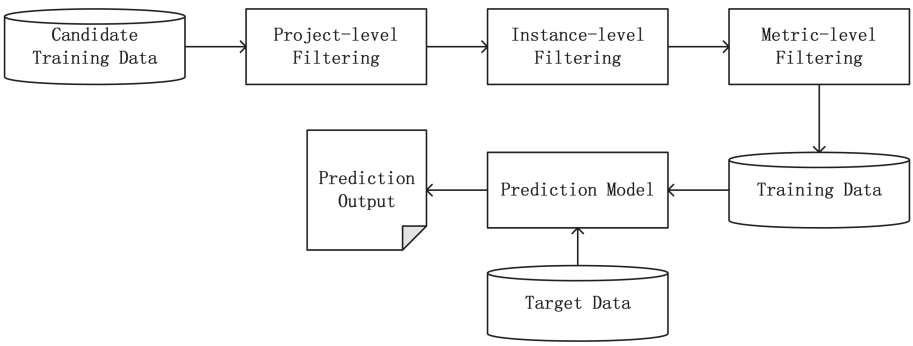

Fig. 1. The process of CPDP using three-level filtering.

3.1 Project-Level Filtering

When filtering projects, source projects that are as similar as the target project type, developer experience, and developer ability as possible are wanted to find. Based on the above requirements, the distribution characteristics of project data are used as the input condition for project similarity judgment including mean, maximum, minimum, standard deviation. The distribution characteristics are determined based on the research of He et al. [15]. Mean can show the central location where project data is relatively concentrated. Maximum and minimum values can show the distribution range of project data. Standard deviation can show how discrete project data is.

In this method, X_{ij} is used to represent the jth metric of the ith instance. The distribution characteristic data of the ith instance is represented as $F_{\mathrm{i}} = \{C_{11}, C_{12}, \ldots C_{jk}\}$, and where C_{jk} is defined as kth distribution characteristics of the jth metric. K-nearest neighbor method based on Euclidean distance of distribution characteristics is used to judge the similarity between projects, that is to calculate Euclidean distance between each candidate source project and the target project separately and select k nearest source projects as the second level source projects. Algorithm 1 gives a detailed description of project-level filtering.

Algorithm 1: The Description of Project-level Filtering

Input:

candidate project $C\left(\left\{S_{s(n)}\right\}_{n=1}^{n=N}\right)$;
distribution characteristics of target project S_{tc};
number of desired project k;
for n=1 to N **do**
$SM_n = Similarity\left(C\left(S_{tc}\right), C\left(S_{s(n)}\right)\right)$;
end for ;
Sort(SM_n, DESC);
$S_c \leftarrow$ Select(SM_n , k);
Return S_c;

3.2 Instance-Level Filtering

The candidate training dataset consists of all the instances from the k source projects filtered at project-level filtering. In the instance-level filtering stage, instances with high similarity to the target project dataset are selected from the candidate training dataset. In the existing research, there are two main ways to filter the instance. One is Burak filter [5] and the other is Peters filter [6]. At the instance-level filtering, our approach based on the second method. First, the candidate training set is used as the center for the instance selection so that each instance of the target project forms its own candidate subset. Then choose the instance centered on the instance in the target project. This is equivalent

to a second filter, which is the "two-way selection" between the instance of the candidate training set and the instance of the target project. It considers not only the more defective information that may be contained in the candidate training dataset but also the objective requirements that require the prediction model to get better prediction performance on the target dataset. Algorithm 2 gives a detailed description of instance-level filtering.

Algorithm 2: The Description of Instance-level Filtering

Input:

candidate training dataset $NSP = \{X_{source(n)}\}_{n=1}^{n=N}$

the instance of the target project S_t;

for i=1 to N **do**

$Array_{target(i)} \leftarrow SelectClosest(X_{source(i)}, S_t)$;

end for ;

while $Array_{target(i)}$ not NULL **do**

$result[i] \leftarrow min(Array_{target(i)})$;

end while ;

Return result[i];

3.3 Metric-Level Filtering

The number of metrics is not directly proportional to the performance of the model. Therefore, metric-level filtering can identify and select a subset of metrics that are highly class-related, which can be regarded as a process of searching and optimizing. The metric-level filtering process includes four parts: the subset generation process, the subset evaluation process, the stopping criterion, and the verification process. The process of metric-level filtering is presented in Fig. 2.

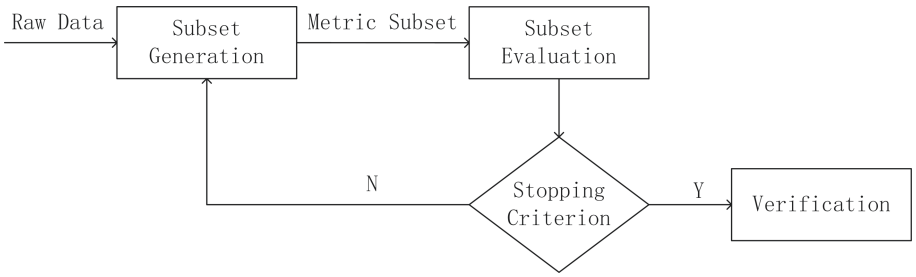

Fig. 2. The process of metric-level filtering.

The generation process of the subset includes the determination of the search starting point, the determination of the direction, and the selection of the search strategy. The starting point of the search is set to the empty set S. As the search

progresses, the best metric from the set of metrics that have never been included in S is selected and added to S continuously. The search strategy is set as sequence search, that is, in the search process, metrics are continuously added to the current subset of metrics according to a certain order, so as to obtain an optimized subset of metrics. To determine the order, the method of information gain is used to arrange the order of the metrics. Information gain considers that the more information a feature brings to the classification system, the more important this feature is and the greater the corresponding information gain is. The so-called amount of information is the entropy, which converts the characteristic probability into the degree to which the feature describes the classification results. Entropy is defined as follows:

$$H(X) = -\sum_{i=1}^{n} p\left(x_i\right) \log p\left(x_i\right) \tag{1}$$

Information gain is the difference between the entropy and the characteristic condition entropy. The information gain from feature T to classification C is defined as follows:

$$IG(T) = H(C) - H(C|T) \tag{2}$$

where $H(C|T)$ is the conditional entropy of classification C under the feature T condition. The information gain represents the degree to which the uncertainty of information decreases when the characteristic T is determined. That is, after the data is classified according to the feature T, the certainty of the classified data is higher than that before the partition.

The method of subset evaluation is generally divided into the filter method and the wrapped method. In the filter method, it generally does not rely on specific learning algorithms to evaluate the subset of metrics. On the contrary, the subsequent learning algorithm is embedded in the metric selection process, in the wrapper method. This method determines its pros and cons by testing the predictive performance of the metric subset on this algorithm, and rarely pays attention to the predictive performance of each metric in the metric subset. Because the evaluation of the filter method has a large deviation from the performance of the subsequent learning algorithm and consider that the current classification performance of cross-project defect prediction is low, the method of embedding the subsequent learning algorithm into the metric selection process is adopted to minimize the performance deviation of the subsequent learning algorithm. The method of information gain is used to determine the search sequence to a certain extent to make up for the shortcoming of the wrapped method's lack of attention to the prediction performance of a single metric.

The stopping criterion is set to complete the search of the metric subset. The result verification is set as the optimal subset of the metric element under the given performance measure.

4 Case Study

Our approach is validated in a case. The experiment is introduced from four aspects: The first is a brief description of data. The second is to introduce the machine learning algorithm used in the experiment. The third is about evaluation performance measures. Finally, the results of the experiment are given.

4.1 Data Setup

The data used in our experiment is part of the NASA metrics data program (MDP) data sets. The repository currently contains 13 module-level data sets explicitly intended for software metrics research [16]. Each dataset contains static code metrics and defect labels for the module. The module here refers to a function or method. Static code metrics used in the NASA metrics data program include lines-of-code (LOC), Halstead, and McCabe-based measures. The development languages used in the NASA metrics data program include C, C++, JAVA, Perl. Considering the impact of the development language on project similarity, we selected data sets in 13 module-level data sets in which the development language is C as our experimental data. Table 1 gives details of the experimental dataset.

Table 1. NASA MDP Data Sets.

Project	# Instance	# Metric	# Defect-prone	% Defect-prone
CM1	344	37	42	12.21%
MC2	127	39	44	34.65%
PC1	759	37	61	8.04%
PC2	1585	36	16	1.01%
PC3	1125	37	140	12.44%
PC4	1399	37	178	12.72%

4.2 Prediction Models

Naive Bayes classifier is a simple statistical learning classifier based on the assumption of feature independence. This assumes that for a given class variable, one particular feature is independent of the others. Naive Bayes classifier has the advantage that other classifications do not. Lewis [17] describes Naive Bayes as a classifier based on Baye's theorem and it decomposed as

$$p\left(C_k|\mathbf{x}\right) = \frac{p\left(C_k\right)p\left(\mathbf{x}|C_k\right)}{p(\mathbf{x})} \tag{3}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of characteristic attributes of the sample data. C_k is k possible outcomes and in our case, it could be 0 or 1.

K-nearest neighbor (K-NN) algorithm is an instance-based classification algorithm. The basic idea is to select the most recent sample in the same attribute class as the training feature. It is usually decided by a majority vote.

The advantage of defect prediction models based on Logistic Regression is that it is easy to understand and realize and it is meaningful for defect rough judgment, but it is easy to unfit and has low classification accuracy.

Random Forests are an ensemble learning method for classification, regression, and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

4.3 Performance Measures

The software defect prediction model refers to the binary problem of whether the software module contains defects, so the performance evaluation index of the classification model in the field of machine learning can be applied to the evaluation of the prediction model. For the sake of simplicity, this study describes the performance evaluation indicators of the prediction model in this paper based on the confusion matrix [18]. Table 2 shows the confusion matrix with four defect prediction results. Here, true positive (TP), false negative (FN), false positive (FP) and true negative (TN) is the number of defective instances that are predicted as defective, the number of defective instances that are predicted as non-defective, the number of non-defective instances that are predicted as defective, and the number of non-defective instances that are predicted as non-defective, respectively.

Table 2. Confusion Matrix.

	Predicted Defective	Predicted non-defective
Actual defective	true positive (TP)	false negative (FN)
Actual non-defetive	false positive (FP)	true negative (TN)

With the confusion matrix, the following performance evaluation measures are adopted, which are commonly used in the defect prediction studies. Table 3 shows the calculation formula of the used performance evaluation measures for defect prediction. Recall,pd [19] is defined as the ratio of the number of defective instances that are correctly classified as defective to the total number of defective instances.

Precision [20] is defined as the ratio of the number of modules that are correctly predicted as defective to the number of modules that are predicted as defective.

The value of AUC (Area Under the Curve) [21] is the area under the roc curve. ROC (Receiver Operating Characteristic) curve is a graphical method to

describe the relationship between the real case rate, pd, and false-positive case rate, pf in the classification model. AUC considers the classifier's classification ability for both positive and negative examples, and can still make a reasonable evaluation of the classifier in the case of imbalanced samples.

F-measure comprehensively considers recall and precision. It is defined as the harmonic mean of recall and precision, which comprehensively considers the overall performance of the model on recall and precision, and can comprehensively reflect the performance of the model. It is widely used in cross project defect prediction research, such as [22,23].

Therefore, the comprehensive indicators AUC and f-measure are used to evaluate the performance of each strategy in our experiments.

Table 3. Performance Evaluation Measures.

Measure	Defined as
Recall (pd)	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
AUC	The area under the ROC curve
f-measure	$\frac{2\times Precision\times Recall}{Precision+Recall}$

4.4 Results

Based on six data sets (CM1, MC2, PC1, PC2, PC3, PC4), four classifiers (KNN, LR, NB, RF) are used to compare the results of the four strategies under f-measure and AUC in our experiments. The experimental results are shown using project-level filtering, instance-level filtering, the filtering method which combined project-level and instance-level, and three-level filtering respectively in the Fig. 3. Besides, under each strategy, the average value of f-measure and AUC on six data sets using four different classifiers was counted in the Table 4. When the same algorithm is used on the same data set and the same performance measure is used as the judgment standard, the percentage of each strategy with the best performance in all experiments is calculated. The results are shown in the Table 5. As a whole, the three-level filtering strategy achieves better prediction performance than the other three strategies. Naive Bayes classifier performs better using our strategy.

Under the strategy we proposed, the f-measure and AUC measure values of each classifier on each data set are calculated. The statistical data is shown in the Table 6. When we focus on the MC2 dataset, we find that Under the f-measure, the Naive Bayes algorithm is the best. Due to NB has the largest f-measure value and the largest f-measure difference with other classifiers based on MC2.

Considering economic benefits and other factors, software defect prediction has not been widely used in engineering practice. Rahman F et al. [21] proposed that the recall value determines the practical validity of the software defect prediction model. Shell et al. [24] pointed out that the recall value of human

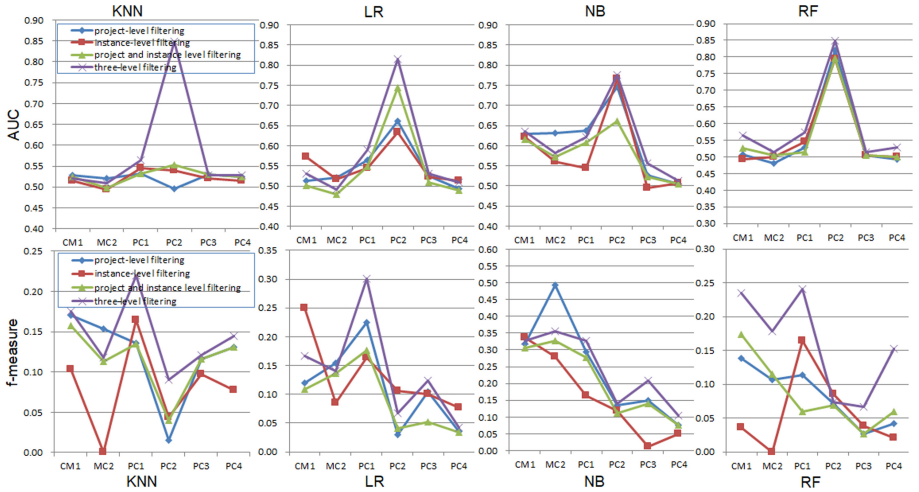

Fig. 3. Results of different strategy on the performance of different prediction models.

Table 4. For f-measure and AUC, the average values of four strategies on six data sets using KNN, LR, NB, RF.

f-measure	Project-level fltering	Instance-level filtering	Project-level and istance-level fltering	Three-level fltering
KNN	0.12	0.08	0.12	0.14
LR	0.11	0.13	0.09	0.14
NB	0.24	0.16	0.21	0.24
RF	0.08	0.06	0.08	0.16
AUC				
KNN	0.52	0.52	0.53	0.58
LR	0.55	0.55	0.55	0.58
NB	0.61	0.58	0.58	0.61
RF	0.56	0.56	0.56	0.59

Table 5. The percentage of each strategy with the best performance in all experiments.

Strategy	f-measure	AUC
Project-level fltering	20.8%	12.5%
Instance-level filtering	8.33%	20.8%
Project-level and instance-level filtering	4.17%	0
Three-level filtering	66.66%	66.66%

Table 6. Under our strategy, the f-measure and AUC values of Naive Bayes and the change rates(%) of KNN, LR, RF classifiers.

f-measure	CM1	MC2	PC1	PC2	PC3	PC4
NB	0.33	0.35	0.33	0.14	0.21	0.10
KNN	−46.7	−66.8	−32.9	−36.1	−42.5	38.5
LR	−49.1	−60.4	−7.8	−51.9	−40.7	−59.2
RF	−28.2	−49.7	−26.3	−47.4	−68.4	45.7
AUC	CM1	MC2	PC1	PC2	PC3	PC4
NB	0.64	0.58	0.62	0.78	0.56	0.51
KNN	−18.0	−12.5	−8.9	9.3	−5.1	3.1
LR	−16.3	−15.7	−5.0	5.1	−4.6	−0.8
RF	−20.2	−17.7	−14.7	5.9	−9.2	−4.0

code review defect is more than 60%. Therefore, recall of different strategies using different algorithms under a single dataset is calculated, as shown in Fig. 4. It can be seen from the figure that our strategy has obvious improvement under the KNN algorithm.

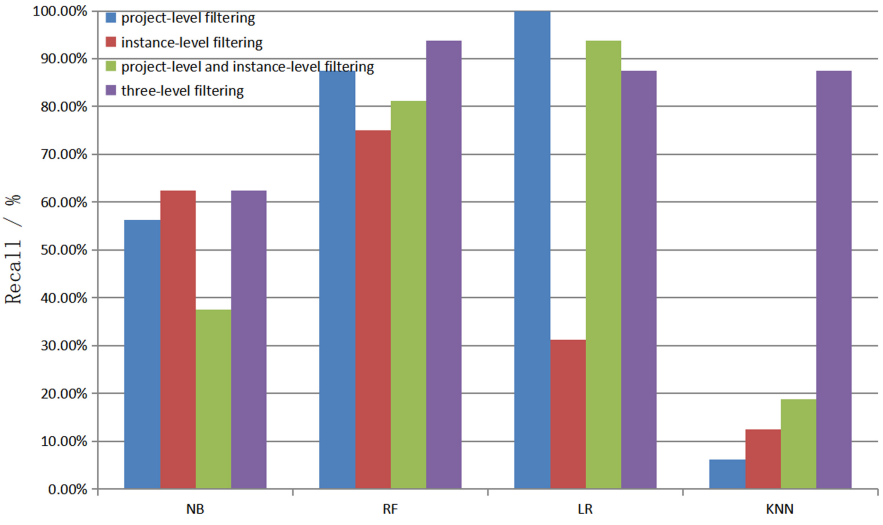

Fig. 4. Recall values of different strategies using different algorithms.

5 Discussion

RQ1: How does our strategy perform on each classifier compared to other strategies?

As is shown in Fig. 3, three-level filtering is the best in most experiments. Even though the performance in individual experiments is not optimal, it is not the worst. Therefore, the proportion of the best performance of each strategy in all the experiments is counted, and the results are shown in the Table 5. The percentage of three-level filtering with the best performance among all experiments is much higher than the other three strategies. According to the Table 4, judging from the average value of different strategies using the same classifier on six data sets for different performance measures, the three-level filtering performed well when using KNN, LR, NB, RF classifiers for defect prediction. The above experimental results can fully prove the superiority of the three-level strategy.

RQ2: Which classifier performs best by our strategy in cross-project defect prediction?

The statistics are shown in the Table 6. NB classifier has absolute advantages in CM1, MC2, PC1, and PC3 data sets. In the PC2 data set, the f-measure of the KNN classifier, LR classifier, and RF classifier is not as good as the NB classifier, but AUC measure is better than Nb, and the improvement range is less than 8%. In the PC4 data set, the LR classifier is better than the NB classifier in performance; the KNN classifier and RF classifier are better than the NB classifier in one measure, and the other measure is worse than NB classifier.

RQ3: Is our strategy practical in specific datasets?

In our limited number of experiments, the recall of the PC2 dataset is best. It can be seen from Fig. 4 that our strategy for the recall using any algorithm is greater than 60%. From this point of view, three-level filtering is feasible in practice.

6 Conclusion

In this work, a three-level training data selection strategy that combines project selection, instance selection, and metric selection is proposed. d. At the same time, the overall performance of the strategy is better than other strategies. The experiment also proves that naive Bayes classifier is more suitable for building a prediction model with simplified training data after three-level filtering. In the limited experimental data set, there is recall value to meet the prediction rate of manual review defects, which further proves that the cross-project defect prediction under three-level strategy has certain practicality.

At present, the recall does not achieve the value of manual review based on most data sets. Our future work will focus on practical validation of methods on more datasets, such as PROMISE.

Acknowledgements. This paper is partly supported by the Pre-research of Civil Spacecraft Technology (No. B0204).

References

1. Zhou, Y., et al.: How far we have progressed in the journey? an examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. **27**, 1–51 (2018)
2. Xu, Z., Yuan, P., Zhang, T., Tang, Y., Li, S., Xia, Z.: HDA: cross-project defect prediction via heterogeneous domain adaptation with dictionary learning. IEEE Access **6**, 57597–57613 (2018)
3. Hosseini, S., Turhan, B., Gunarathna, D.: A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. Softw. Eng. **99**, 1–40 (2017)
4. Turhan, B.: On the dataset shift problem in software engineering prediction models. Empir. Softw. Eng. **17**, 62–74 (2012). https://doi.org/10.1007/s10664-011-9182-8
5. Turhan, B., et al.: On the relative value of cross-company and within-company data for defect prediction. Empir. Softw. Eng. **14**, 540–578 (2009). https://doi.org/10.1007/s10664-008-9103-7
6. Peters, F., et al.: Better cross company defect prediction. In: 10th Working Conference on Mining Software Repositories (MSR) (2013)
7. He P., et al.: Simplification of training data for cross-project defect prediction. Computer Science (2014)
8. Yu, Q., Qian, J., Jiang, S., Wu, Z., Zhang, G.: An empirical study on the effectiveness of feature selection for cross-project defect prediction. IEEE Access **7**, 35710–35718 (2019). https://doi.org/10.1109/ACCESS.2019.2895614
9. Briand, L.C., et al.: Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans. Softw. Eng. **28**(7), 706–720 (2002)
10. Herbold, S.: Training data selection for cross-project defect prediction. In: Proceedings of the 9th International Conference on Predictive Models in Software Engineering (2013)
11. Kawata, K., Amasaki, S., Yokogawa, T.: Improving relevancy filter methods for cross-project defect prediction. In: Software Engineering & Advanced Applications, vol. 619. IEEE (2015)
12. Cui, C., Liu, B., Wang, S.: Isolation forest filter to simplify training data for cross-project defect prediction. In: 2019 Prognostics and System Health Management Conference (2019)
13. He, P., Li, B., Liu, X., Chen, J., Ma, Y.T.: An empirical study on software defect prediction with a simplificd metric set. Inf. Soft. Technol. **59**, 170–190 (2015)
14. Amasaki, S., Kawata, K., Yokogawa, T.: Improving cross-project defect prediction methods with data simplification. In: Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications (2015)
15. He, Z., Shu, F., Yang, Y., Li, M., Wang, Q.: An investigation on the feasibility of cross-project defect prediction. In: Proceedings Eighth IEEE Symposium on Software Metrics (2012)
16. Gray, D., et al.: Reflections on the NASA MDP data sets. IET Softw. **6**, 549–558 (2012)
17. Lewis, D.D: Naive (Bayes) at forty: the independence assumption in information retrieval. In: European Conference on Machine Learning (1998)
18. Witten, I.H., et al.: Data mining: practical machine learning tools and techniques. ACM Sigmod Rec. **31**, 76–77 (1999)

19. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
20. Buckland, M., Fredric, G.: The relationship between recall and precision. J. Am. Soc. Inf. Sci. **45**, 12–19 (1994)
21. Rahman, F., et al.: Recalling the 'Imprecision' of cross-project defect prediction. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (2012)
22. Nam, Ja., et al.: Transfer defect learning. In: Proceedings of the 2013 International Conference on Software Engineering (2013)
23. Kim, S., Whitehead, E.J., Zhang, Y.: Classifying software changes: clean or buggy? IEEE Trans. Softw. Eng. **34**(2), 181–196 (2008)
24. Shull, F., et al.: What we have learned about fighting defects. In: Proceedings 8th IEEE Symposium on Software Metrics (2002)