Software Defect Prediction Using Dictionary Learning

Hongyan Wan[1,2], Guoqing Wu[1,2], Ming Cheng[1,2], Qing Huang[1,2], Rui Wang[1,2] and Mengting Yuan[1,2*]
1. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
2. School of Computer, Wuhan University, Wuhan 430072, China
E-mail: { why0511, wgq, chengming, qh,wangrui1989,ymt }@whu.edu.cn

***Abstract*—With the popularization of software version control system and defect tracking tools, large amounts of software development data is recorded. How to effectively use these data to improve the quality of software development, has become a hot topic in recent years. Software defect prediction technology can take full advantage of the historical data to build predictive models and automatically detect defective modules for efficient software test to improve the quality of a software system. But the class-imbalanced data makes the prediction model classifying a modules as a defective-free one easily, while the misclassification of defective modules generally incurs much higher cost risk than the misclassification of defective-free ones. To resolve this problem, we propose a cost-sensitive software defect prediction method using dictionary learning. It iteratively optimizes the classifier parameters and the dictionary atoms, to ensure that the extracted features (sparse representation) are optimal for the trained classifier; Moreover, we take the different misclassification costs into account, increasing the punishment on misclassification defective modules in the procedure of dictionary learning, making classification inclining to classify a module as a defective one. Experimental results on the 10 class-imbalanced data sets of NASA show that our method is more effective than other methods.**

Keywords- Software defect prediction; Dictionary learning; Cost-sensitive; Bilevel optimization; Sparse coding

I. INTRODUCTION

In recent years, soft version control system and defect tracking system have widely used in software project, so that each code submission, each defect report comments are fully recorded. How to make use of these data to measure the efficiency, calculate the cost, and predict the quality of the products becoming a hot topic in the software development of big data background [1]. The software defect prediction technology can take full advantage of the software historical data to build the prediction model, which can accurately predicts whether the software module contains the defect and allocate the test resource to analyze the defective module in the early stage of the system development, thus reducing the software development and maintenance cost [2].

The traditional software defect prediction mainly uses the classification technology of machine learning [3][4], generally divided into two stages: the first stage is the feature extraction, analysis the attribute feature of input software model (such as Halstead, McCabe and others), generate feature vector. The second stage is the defect identification, using the previously trained classifier model to discriminate whether the input software module is defective. Commonly used method include Support Vector Machine (SVM) [5], Naive Bayesian (NB) [6], Neural Network [7] and so on. The performance of the traditional method is seriously limited by the software features, and the feature selection is the difficulty of software defect prediction, lack of common feature representation and feature selection algorithms. Through in-depth research on the characteristics of defect prediction tasks and defect data, we can find that sparseness is a general nature of defect prediction: 1) in general, the defect module is distributed sparsely in the software data set, that is, the number of defective modules is far less than defective-free modules in the software system, which makes the defect data has the class-imbalance characteristics[8][9]; 2) the software module to be tested is usually similar to the historical data of the project and belongs to the same, so it can be sparsely represented by a specific similar historical software module.

With the development of theory and application in Sparse Representation based Classification (SRC), it has been widely applied in pattern recognition, image classification, defect prediction, and so on[10][11][12]. Compared with the traditional classification algorithms, SRC has better discrimination and robustness, it consists of three parts: one is over-complete dictionary; the second is the linear representation under a specific sparse constraints based on the over-complete dictionary; the third is the classification according to the over-complete dictionary and the sparse coefficient of the input data. Wright et al. [13] proposed a SRC-based face recognition method that achieved good results. Jing et al. [14] are the first to apply the dictionary learning technology to the field of software defect prediction, and achieved good results by using the over-complete dictionary prediction defect software module. However, the classification loss function of the method was defined as data sample reconstruction error, making the classification performance mainly dependent on the quality of the training data.

Based on the sparseness principle of software defects, this paper proposes a software defect prediction model based on cost-sensitive dictionary learning (CSDL) under the background of large data, and designs a projection first-order stochastic gradient descent algorithm based on sparseness. The algorithm solves the objective function and verifies the validity of the above model and algorithm on the 10 class-imbalanced data sets provided by NASA. The method is divided into two stages: 1) training the over-complete dictionary, extract the features (sparse coding); 2)

* Corresponding author.
DOI reference number: 10.18293/SEKE2017-188

training classifier according to the extracted features. Training dictionary to solve the sparse coding or training classifier parameters alone cannot get the ideal classification effect. In this paper, we propose a method to iteratively optimize the parameters of the classifier and the dictionary atom to ensure the feature extraction (sparse representation) under the optimal classification performance. At the same time, we take full account of the impact of different misclassification in the process of learning the dictionary, improve the cost of misclassifies a defective module as defective-free one, so that the resulting dictionary is more inclined to classification defect module. Because sparseness is a kind of non-related and versatile features, the model and algorithm proposed in this paper are superior to other traditional methods.

II. RALATED WORK

The purpose of software defect prediction is to automatically identify software modules that contain defects, which are critical to ensuring software quality.

Menzies et al. [15] argue that software engineering data sets are class-imbalanced, using data sampling and lifting algorithms to improve the performance of the model. However, the sampling strategy will change the distribution of the source data, affecting the effect of the prediction. Sun et al. [16] proposed a coding-based integrated learning approach that transforms the class-imbalanced defect data into multi-class balanced data to avoid the class-imbalance problem by specific coding strategies. Jiang et al. [17] pointed out that the performance of the defect prediction model is affected by different misclassification cost. In this paper, on the one hand, cost-sensitive learning method is used to produce the smallest misclassification cost, which makes it easy to classify the defective module with higher misclassification cost, and reduce the class-imbalance problem. On the other hand, considering the different misclassification cost, so that the prediction model is more inclined to classify the defect module, thus improve the prediction performance of the model.

In the sparse representation based classification method, it is necessary to set the dictionary in advance. Wright et al. [13] used all classes of training samples as a dictionary coding test sample, and classify the test samples through minimizing the reconstruction errors. However, the dictionary which they proposed cannot represent test data efficiently due to the presence of noise in the original data set. In addition, the number of dictionary atoms is too large to increase the complexity of coding. Mariral et al. [18] proposed a discriminant dictionary learning method by training the classifier coding coefficients and validating them in the digital recognition and texture classification. As the classification loss function in the SRC is usually defined as a reconstruction error of data samples. Therefore, the performance of the classifier mainly depends on the quality of training data and cannot achieve good classification effect. Jiang et al. [19] integrated the identification of sparse coding error and classification error into a target function to improve the representation and discriminant of the dictionary, joint the learning dictionary with a linear classifier. Although the above methods can achieve better classification accuracy in the corresponding fields, the imbalanced defect data in the field of software defect prediction makes the learning dictionary tendency classification module as defect-free, and the cost of the misclassified defect module is far more than the defect-free module. Therefore, full consideration of this information can be more effective in improving the performance of defect prediction.

In this paper, we optimize the classifier parameters and dictionary atoms iteratively in the process of dictionary learning, to ensure that the feature (sparse representation) is extracted under the optimal classification performance.

III. COST-SENSITIVE DICTIONARY LEARNING METHOD FOR BIG DATA

This section describes the defect prediction process of CSLD method in two stages, and the concrete process is shown in Figure 1. The first stage is to optimize dictionary and the classifier parameters iteratively; the second stage is the classification. In software defect prediction, a test module is usually similar to a partial history module and belongs to the same class (defect or defect-free class), that is, the test module can be compressed as a linear combination of a small number of the same class of modules in historical training data, this representation is usually sparseness. However, classification loss function in the SRC is usually defined as the reconstruction error of the data samples, it is heavily rely on the data quality, and cannot achieve good classification effect. Therefore, this paper presents a dictionary learning method, which optimize the dictionary atoms and classifier parameters iteratively, and applies it to the task of software defect prediction. In addition, there are two types of misclassification errors in the software defect prediction. The type I misclassifies a defective-free module as defective module, while the type II misclassified a defective module as defective-free module, the cost of them is different. In the process of dictionary learning, we fully consider the misclassification costs of type I and II errors to reduce the minimum misclassification cost and improve the prediction performance.

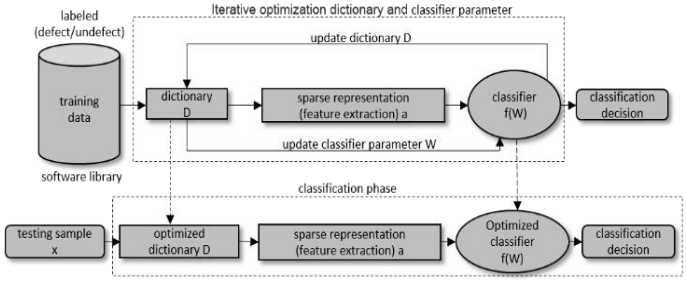

Figure 1. The predict process of CSDL model

The main process of CSLD method is shown in Figure 1. First, using KSVD [20] technology to initialize over-complete dictionary D. In the iterative optimization stage, the sparse coding (feature extraction) a of training samples x in over-complete dictionary D is solved and used to optimize the parameters of the classifier; at the same time, in order to ensure that the sparse coding is under the premise of the optimal classification performance, we apply the algorithm 1 which is proposed in this paper to update

the over-complete dictionary iteratively. In the test classification phase, the test sample solves the sparse coding in the optimized dictionary and inputs it into the optimization classifier. We use logic loss function as the objective function of the classifier.

Sparse coding depends on the over-complete dictionary, its representation is $D=(d_1,d_2,\cdots,d_p),d_i\in\mathbb{R}^m(m\ll p)$, where d_i represents a base vector in the dictionary, referred to as atom. Each sample x can be combined linearly by dictionary atoms, $a\in\mathbb{R}^p$ is the coefficient of sparse representation of x in the dictionary, it is solved by Elastic Net [21].

A. Feature Extraction based on Sparse Coding

Assuming that the software module sample space is $\mathcal{X}$, the label space is $\mathcal{Y}$, the training sample set is $X=\{x_1,x_2,\ldots,x_n\}\in\mathbb{R}^{m\times n},X\in\mathcal{X}$, where n is the number of sample modules, m is the number of attributes of the sample. The module label $y_i\in Y$ indicates the label of the software module x_i. In the software defect prediction model $Y\in\mathcal{Y}=\{+1,-1\}$, where "+1" denotes a defective module, "-1" denotes a defective-free module. Given training sample set $L=\{(x_1,y_1),\cdots,(x_i,y_i)\}(i=1,\cdots,n)$, $D=(d_1,d_2,\cdots,d_p)\in\mathbb{R}^{m\times p}$ is the learned over-complete dictionary, $A=(a_1,a_2,\cdots,a_n)\in\mathbb{R}^{p\times n}$ is the sparse coding based on D, the sparseness is guaranteed by the l_1 norm of the matrix as a regularization term. It should be noted that each sub-dictionary should have the same number of dictionary atoms, in general, the number of sub-dictionary atoms is equal to the dimension of the data. Elastic Net is used to solve sparse coding coefficients:

$$A=\arg\min_A\frac{1}{2}\|X-DA\|_F^2+\lambda_1\sum_i\|a_i\|_1+\lambda_2\|A\|_F^2 \quad (1)$$

It can also be expressed as a sparse coding for a single sample x_i, namely:

$$a_i=\arg\min_{a_i}\frac{1}{2}\|x_i-Da_i\|_2^2+\lambda_1\|a_i\|_1+\lambda_2\|a_i\|_2^2 \quad (2)$$

where λ_1 and λ_2 are the regularization of the parameters used to balance these three terms. When set $\lambda_2=0$, the above formula is a l_1 sparse decomposition problem. It should be noted that, must set $\lambda_2>0$, to ensure that the objective function is differentiable.

B.Cost-Sensitive Dictionary Learning

In this paper, an optimal classification model is obtained according to the input feature for the minimum classification loss, and the cost of different misclassification is considered in the process of dictionary construction [22]. In the task of software defect prediction, we use Cost(+1) to represent the cost of misclassifying a defective module as defective-free module, and Cost(-1) represents the cost of misclassifying a defective-free module as defective module.

Using the iterative optimization sparse coding $a_i(x_i,D)$ to train the classifier parameter W, and using logistic loss as a classifier objective function. Therefore, the loss function can be defined as:

$$T(A(X,D),W)=\sum_{i=1}^n\ell(y_i,W,a_i(x_i,D)) \quad (3)$$

Where $\ell(y_i,W,a_i(x_i,D))=Cost(y_i)\log(1+e^{-y_iW^Ta_i(x_i,D)})$ is a weighting logic loss function, $a_i(x_i,D)$ is the sparse coding of sample x_i in dictionary D, $Cost(y_i)$ is the cost of misclassification.

Finally, the joint minimization objective function proposed in this paper can be expressed by the two-level optimization model [23]:

$$\begin{aligned}&\min_{D,W}T(A,W)+\frac{v}{2}\|W\|_2^2\\ &s.t.\quad A=\arg\min_A\frac{1}{2}\|X-DA\|_F^2+\lambda_1\sum_i\|a_i\|_1+\lambda_2\|A\|_F^2\end{aligned} \quad (4)$$

In the test phase, each test sample is sparsely encoded on the dictionary D according to the formula (2), and then the resulting sparse coefficients is used as the input of the previous optimal classifier. Finally, the test samples are classified according to the formula (4).

C. Optimization Algorithm

We constructing the projection first-order stochastic gradient descent algorithm (SGD) to solve the formula (4), using KSVD technology to initialize sub-dictionary for each class, as shown in algorithm 1. It consists of an outer SGD loop that increments the sample training data, each time the gradient of the classifier parameters W and dictionary D for sample is solved, and update W and D.

Algorithm 1 Stochastic Gradient Descent Algorithm (SGD) to solve formula (4)

Input: $(X,Y);v,\lambda_1,\lambda_2;D_0,W_0$ (the initial dictionary and classifier parameters); $Iter$ (the number of iterations); t_0,ρ (learning rate)

Output: D,W

1. For *t*=1 to $Iter$ do
2. Draw a subset (X_t,Y_t) from (X,Y);
3. Sparse coding: computer A^* using Feature -Sign algorithm:
   $A^*=\arg\min_A\frac{1}{2}\|X_t-DA\|_F^2+\lambda_1\sum_i\|a_i\|_1+\frac{\lambda_2}{2}\|A\|_F^2$;
4. Compute the active set: Λ (the nonzero suport of A);
5. Compute β^*:Set $\beta^*_{\Lambda^c}=0$ and
   $\beta^*_\Lambda=(D_\Lambda^TD_\Lambda+\lambda_2I)^{-1}\nabla_{A_\Lambda}[T(A,W)]$;
6. Choose the learning rate:
   $\rho_t=\min\left(\rho,\rho\left(t_0/t\right)\right)$;
7. Update D and W by a projected gradient step:
   $W=\prod_W\left[W-\rho_t(\nabla_WT(A,W)+vW)\right]$
   $D=\prod_D\left[D-\rho_t(-D\beta^*A^T+(X_t-DA)\beta^{*T})\right]$,
   where $\prod_W$ and $\prod_D$ are respectively orthogonal projections on the embedding space of W and D;

8. end for
9. return W and D.

In practice, the distribution of data is usually unknown, we sample in a random sorted training set. In addition, in order to allow each iteration to sample more samples rather than sampling only a single sample, this paper adopts the minibatch strategy. In order to get stable results in the experiment, 200 samples were sampled at each time.

IV. Experiment and Result Analysis

This section introduces the experimental setup in detail, including the dataset, the evaluation metric and the analysis of the experimental results.

A. Experimental Subject

Table I. NASA DATASET

dataset	No.attr	No.def	Defect modules
CM1	38	42	12.21%
JM1	21	1759	18.34%
KC1	21	325	15.54%
KC3	40	36	18.00%
MC2	40	44	34.65%
MW1	20	27	10.23%
PC1	20	61	8.04%
PC3	20	140	12.44%
PC4	38	178	12.72%
PC5	39	503	2.96%

The experimental data for this paper are based on 10 class-imbalanced data sets provided by NASA, as shown in table II. The NASA project is the software system or subsystem of NASA, which contains static code metrics for software modules and the corresponding defect labels (defective software modules are labeled as "Y", and defective-free module are labeled as "N"). The ratio of the class-imbalance, that is, the percentage of the number of minority classes and the number of majority classes ranges from 2.96% to 34.65%, the size of the data set namely the number of software modules is between 127~17001.

B. Performance Evaluation

Generally, the performance of the prediction model is evaluated synthetically by using recall, precision, type I error rate (Err1), type II error rate (Err2) and F1 value.

The metrics of model evaluation are defined as follows:

Recall: The ratio of the number of modules correctly predicted as defect to the number of real defective modules.

$$recall = \frac{TP}{TP + FN} \tag{5}$$

Precision: The ratio of the number of modules correctly predicted as defect to the number of modules predicted as defect.

$$precision = \frac{TP}{TP + FP} \tag{6}$$

Recall and precision are a contradictory measure, so it is not appropriate to use only one of them to evaluate the performance of the prediction model. The F1-measure can combine them to evaluate models, F1-measure is widely used in performance evaluation of prediction model and other software engineering research fields [12]. The calculation formula of F1-measure is the harmonic mean of recall and precision:

$$F1 = \frac{2 \times recall \times precision}{recall + precision} \tag{7}$$

Type I error rate (Err1): The ratio of the number of falsely predicted as defective modules and the number of defective-free modules in real.

$$Err_1 = \frac{FP}{TN + FP} \tag{8}$$

Type II error rate (Err2): The ratio of the number of falsely predicted as defective-free modules and the number of defective modules in real.

$$Err_2 = \frac{FN}{TP + FN} \tag{9}$$

From the above ratio relationship can be seen, Err1 is relative to the real defective-free module predict as defective module, and Err2 is relative to the real defective module predict as defective-free module, both of them should be used simultaneously. Due to the different costs of detection and maintenance of Err1 and Err2, the influence of the cost of misclassification on the model is defined by the Expect Cost of Misclassification (ECM) as shown in equation (10), where C1 and C2 are the cost of Err1 and Err2 respectively, corresponds to Cost (-1) and Cost (+1) in the cost matrix, Pndf and Pdf represent the prior probabilities of the defective-free modules and defective modules, respectively. In many cases, as the misclassification cost of a single class is difficult to obtain, the influence of different misclassification costs on prediction model is often analyzed by using the misclassification cost ratio instead of a single misclassification cost, as shown in equation (11).

$$ECM = C_1 Err_1 P_{ndf} + C_2 Err_2 P_{df} \tag{10}$$

$$NECM = Err_1 P_{ndf} + (C_2 / C_1) Err_2 P_{df} \tag{11}$$

C. Parameter Settings

Similar to previous work in literature [16], the parameters λ_1 and λ_2, regularization parameter v, and learning rate ρ of Elastic-net are determined by cross validation, we use heuristics to reduce the search space of these parameters. The specific process is as follows: Firstly, we need λ_2 >0 when we prove that the objective function is different, and the choice of small λ_2 value is a necessary condition for the convergence of the algorithm. However, the experimental results show that the better sparse representation can be obtained when setting λ_2 =0; Secondly, we can achieve better reconstruction results when the parameter λ_1 is near 0.025, therefore we test in the given range $\lambda_1 = 0.025 + 0.0125k(k \in \{-3, \cdots, 3\})$ to achieve the parameter λ_1; Thirdly, when there are a large number of training samples, the regularization parameter v can be set as an arbitrary small value, such as setting v =10^{-5}, without enough training samples, this parameter is set by cross validation; Finally, the maximum number of iterations is set to 500. In addition, similar to the literature [14], in order to emphasize the influence of risk costs on the prediction model, according to the actual situation of the project, we set different C_2/C_1 values of misclassification costs.

In all experiments, if not specified, CSDL parameter settings are obtained through 10-fold cross validation, so as to avoid over-learning. Although our experiments verify that the choice of these parameters can achieve better prediction results, the finer parameter adjustment can further improve the performance of the algorithm.

D. Experimental Design

We designed different experiments to evaluate the proposed CSDL algorithm, and compared it with the defect prediction algorithms which are proposed in recent years, including Cost-sensitive Discriminative Dictionary Learning (CDDL) [14], SVM [5], NB [6]，Coding based Ensemble Learning (CEL)[16] and Cost-Sensitive Boosting Neural Network (CSBNN)[17].

Experiment one. Comparison of experimental results. The experiment was used to explore the effectiveness of the proposed method, and verify the performance of the CSDL algorithm. It was running on the 10 NASA datasets, and compared with 5 defect predictive algorithms which were proposed in recent years. We try to follow the parameters of the algorithm in the original document.

Experiment two. The impact analysis of different misclassification costs rate, which explore the effect of predicted model performance under the different rate of misclassification costs. We set the misclassification costs rate of C2/C1 range from 1 to 10.

E. Experimental Results and Analysis

We compared the CSDL with SVM, NB, CEL, CSBNN and CDDL, using 10-fold cross validation on the 10 NASA datasets, and calculate 10 sets of evaluation indicators (the value of F1) respectively. We focus on the study of optimal performance of the algorithm corresponding to other two cost-sensitive algorithms CSBNN and CDDL, so it is not need to have the same misclassification costs among them. The results are shown in TABLE II. The last row of table shows the average F1 on all the experimental datasets, the best F1 are presented with boldface.

From Table II, we see that the average F1-measure of CSDL is the highest. Firstly, we compared CSDL with NB, SVM and CEL, CSDL can obtain the best F1-measure in all of the experimental data sets. For example, in the KC1 data set, the F1-measure of NB, SVM and CEL are 0.376, 0.295 and 0.352 respectively, while the F1-measure of CSDL is 0.446.

Secondly, we compared CSDL with two cost-sensitive algorithm CSBNN and CDDL. In order to obtain the optimal predict performance, we select different misclassification costs to different projects. Compared with CSBNN, among all of the NASA data sets, CSDL can obtain the better predict performance through adjustment corresponding misclassification cost rate. For example, in the PC3 data set, the optimal F1-measure of CSDL is 0.435, which is higher than CSBNN’s 0.382. While the F1-measure of CSDL are not all higher than CDDL. For example, in the JM1 data set, the optimal F1-measure of CSDL is 0.380, which is less than CDDL’s 0.395. In most cases, the F1-measure of CSDL is higher than CDDL.

In order to evaluate the performance of CSDL fairly and objectively, we make use of Wilcoxon signed rank sums test method to analysis the experimental results of F1-measure at a 5% significance level. That is, the performance difference between two comparison methods were statistically significant when p is less than 0.05. As shown in Table III, in all NASA data sets, only one value of p is larger than 0.05. Thus, the performance difference between CSDL and other five methods were statistically significant.

TABLE II. F1-MEASURE OF DIFFERENT METHODS

DATASET	NB	SVM	CEL	CSBNN	CDDL	CSDL
CM1	0.325	0.214	0.279	0.312	0.366	**0.398**
JM1	0.332	0.311	0.341	0.383	**0.395**	0.380
KC1	0.376	0.295	0.352	0.405	0.435	**0.446**
KC3	0.385	0.382	0.353	0.384	0.421	**0.455**
MC2	0.456	0.521	0.501	0.567	**0.632**	0.618
MW1	0.312	0.275	0.278	0.336	0.382	**0.395**
PC1	0.284	0.362	0.327	0.315	0.415	**0.421**
PC3	0.291	0.87	0.365	0.382	0.413	**0.435**
PC4	0.367	0.475	0.492	0.463	0.541	**0.577**
PC5	0.331	0.176	0.357	0.365	**0.600**	0.576
AVG.	0.346	0.330	0.364	0.391	0.460	**0.471**

TABLE III. WILCOXON RANK TEST P VALUE OF CSDL AND OTHER METHODS

DATASET	CSDL				
	NB	SVM	CEL	CSBNN	CDDL
CM1	2.95×10^{-9}	1.21×10^{-10}	3.71×10^{-5}	6.73×10^{-7}	2.91×10^{-7}
JM1	1.96×10^{-11}	3.92×10^{-6}	6.16×10^{-9}	0.1323	1.54×10^{-3}
KC1	3.66×10^{-9}	9.12×10^{-11}	5.74×10^{-7}	6.21×10^{-8}	9.61×10^{-6}
KC3	7.53×10^{-5}	5.71×10^{-3}	1.19×10^{-5}	3.26×10^{-6}	8.71×10^{-5}
MC2	6.68×10^{-8}	5.69×10^{-6}	3.21×10^{-6}	1.73×10^{-6}	9.91×10^{-8}
MW1	3.19×10^{-8}	2.91×10^{-5}	2.27×10^{-4}	1.88×10^{-5}	2.57×10^{-8}
PC1	1.31×10^{-4}	5.79×10^{-5}	3.29×10^{-6}	3.99×10^{-8}	5.41×10^{-7}
PC3	4.25×10^{-8}	2.73×10^{-10}	4.05×10^{-6}	4.26×10^{-5}	5.89×10^{-6}
PC4	9.37×10^{-11}	4.43×10^{-8}	4.75×10^{-8}	5.42×10^{-9}	4.19×10^{-7}
PC5	4.13×10^{-13}	4.28×10^{-7}	4.63×10^{-10}	4.63×10^{-11}	2.55×10^{-5}

We compared CSDL with other two cost-sensitive algorithms CSBNN and CDDL, and set different misclassification cost rate to analysis how it affects the module performance. We make use of evaluation index Err1, Err2 and NECM to describe the impact of misclassification cost rate on the overall cost. The performance of the model is evaluated synthetically by the F1-measure. Due to space limitations, only the experiment results of data sets KC1, CM1 and PC1 under the different misclassification cost rate are given, we set C_2/C_1 range from 1 to 10, as shown in Figure 2~5.

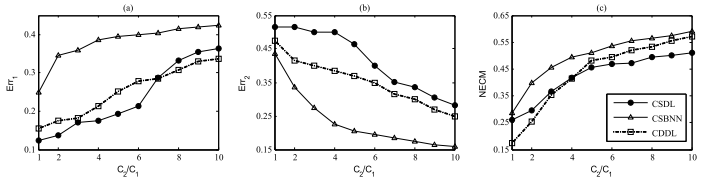

Figure 2. The performance comparison of KC1 dataset

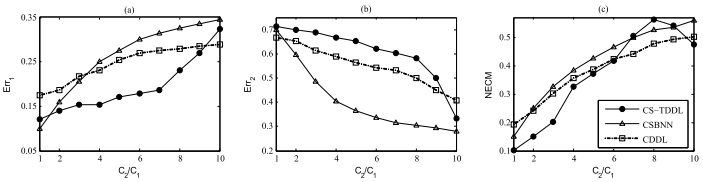

Figure 3. The performance comparison of CM1 dataset

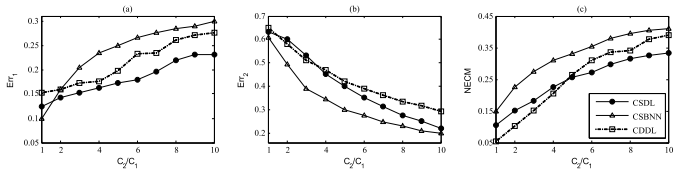

Figure 4. The performance comparison of PC1 dataset

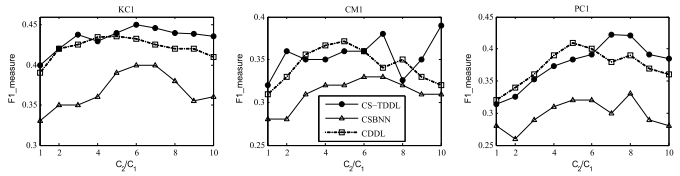

Figure 5. The F1-measure comparison on the KC1,CM1 and PC1

Figure 2~4 show that, with the increase in the misclassification cost rate, more and more defective modules were successfully detected, while the number of detective-free modules detect as defective modules also increases. The CDDL algorithm is more smooth in the Err1, Err2 indicators. CSDL and CDDL in the performance of NECM is always better than CSBNN algorithm. In the data sets KC1 and PC1, when the misclassification cost rate is less than 4, the performance of NECM on CSDL is not better than CDDL, but as the misclassification cost rate increase(larger than 5), the performance of NECM is significantly better than other methods. In the data set CM1, CSDL can get the best NECM performance when the misclassification cost rate is lower than 6, but the NECM performance of the CSDL algorithm decreases with the increase of the misclassification cost rate. In most cases, CSDL is always able to get the best NECM performance by effectively adjusting the misclassification cost rate.

In addition, the performance of three cost-sensitive algorithms on KC1, PC1, CM1 datasets is evaluated synthetically using F1-measure. As can be seen from Figure.5, the F1 performance of the prediction model changes significantly with the increase of the misclassification cost rate. Although in some cases, the F1 performance of CSDL is not optimal, but we can always make CSDL algorithm to achieve better F1 value by adjustment misclassification cost rate reasonably. In the actual application scenario, the software module's misclassification cost rate is difficult to obtain. As a result, it is often necessary to obtain an optimal predictive performance through an extensive search process for the misclassification cost rate.

V. CONCLUSIONS

In this paper, we propose a big data oriented cost-sensitive dictionary learning method to predict software defect, and use the logic regression classifier as the final classification judgment result. We iteratively optimize the dictionary atom and classifier parameters, and take full account of the different misclassification cost rate to improve the performance of the classifier. The experimental results show that our method has better detection effect than the other five methods, and can provide better support for software testing.

ACKNOWLEDGMENT

The research in this paper was partially supported by the National Natural Science Foundation of China under Projects No. 61373039, No. 61170022, No. 61003071 and No. 91118003.

REFERENCES

[1] G. Czibula, Z. Marian, and I. G. Czibula, “Software defect prediction using relational association rule mining,” Information Sciences, vol. 264, pp. 260–278, 2014.
[2] Pizzi N J. A fuzzy classifier approach to estimating software quality [J]. Information Sciences, 2013, 241: 1-11.
[3] Okutan A, Yildiz OT. Software defect prediction using Bayesian networks, Empir. Softw. Eng, 2014, 19(1):154-181
[4] Cheng M, Wu G, Wan H, et al. Exploiting Correlation Subspace to Predict Heterogeneous Cross-Project Defects[J]. International Journal of Software Engineering and Knowledge Engineering, 2016, 26(09n10): 1571-1580.
[5] Elish K, Elish M. Predicting Defect-prone Software Modules Using Support Vector Machines. Journal Systems and Software, 2008,81(5): 649-660
[6] Wang T, Li WH. Naïve Bayes Software Defect Prediction Model. Int. Conf. Computational Intelligence and Software Engineering. Wuhan: IEEE, 2010.1-4
[7] Zheng J. Cost-sensitive boosting neural networks for software defect prediction. Expert Systems with Applications,2010,37(6) : 4537–4543
[8] Zhou ZH, Liu XY. Training cost-sensitive neural networks with methods addressing the class imbalance problem. IEEE Trans. Knowledge and Data Engineering, 2006, 18(1):63–77
[9] Liu X Y, Wu J, Zhou Z H. Exploratory undersampling for class-imbalance learning[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2009, 39(2): 539-550.
[10] Xu Z, Liu Y, Mei L, et al. Semantic based representing and organizing surveillance big data using video structural description technology[J]. Journal of Systems and Software, 2015, 102: 217-225.
[11] Liu J, Yu X, Xu Z, et al. A cloud‐based taxi trace mining framework for smart city[J]. Software: Practice and Experience, 2016.
[12] Zhu G, He C, Shunxiang Z, et al. Weighted Indication-Based Similar Drug Sensing[J]. International Journal of Software Science and Computational Intelligence (IJSSCI), 2015, 7(1): 74-88.
[13] Wright J, Yang AY, Ganesh A, Sastry SS, Ma Y. Robust face recognition via sparse representation. IEEE Trans. Pattern Anal. Mach. Intell., 2009,31(2):210–227
[14] Jing XY, Ying S, Zhang ZW, Wu SS,Liu J. Dictionary learning based software defect prediction. In: Proc. of the 36th Int’l Conf. on Software Engineering. Hyderabad: ACM, 2014. 414–423
[15] Menzies T, Dekhtyar A, Distefano J,Greenwald J. Problems with precision: A response to comments on data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng., 2007, 33(9):637–640
[16] Sun Z, Song Q, Zhu X. Using coding-based ensemble learning to improve software defect prediction. Systems, Man, and Cyb ernetics, Part C: Applications and Reviews, IEEE Transactions on,2012, 42(6):1806–1817
[17] Jiang Y, Cukic B, Menzies T. Cost curve evaluation of fault prediction models. IEEE Int. 19th International Symposium on Software Reliability Engineering.Seattle:IEEE,2008.197-206
[18] Mairal J, Bach F, Ponce J. Task-driven dictionary learning. IEEE Trans. Pattern Anal. Mach. Intell., 2012, 34(2):791–804
[19] Jiang Z, Lin Z, and Davis L. Learning a discriminative dictionary for sparse coding via label consistent K-SVD. In: Proc. of the Int’l Conf. on CVPR,Barcelona:IEEE,2011.1697–1704
[20] Aharon M, Elad M, Bruckstein A. k-svd: An algorithm for designing overcomplete dictionaries for sparse representation. Signal Processing, IEEE Transactions on, 2006,54(11):4311–4322
[21] Zou H, Hastie T. Regularization and Variable Selection via the Elastic Net. J. Royal Statistical Soc. Series B, 2005,67(2): 301-320
[22] Cheng M, Wu G, Yuan M, et al. Semi-supervised Software Defect Prediction Using Task-Driven Dictionary Learning[J]. Chinese Journal of Electronics, 2016, 25(6): 1089-1096
[23] Colson B, Marcotte P, Savard G.An overview of bilevel optimization. Ann. Oper. Res., 2007, 153(1):235–256