Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 675368, 14 pages
http://dx.doi.org/10.1155/2014/675368

Research Article

Discrimination Analysis for Predicting Defect-Prone Software Modules

Ying Ma,[1] Ke Qin,[2] and Shunzhi Zhu[1]

[1] *School of Computer and Information Engineering, Xiamen University of Technology, Xiamen 361024, China*
[2] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China*

Correspondence should be addressed to Shunzhi Zhu; szzxmut2013@gmail.com

Received 19 October 2013; Revised 23 December 2013; Accepted 14 January 2014; Published 25 February 2014

Academic Editor: Osamu Mizuno

Software defect prediction studies usually build models without analyzing the data used in the procedure. As a result, the same approach has different performances on different data sets. In this paper, we introduce discrimination analysis for providing a good method to give insight into the inherent property of the software data. Based on the analysis, we find that the data sets used in this field have nonlinearly separable and class-imbalanced problems. Unlike the prior works, we try to exploit the kernel method to nonlinearly map the data into a high-dimensional feature space. By combating these two problems, we propose an algorithm based on kernel discrimination analysis called KDC to build more effective prediction model. Experimental results on the data sets from different organizations indicate that KDC is more accurate in terms of F-measure than the state-of-the-art methods. We are optimistic that our discrimination analysis method can guide more studies on data structure, which may derive useful knowledge from data science for building more accurate prediction models.

1. Introduction

Defect-prone software modules prediction is very critical for the high-assurance and mission-critical systems. It tries to estimate a functional relationship between the features of the software modules and the quality of the modules. Many software engineering researchers apply data mining methods on different software data sets. However, there are rarely researchers analyzing the inner structure of the data sets, either because it needs a good technical background on data science or the modules developed belong to strange domains for the local companies. Building prediction models requires solving binary classification problem as many pattern recognition applications. Many pattern recognition approaches are applied to build predictors but have different performances on different data sets. As pointed out by Khoshgoftaar et al. [1] and Menzies et al. [2], majority of the defects in a software system are located in a small percentage of the program modules; software defect data sets are highly class-imbalanced. Since then, many specific approaches to handling class-imbalanced problem were proposed in software defect prediction, such as sampling approaches, cost-sensitive approaches, feature selection approaches, and ensemble approaches. In addition to the class-imbalanced property, we think that software data sets have another property, that is, nonlinear separability. On data sets with better separability, most methods will yield good performances, while on data sets with worse separability, most methods will perform poorly. But, to the best of our knowledge, very few studies focused on the prediction model based on the inherent property of the software data sets.

This paper makes the following contributions. (1) We newly introduced the kernel based discrimination analysis on software data sets, to gain insight into the inherent property of the data used in defect prediction. The results of the analysis suggested that data sets used in this field have a nonlinearly separable property, which may require nonlinear algorithm to build predictors to improve the performance. (2) By comparing the transformation results of the linear discrimination analysis with kernel discrimination analysis, we proposed a kernel based algorithm to build defect predictor, which addressed the nonlinearly separable and class-imbalanced problems. (3) We conducted our experiments on data sets drawn from different projects and different companies

. The experimental results show that the proposed algorithm gives better performance on all the data sets when compared with the state-of-the-art methods.

The rest of this paper is organized as follows. Section 2 briefly reviews the background of the discrimination analysis techniques and software defect prediction algorithms. Based on the theories of linear and kernel discrimination, Section 3 presents our algorithm for building the defect predictor. Section 4 describes the software defect data sets, performance metrics used in this study, and shows the experimental results with discussions. Section 5 finalizes the paper with conclusions and future works.

2. Related Work

2.1. Discrimination Analysis. Most recently, ignoring the dependence among the variables, Menzies et al. [2] proposed a famous method based on naive Bayes classifier to build defect prediction model. But Fan et al. [3] hold that the theoretical misclassification rate of the naive Bayes classifier is higher than that of linear discrimination analysis method. Linear discriminant analysis (LDA) [4] is a classical multivariate technique for supervised learning, especially for classification problems which need projecting data vectors that belong to the real classes. It has been widely used in many applications such as traffic incident detection [5], face recognition [6], document classification [7], speech recognition [8], and image classification [9]. The linear discrimination analysis methods can find a compact representation of the original data when the data form a linear subspace. However, in the distribution of some data such as face images, which is highly nonlinear and complex, it cannot find this compact representation. It is therefore reasonable that when linear discrimination analysis methods fail to provide reliable results, we should try these nonlinear methods to achieve robust performances. A number of nonlinear methods have been developed to deal with these shortcomings of the linear discrimination analysis methods such as kernel-based approaches.

Kernel based discrimination analysis (KDA) has good performances in many applications such as face recognition [10], information retrieval [11], image classification [12]. Most recently, the discriminant analysis method is used to combat the class-imbalanced problem, which exists within the colon cancer data, lymphoma data, lung cancer data, breast cancer data, and gene-imprint data [13]. In order to find the defective modules, we also emphasize the importance of considering the class imbalance during software quality modeling. We found that the kernel based discrimination analysis has good performance on software defect prediction.

2.2. Software Defect Prediction. Software defect prediction is to predict the defect-prone modules for the next release of software or cross project software, as shown in Figure 1. With the software metric research advance, more and more researchers apply machine learning methods to predict defective software modules, such as interpretable models [14], J4.8 decision tree [15], Bayesian nets [16], ensemble method

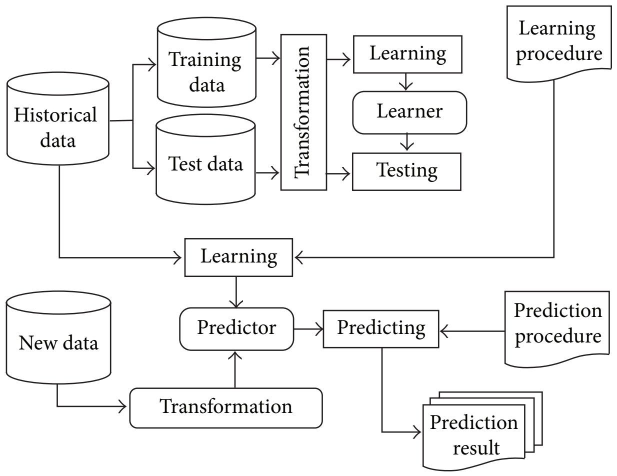


Figure 1: Software defect prediction procedure.

[17], transfer learning [18], asymmetric learning [19, 20], active learning [21. These articles compare the performance of learning methods and endorse the use of static code attributes for predicting defective modules. There are also a few articles reporting that the further progress in learning defect predictors may not come from better algorithms but come from more information content of the training data, such as [22].

The studies [23, 24] used the PCA and LDA to predict the fault-prone module directly, without analyzing the separability of data sets. We not only analyze the nonlinear separability, which is the property of the software data sets but also consider class-imbalanced problem which were widely studied recently [17].

Most recently, the kernel methods were used in software engineering to estimate the software effort [25]. But rare articles report the performance of predictor based on kernel methods for software defect prediction. In this paper we focus on nonlinear separable and class-imbalanced problems in software defect prediction. Based on the kernel discrimination analysis, a new classifier is proposed to provide the technique, which transforms low-dimensional input space into a high-dimensional feature space so as to make the software data separable in the new space and then calculates the local mean distances using the class distribution information to find the minority of defective modules.

3. Defect Prediction Based on Discrimination Analysis

In this section, we introduce the linear discriminant analysis technic and describe the kernel discrimination analysis based on the linear version. Then, we propose a kernel discrimination classifier, which is more suitable for building software defect predictor based on the inherent property of the software data sets.

Table 1: Symbols of the discriminant analysis.

Symbol	Description
S_b, S_b^{Φ}	The between-class scatter matrix
S_w, S_w^{Φ}	The within-class scatter matrix
n_i	The number of the instances of ith class
W, Ψ	The transformation matrix
c_1, c_2	The classes of nondefective and defective software module
μ	Mean vector of total instances
u_i	Mean vector of ith class
x_j^i	The jth instances in the ith class
T_r, T_r'	Original and transformed training data
T_e, T_e'	Original and transformed test data

3.1. Discrimination Analysis on Software Data Sets. Since software data sets used in the software defect prediction are drawn from varied systems which are written in different language, developed by different company, applied in different domains, the individual data sets appear to have quite different structure. The discrimination analysis theory provides a good method to give insight into the data distributions. Since the concept of discrimination analysis belongs to the experts' knowledge of artificial intelligence and knowledge engineering field, we should describe this technic to migrate the knowledge from data science to software engineering.

Here, we would like to predict the defective modules and nondefective modules in software by solving binary classification problem. Therefore, we show the discriminant analysis method constrained to two classes. Firstly, we calculate the between-class scatter matrix S_b and the within-class scatter matrix S_w for training data. These two matrixes are shown as (1) and (2), using the symbols in Table 1:

$$S_b = \sum_{i=1}^{2} n_i \left(u_i - u\right)\left(u_i - u\right)^{\top}, \tag{1}$$

$$S_w = \sum_{i=1}^{2}\sum_{j=1}^{n_i} \left(x_j^i - u_i\right)\left(x_j^i - u_i\right)^{\top}, \tag{2}$$

$$J_F(W) = \frac{W^T S_b W}{W^T S_w W}, \tag{3}$$

$$W_{\text{opt}} = \arg\max \frac{\left|W^T S_b W\right|}{\left|W^T S_w W\right|}. \tag{4}$$

Then, in order to find the maximum points of $J_F(W)$, we set derivative of (3) equal to zero. This means that when W is an eigenvector of $S_w^{-1} S_b$, the separation will be equal to the corresponding eigenvalue. By substituting (1) and (2) into (4), we get $J_F(W)$ as follows:

$$\begin{aligned} J_F(W) &= \frac{W^{\top} \sum_{i=1}^{2} n_i \left(u_i - u\right)\left(u_i - u\right)^{\top} W}{W^{\top} \sum_{i=1}^{2}\sum_{j=1}^{n_i} \left(x_j^i - u_i\right)\left(x_j^i - u_i\right)^{\top} W} \\ &= \frac{\sum_{i=1}^{2} n_i W^{\top} \left(u_i - u\right)\left(u_i - u\right)^{\top} W}{\sum_{i=1}^{2}\sum_{j=1}^{n_i} W^{\top} \left(x_j^i - u_i\right)\left(x_j^i - u_i\right)^{\top} W}. \end{aligned} \tag{5}$$

Suppose $R = W^{\top}(u_i - u)$, then $W^{\top}(u_i - u)(u_i - u)^{\top} W = RR^{\top}$ in the subspace; the original data points to be discriminated are projected as follow.

$$\tau = W^{\top} x. \tag{6}$$

Set the dimension of w equal to the dimension of the training data, then we can obtain the classifier with the threshold as prior probability $\log(p(c_1)/p(c_2))$, as in [26]. We can see that the objective of linear discriminant analysis approach is to maximize the ratio of between-class variance to withinclass variance. Therefore, we can exploit it to analyze the software data so as to give insight into the separability of the defective and nondefective classes. We will see that the software data are nonlinearly separable as shown in Section 4.

3.2. Kernel Discrimination Classifier. To deal with the software data sets which are nonlinearly separable, we perform nonlinear mapping $\Phi(x)$ to transform the input vectors X to a higher dimensional feature space. Then, a new classifier based on kernel discrimination analysis (KDC) is proposed to deal with the nonlinearly separable and class-imbalanced problem, which are often inherent in software defect prediction. In the kernel discrimination analysis (KDA) [27], the between-class scatter matrix is as follows:

$$S_b^{\Phi} = \sum_{i=1}^{2} n_i \left(\overline{u_i} - \overline{u}\right)\left(\overline{u_i} - \overline{u}\right)^{\top}. \tag{7}$$

And the within-class scatter matrix is

$$S_w^{\Phi} = \sum_{i=1}^{2}\sum_{j=1}^{n_i} \left(\Phi\left(x_j^i\right) - \overline{u_i}\right)\left(\Phi\left(x_j^i\right) - \overline{u_i}\right)^{\top}, \tag{8}$$

where $\overline{u_i} = (1/n_i)\sum_{j=1}^{n_i} \Phi(x_j^i)$ is class-conditional mean vector, $\overline{u}$ is mean vector of total instances, $\Phi(x_j^i)$ is the jth instances in the ith class, and n_i is the number of instances of the ith class. Then, the modified objective function is given as follows:

$$\Psi_{\text{opt}} = \arg\max J_{F(W)}^{\Phi} = \max \frac{\left|\Psi^T S_b^{\Phi} \Psi\right|}{\left|\Psi^T S_w^{\Phi} \Psi\right|}. \tag{9}$$

To maximize $J_{F(W)}^{\Phi}$, (9) can be transformed into a nonlinear eigenvalue problem [28]. Then, we can find the maximum eigenvalues of $(S_w^{\Phi} + \lambda I)^{-1} S_b^{\Phi}$, where λI is a regularizing diagonal term introduced to improve the numerical stability of the inverse computation as described by Shawe-Taylor and Cristianini [27].

Table 2: Data sets.

Project	Examples	%*Defective*	Description
NASA data sets			
pc1	705	9.66	Flight software
pc2	745	2.85	Flight software
pc3	1077	12.4	Flight software
pc4	1458	12.2	Flight software
kc2	522	20.49	Storage management
kc3	194	18.56	Storage management
cm1	327	12.84	Spacecraft instrument
mw1	253	10.67	A zero-gravity experiment
SOFTLAB data sets			
ar3	63	12.7	Embedded controller
ar4	107	18.69	Embedded controller
ar5	36	22.22	Embedded controller
ar6	101	16.0	Embedded controller

After the analysis as described above, we can get eigenvector vectors $\{t_q\}_1^m \in \Psi$ corresponding to the maximum eigenvalues of this eigenvalue problem. Finally, the original features can be projected to the new spaces by transformation matrix Ψ as follows:

$$T = \Psi \cdot \Phi(x) = \sum_{i=1}^{n} k(x_i, x). \quad (10)$$

In order to combat the class-imbalanced problem, we propose a kernel discrimination classification (KDC) based on local mean vector. Considering the correlation between transformed features and the class distribution, KDC retrieves the loss caused by the class-imbalanced problem. Firstly, we compute the X^i_{NEC}, neighbors of class i for every transformed instances $x \in T'_e$:

$$X^i_{\text{NEC}} = \left\{x_j \mid x_j \in T'_r,\ d\left(x, x_j\right) \leqslant d\left(x, x_K\right)\right\}, \quad (11)$$

where x_K is the kth nearest neighbor. When $k = 1$, KNN has the special form 1-NN rule. Then, we calculate the mean distances of each class:

$$D^i_{\text{NEC}} = \frac{1}{\left|X^i_{\text{NEC}}\right|} \sum_{x_j \in X^i_{\text{NEC}}} d\left(x, x_j\right). \quad (12)$$

Assign x to the class c if the distance between the local mean vector for c and the query pattern is minimum:

$$M(x) = \arg\min D^i_{\text{NEC}}. \quad (13)$$

KDC is summarized as in Algorithm 1. It originates from the need to combat the nonlinearly separable and class-imbalanced problem in the classification. It not only balances the distribution of data sets, but it also inherits the advantage of kernel method, which can conduct quite general dimensional feature space mappings.

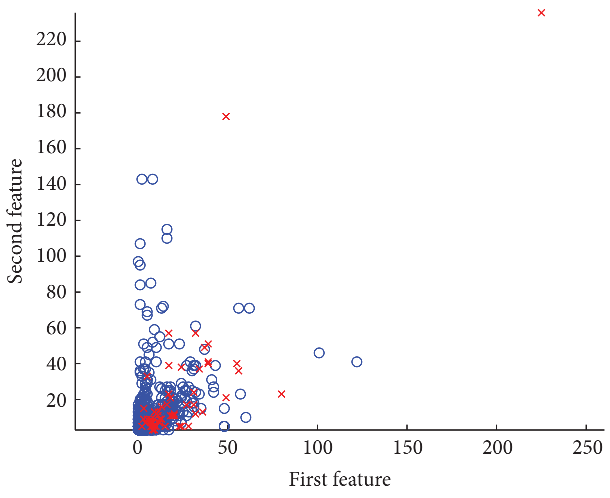
4. Experiments

In this section we evaluate KDC algorithm empirically. First of all, we use two types of discrimination methods LDA and KDA to analyze the well-known data sets used in the software defect prediction. And then, based on the analysis result, we investigate the performance of our method compared with the other three methods. We focus on the visualization and interpretation of the multivariable data so as to analyze the inherent property of the data sets used in the software defect prediction.

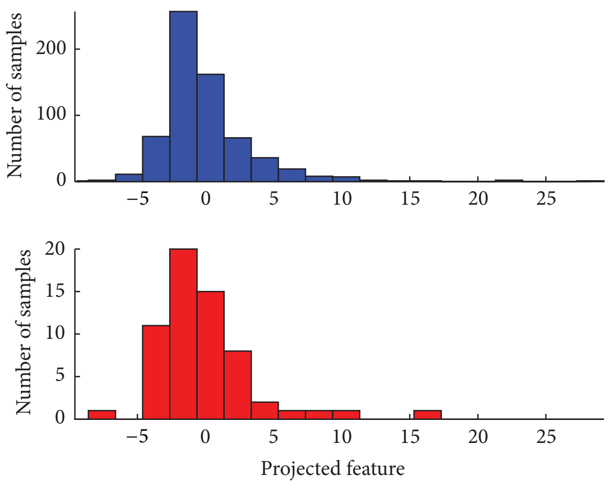
4.1. Data Set. In this study, twelve well-known data sets in software defect prediction are analyzed, including the eight sets used in [2] as well as four additional data sets used in [29], as shown in Tables 2 and 3. They are from NASA projects developed at different sites by different teams and from projects of Turkish software company (SOFTLAB) which is related to the embedded controller for white goods, respectively. Since these data sets are collected from different companies or different projects developed by different languages, they are under different distributions.

4.2. Discrimination Analysis on Software Data Sets. Firstly we conduct the discrimination on the data sets from NASA. Each Figure (a) shows the distributions of the defective modules and nondefective modules on two dimensions. Figures (b) and (c) depict the histograms of the first feature values obtained by LDA and KDA—the vertical axis corresponds to number of instances, and the horizontal axis to the project values on the first feature values. Note that the two dimensions of the data as shown in Figure (a) are the first two features, and the first features in Figures (b) and (c) are obtained from the projection from all the input features. The first 2D data of the original data pc1 is depicted in Figure 2(a). Since the distribution of the first two dimensions of the positive data and negative data is very similar, it is very hard to classify the two types without discrimination analysis. However, even projected onto one dimension using LDA, this data set is still mixed together as shown in Figure 2(b). Compared with LDA, when conducting KDA, we have found that the different patterns can be separated as shown in Figure 2(c). It means that the pc1 data set is a multimodel data set, which is nonlinearly separable. The result of the analysis on each data set of different companies contracted with NASA is very similar to this data set, as shown in Figures 2, 3, 4, 5, 6, 7, 8, and 9. In order to investigate the inherent property of the data sets used in software defect prediction, we also apply the discrimination analysis on the (SOFTLAB) data sets from local company, as shown in Figures 10, 11, 12, and 13. We can see that all the data sets have the property of nonlinear separability, which requires more sophisticated classification.

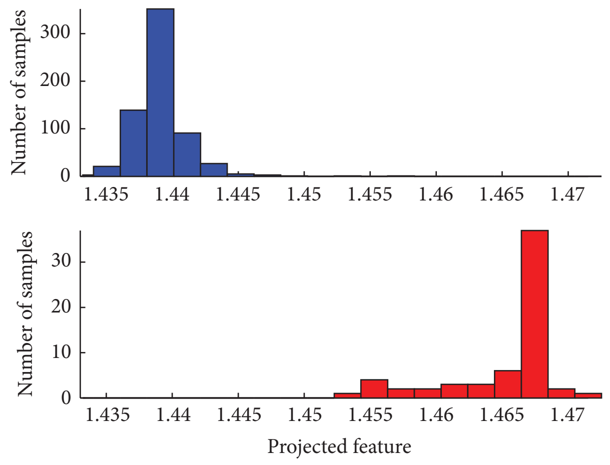
Each Figure (c) shows that KDA separates the software modules with defect from nondefect modules reasonably well. In addition to maximizing the between-class variance, KDA also tries to minimize the inner-class variance of each class. Another interesting finding from the figures is that



(a) First 2D data

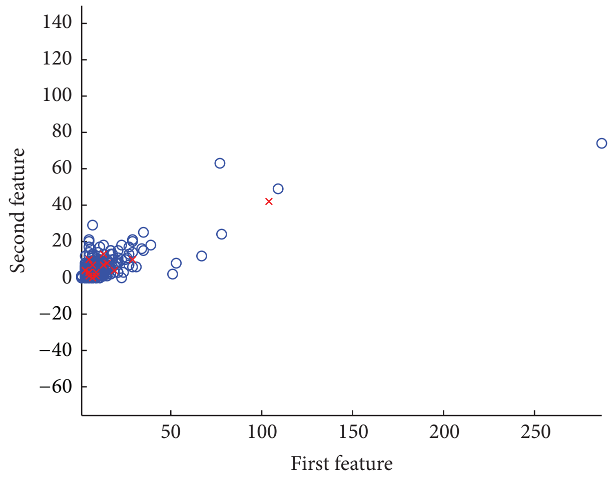


(b) LDA 1D project

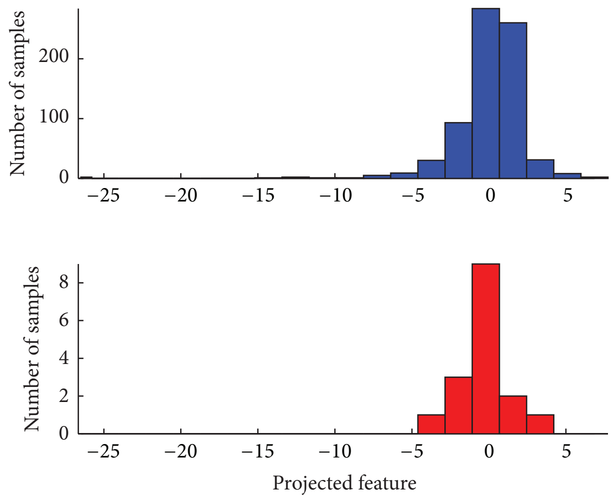


(c) KDA 1D project

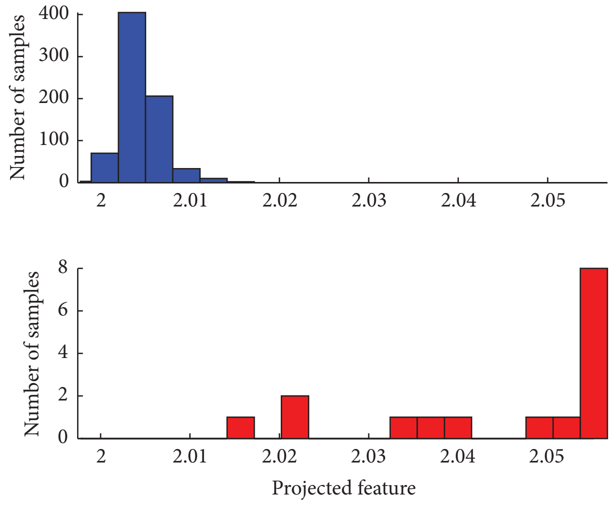
Figure 2: Discrimination analysis on pc1 data set.



(a) First 2D data

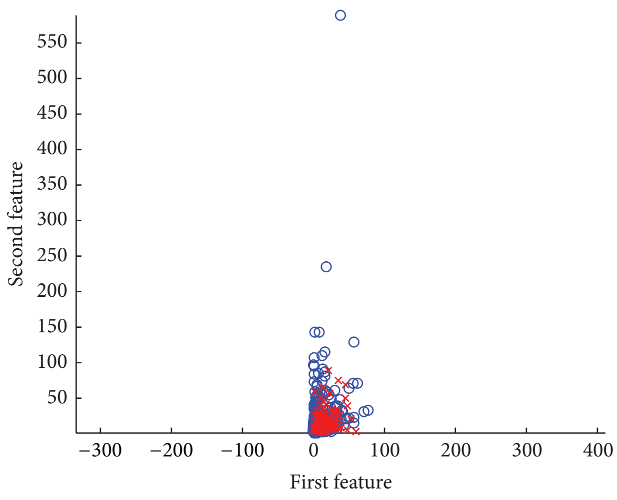


(b) LDA 1D project

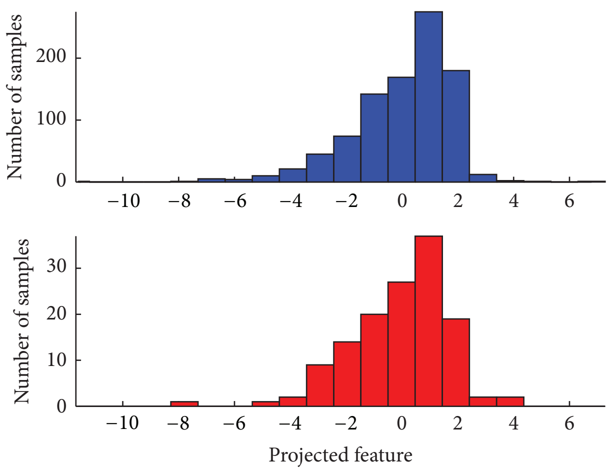


(c) KDA 1D project

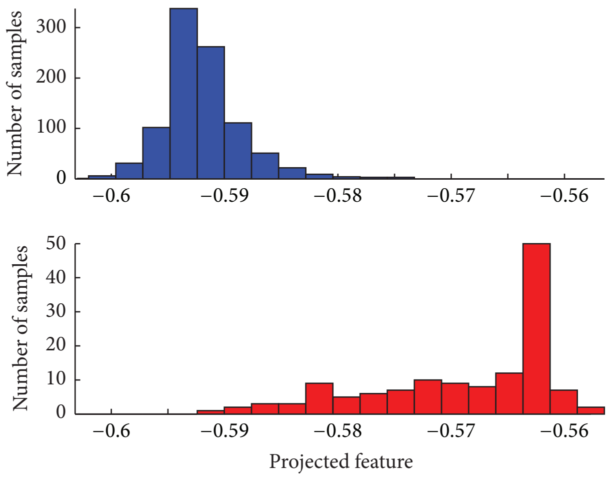
Figure 3: Discrimination analysis on pc2 data set.



(a) First 2D data

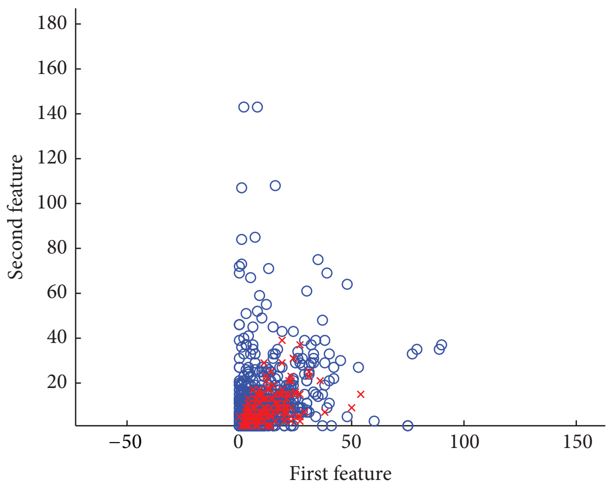


(b) LDA 1D project

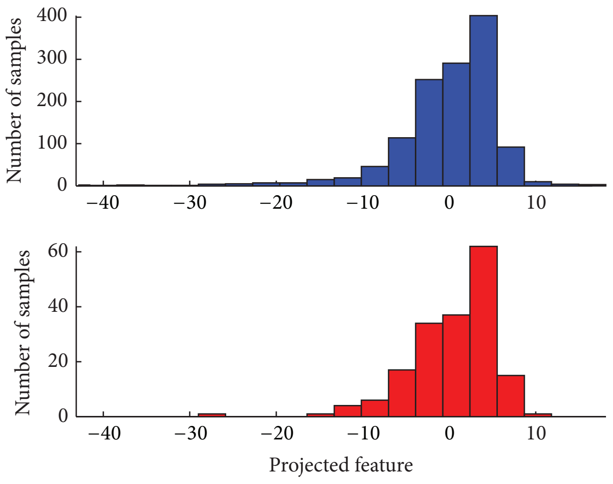


(c) KDA 1D project

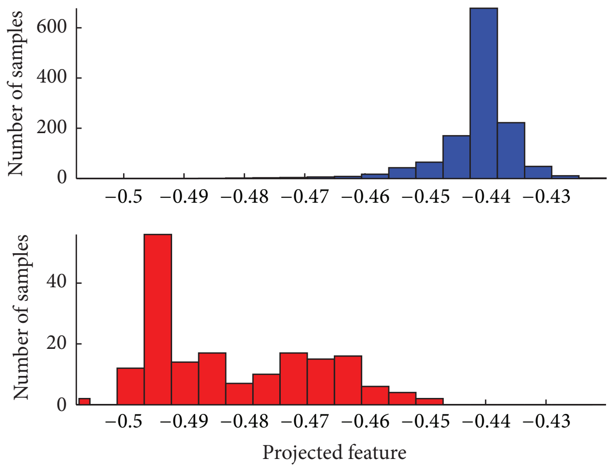
Figure 4: Discrimination analysis on pc3 data set.



(a) First 2D data

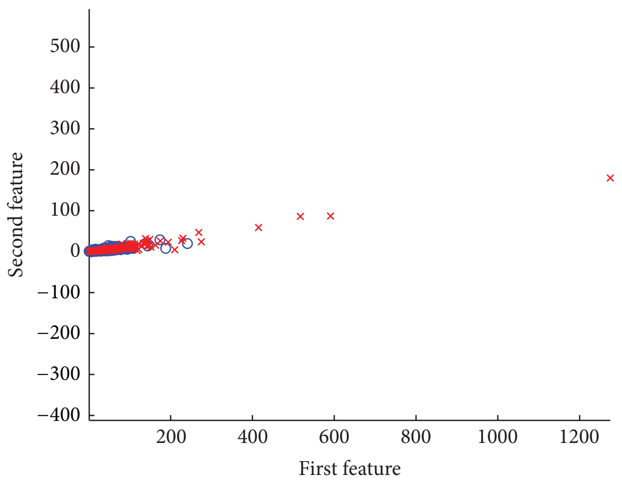


(b) LDA 1D project

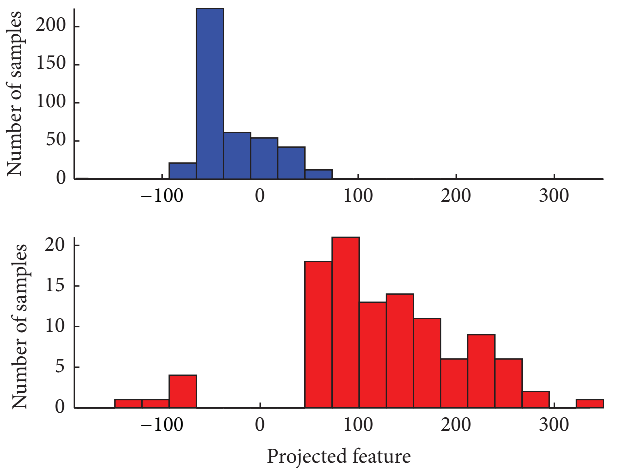


(c) KDA 1D project

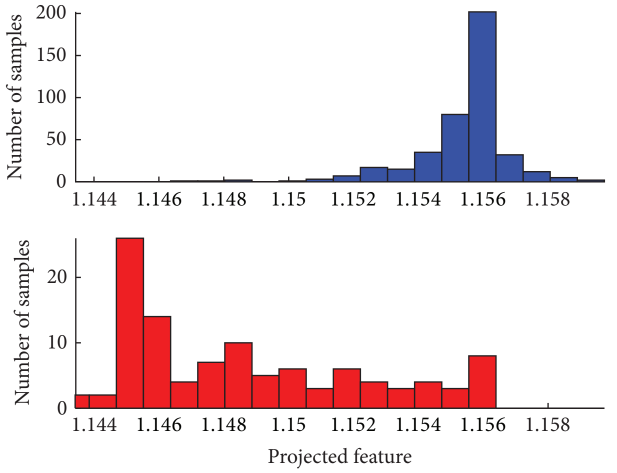
Figure 5: Discrimination analysis on pc4 data set.



(a) First 2D data

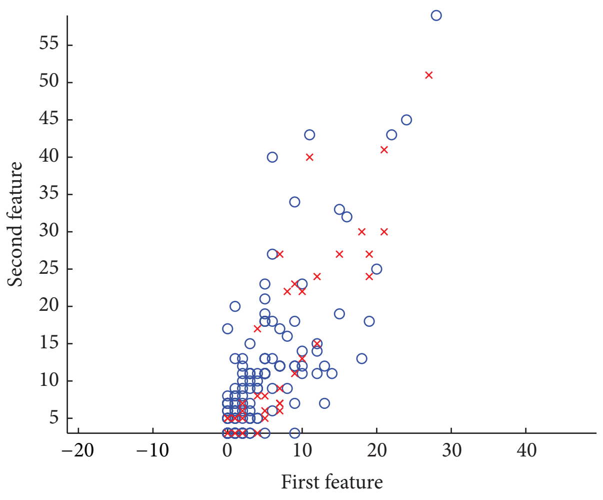


(b) LDA 1D project

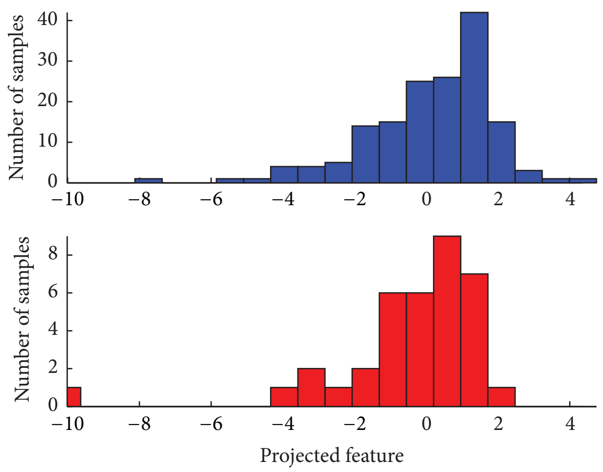


(c) KDA 1D project

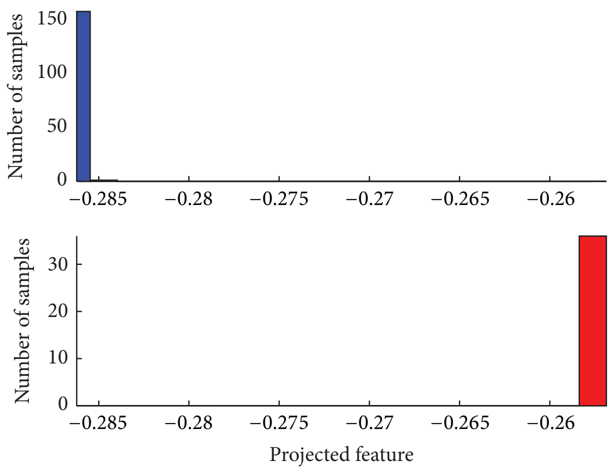
Figure 6: Discrimination analysis on kc2 data set.



(a) First 2D data

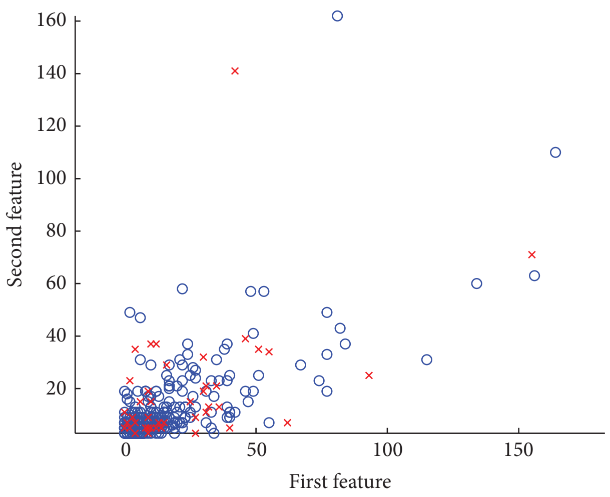


(b) LDA 1D project

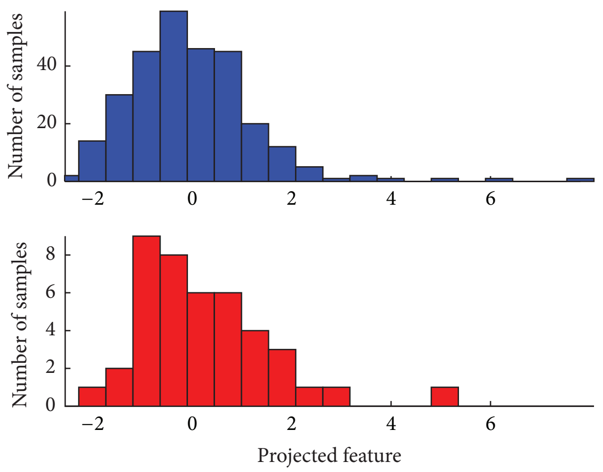


(c) KDA 1D project

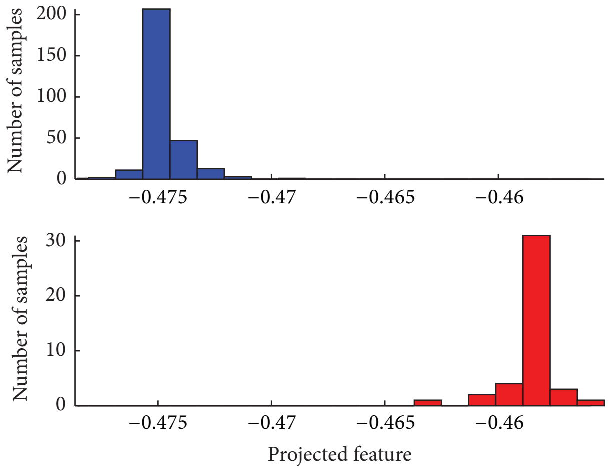
Figure 7: Discrimination analysis on kc3 data set.



(a) First 2D data

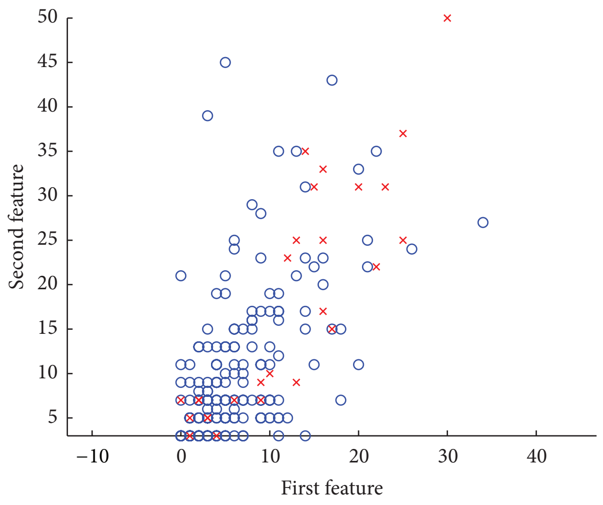


(b) LDA 1D project

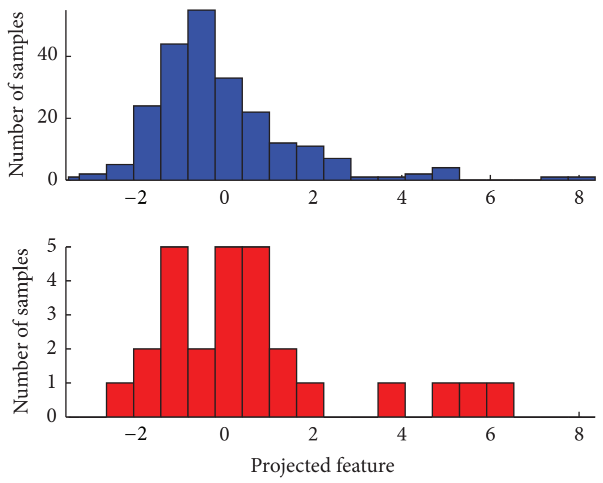


(c) KDA 1D project

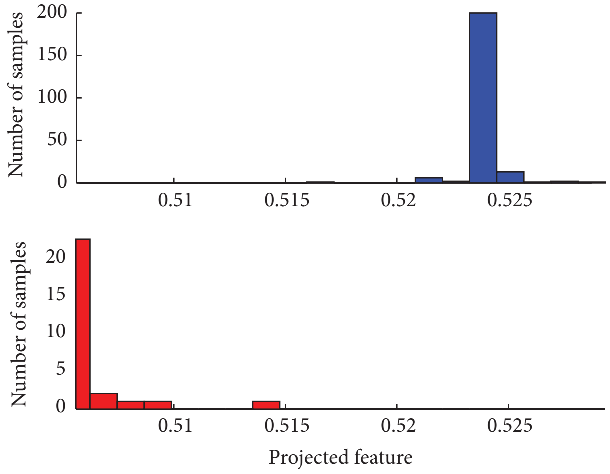
Figure 8: Discrimination analysis on cm1 data set.



(a) First 2D data

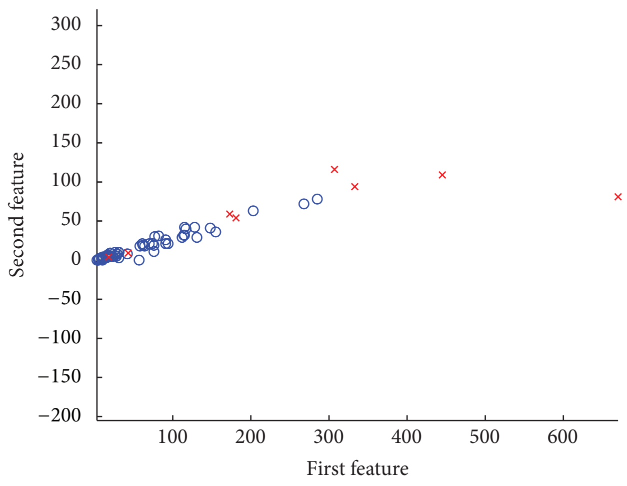


(b) LDA 1D project

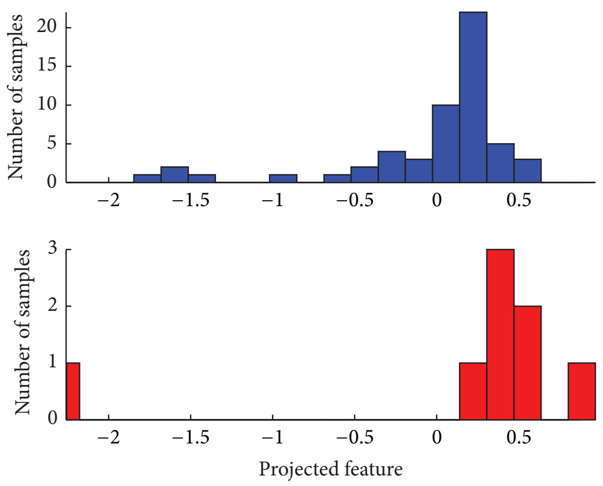


(c) KDA 1D project

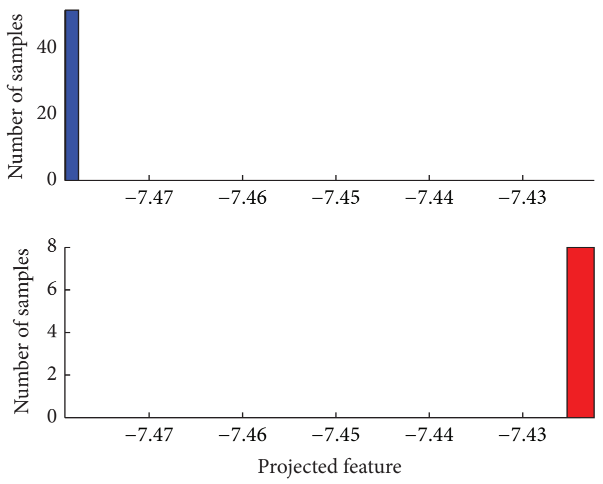
Figure 9: Discrimination analysis on mw1 data set.



(a) First 2D data

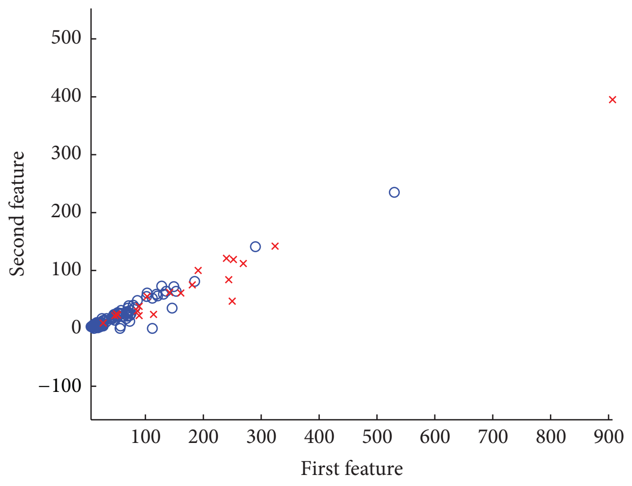


(b) LDA 1D project

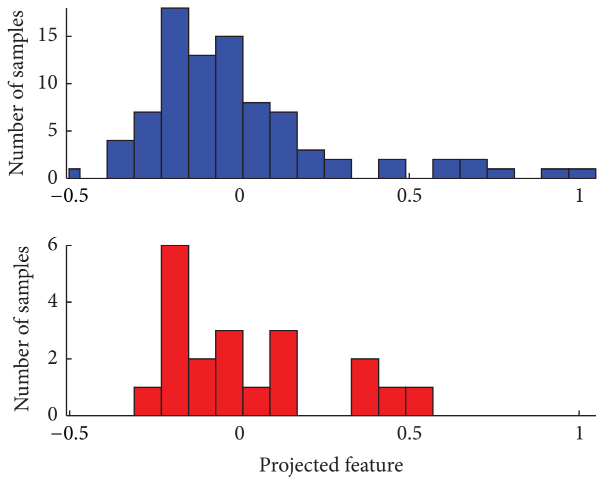


(c) KDA 1D project

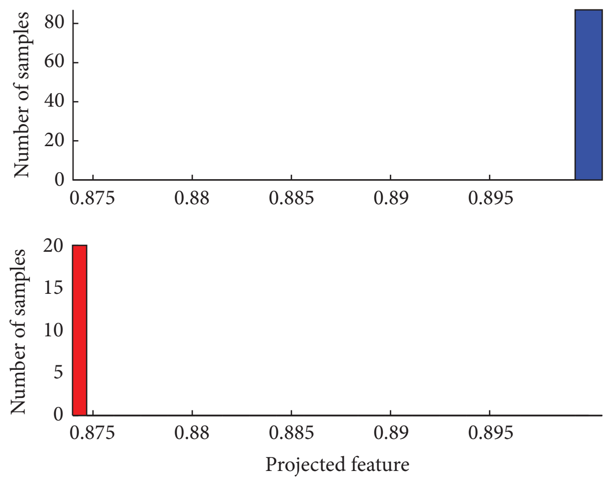
Figure 10: Discrimination analysis on ar3 data set.



(a) First 2D data

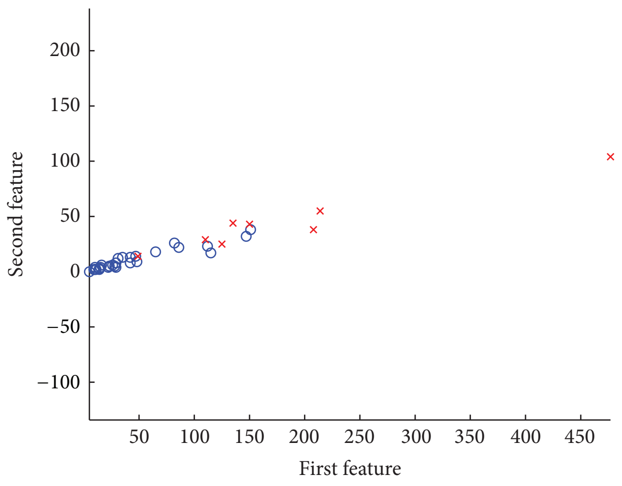


(b) LDA 1D project

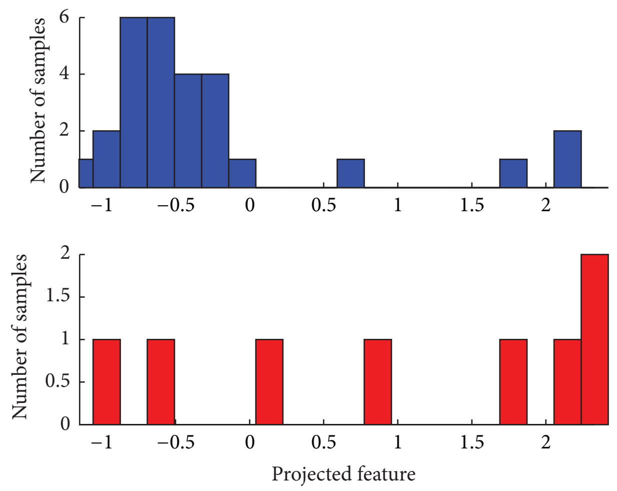


(c) KDA 1D project

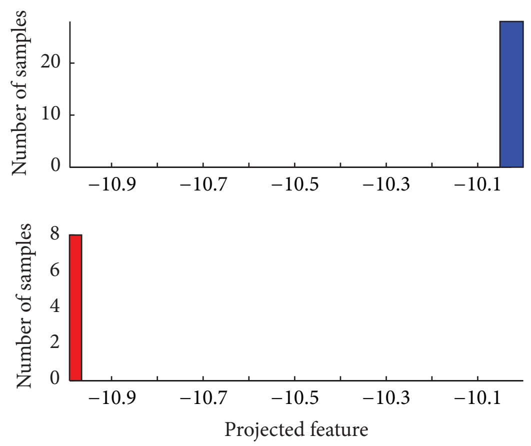
Figure 11: Discrimination analysis on ar4 data set.



(a) First 2D data

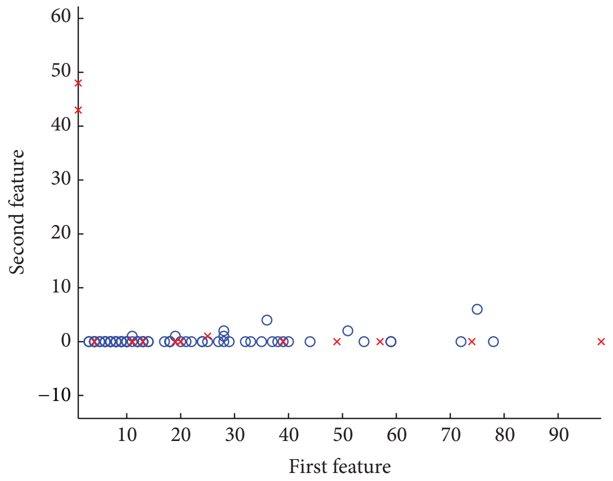


(b) LDA 1D project

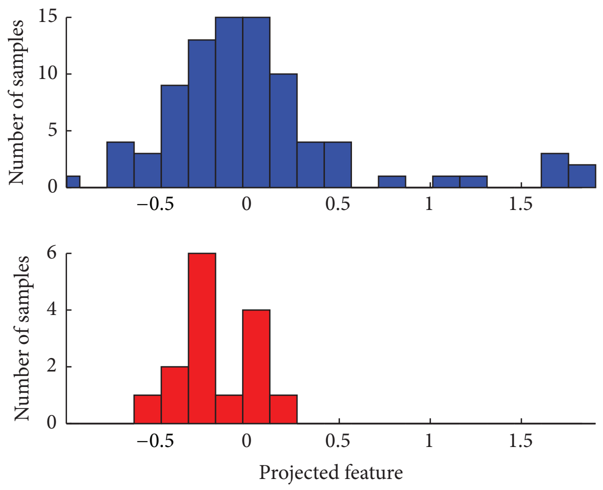


(c) KDA 1D project

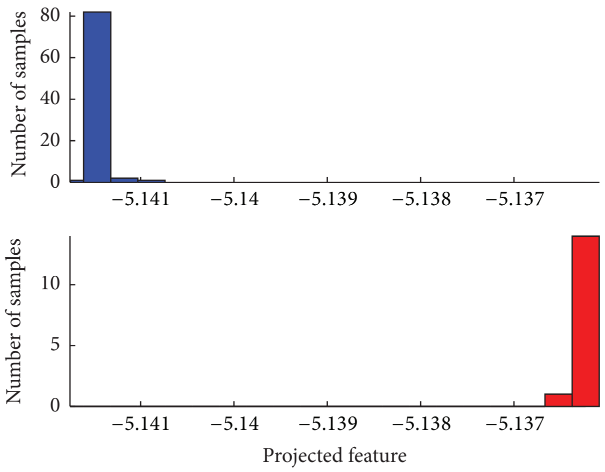
Figure 12: Discrimination analysis on ar5 data set.



(a) First 2D data



(b) LDA 1D project



(c) KDA 1D project

Figure 13: Discrimination analysis on ar6 data set.

Table 3: Software metrics.

Type	Description or formula
Mccabe	CYCLOMATIC_COMPLEXITY: $v(G) = e - n + 2$;
	ESSENTIAL_COMPLEXITY: $ev(G)$;
	DESIGN_COMPLEXITY: $iv(G)$;
	LOC_TOTAL;
Halstead	NUM_OPERANDS: N_1
	NUM_OPERATORS: N_2
	NUM_UNIQUE_OPERANDS: μ_1
	NUM_UNIQUE_OPERATORS: μ_2
	CONTENT: $I = \widehat{L} * V = (2/\mu_1) * (\mu_2/N_2)$
	DIFFICULTY: $D = 1/L$
	EFFORT: $E = v/L$
	ERROR_EST
	LENGTH: $N = N_1 + N_2$
	LEVEL: $V^\star/V = (2 + \mu_2^\star) \log_2 (2 + \mu_2^\star)$
	PROG_TIME: $T = E/18$ seconds
	VOLUME: $V = N\log_2(\mu)$
Loc	LOC_BLANK
	LOC_CODE_AND_COMMENT
	LOC_COMMENTS
	LOC_EXECUTABLE
	NUMBER_OF_LINES
Other	NODE_COUNT: number of nodes found in a given module, n
	EDGE_COUNT: words and phrases must be provided, e.
	BRANCH_COUNT
	CALL_PAIRS
	CONDITION_COUNT
	CYCLOMATIC_DENSITY
	DECISION_COUNT: number of decision points in a given module
	DESIGN_DENSITY: $iv(G)/v(G)$
	ESSENTIAL_DENSITY: $(ev(G) - 1)/(v(G) - 1)$
	PARAMETER_COUNT
	GLOBAL_DATA_COMPLEXITY
	GLOBAL_DATA_DENSITY
	MAINTENANCE_SEVERITY: $ev(G)/v(G)$
	MODIFIED_CONDITION_COUNT
	MULTIPLE_CONDITION_COUNT
	NORMALIZED_CYLOMATIC_COMPLEXITY
	PERCENT_COMMENTS

the first feature obtained by KDA has a strong positive correlation to the defective modules. While LDA is incapable of providing correct classification because of its linear nature, KDA can usually provide correct classification through nonlinear transformations. Therefore, it is able to produce linear separable features for such data that are from the input space and have bad linear separability. KDA is to find a nonlinear projection direction, by which the original inputs can be mapped into a high dimension feature space, where they were linearly separable, and then the LDA was employed.

KDA produces a nonlinear decision boundary, which is very useful in defect prediction since classes are not always well separated by a linear function. After transforming the low-dimensional input space into a high-dimensional feature space, the data set in the new feature space becomes linearly separable. Theoretically speaking, the kernel function is able to implicitly and explicitly map the input space, which may not be linearly separable, into an arbitrary high-dimensional feature space that can be linearly separable. What is more, we calculate the local mean distance for each class to combat the class-imbalanced problem, as shown in the Algorithm 1. The performances of KDC can also be seen from the following experiment.

4.3. Performance Measures. To evaluate the performance of the prediction model, we can use the confusion matrix of the predictor from Witten and Frank [30]. In the confusion matrix, True Positive (TP) is the number of defective modules predicted as defective; False Negative (FN) is the number of defective modules predicted as nondefective; False Positive (FP) is the number of nondefective modules predicted as defective; True Negative (TN) is the number of nondefective modules predicted as nondefective.

Since F-measure [31] value serves as a good singular performance metric when dealing with the class-imbalanced problem, it is widely used in the software defect prediction field [32, 33]. It can be expressed as follows:

$$F\text{-measure} = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}}, \tag{14}$$

where recall = TP/(TP + FN) is the probability of true defective modules to the number of defective modules, and precision = TP/(TP + FP) is the probability of true defective modules to the number of modules predicted as defective.

4.4. Result. In order to investigate the performance of KDC (Gaussian kernel $K(x, y) = \exp(-\|x - y\|^2)$ is used here), we compare it with J4.8 decision trees [15, 34], Naive Bayes [2], random forest (RF) [35], AdaBoost [17], Smote [36], and linear discrimination analysis based classifier (LDC) [26]. The details are as follows.

(i) Under each labeled rate, each data set is divided into ten random partitions.

(ii) The defect predictor is built from nine partitions and tested on the remaining partition for each method.

(iii) Running ten times follows the steps above.

(iv) For comparing the results for these methods, we conducted Mann-Whitney U-Test (Mann-Whitney U test is a nonparametric statistical hypothesis test to compare two independent groups of sampled data, which is without an assumption of a normal distribution. For details see [37]. That is, we speak of two

```
**Require**:
    Normalized training data, $T_r$;
    Labels vector of the training data, $Y$;
    Normalized test data, $T_e$;
    Kernel type, kernel;
**Ensure**:
    KDC classifier, $M$;
(1) $K = \text{kernelize}\,(T_r, T_r, kernel, kernelparam)$;
    $K_t = \text{kernelize}\,(T_r, T_e, kernel, kernelparam)$;
(2) $\ell = size\ (K, 1)$;
(3) $\ell^+ = (\sum Y + \ell)\,/2$;
(4) $Y^+ = 0.5 * (Y + 1)$;
(5) $ell^- = \ell - \ell^+$;
(6) $Y^- = Y^+ - Y$;
(7) $t = size\,(K_t, 2)$;
(8) $\Upsilon = ones\,(\ell, 1) + Y * ((\ell^- - \ell^+)\,/\ell^{-1})$ ;
(9) $\tau^+ = 2 * \ell^-/(\ell * \ell^+)$;
(10) $\tau^- = 2 * \ell^+/(\ell * \ell^-)$ ;
(11) $B = \text{diag}\,(\Upsilon) - (\tau^+ * Y^+) * Y^+ - (\tau^- * Y^-) * Y^-$;
(12) $T = (B * K + \lambda * eye\,(\ell, \ell))\,Y^{-1}$;
(13) $T_r' = T * T_r, T_e' = T * T_e$;
(14) Collecting the neighbourhoods of $x' \in T_e$ using (12);
(15) Calculate the mean distances using (13);
(16) Classify $x$ as $M\,(x) = \arg\min\ D^i_{\text{NEC}}$;
(17) **return** $M$;
```

Algorithm 1: Kernel discrimination classifier (KDC).

Table 4: Statistical F-measure values of six classifiers on all data sets. The line $w/t/l$ means that the algorithm at the corresponding KDC wins in w data sets, ties in t data sets, and loses in l data sets, compared with the algorithm at the corresponding column.

Project	J4.8	Naive Bayes	RF	AdaBoost	Smote	KDA	LDC
ar3	0.500	0.667	0.633	0.377	0.455	0.702	0.500
ar4	0.474	0.514	0.487	0.443	0.457	0.560	0.485
ar5	0.625	0.667	0.490	0.464	0.575	0.650	0.533
ar6	0.105	0.357	0.254	0.237	0.203	0.201	0.087
pc1	0.271	0.344	0.302	0.347	0.393	0.470	0.031
pc2	0.110	0.055	0.140	0.101	0.100	0.150	0.105
pc3	0.302	0.262	0.357	0.340	0.401	0.436	0.121
pc4	0.503	0.418	0.401	0.437	0.423	0.450	0.342
mw1	0.195	0.390	0.367	0.220	0.283	0.357	0.340
kc2	0.522	0.511	0.474	0.530	0.517	0.650	0.492
kc3	0.357	0.406	0.306	0.328	0.336	0.433	0.433
cm1	0.293	0.256	0.277	0.244	0.217	0.397	0.05
$w/t/l$	10/0/2	8/1/3	10/1/1	6/4/2	7/5/0	—	10/0/2

results for a data set as being "significantly different" only if the difference is statistically significant at the 0.05 level according to the Mann-Whitney U-Test.) with level of significance: 5% ($P = 0.05$).

We calculate the means and variances of F-measure values of running 10 times' results for these methods, as summarized in Table 4. It shows that on all the data sets, KDC achieves higher F-measure values than J4.8 significantly. Although all the methods fail to build practical predictors on data sets such as pc2, we can still consider that the KDA has the best performance. Note that this data set has an extreme imbalance ratio, which is too low (only 21 defective modules in 745 modules) to express the information of the defective modules.

4.5. Threats to Validity. As every empirical experiment, our results are subject to some of the potential threats to validity. Firstly, in this study, we validated our findings on open data sets with different characteristics, from two different organizations, that is, NASA and SOFTLAB. By doing so, we have gained more confidence in the validity of the results reported in this paper. Secondly, since systems are developed for different domains or different applications, someone could think more about the ability of application of our method in industrial practice. Therefore, the replicated studies examining our method on other software systems will be useful to generalize our findings and improve our method. Finally, to the best of our knowledge, there is little current negative criticism for Mann Whitney as a statistical test for comparing data miners, but many for others. Although this is often misunderstood as a criticism of empirical studies, this study shows encouraged results with kernel discrimination analysis method. Our method should encourage more researchers to run similar studies on more kernel based method and deepen the understanding of the inherent property of the software engineering data. Our study would be replicated with more projects, different metrics, and replaced by more sophisticated method.

5. Conclusion and Future Work

In this paper, we addressed the issue of analysis of the property of the data sets in software defect prediction. By conducting the linear discrimination analysis on software data sets, we found that these data could not be separated by LDA. The results projected by LDA and KDA showed that these data sets had a property of nonlinear separability. Motivated by the analysis, we proposed a new algorithm KDC based on the kernel discrimination analysis to build defect predictor. It can tackle the nonlinearly separable and class-imbalanced problems. Experiments show that KDC can give good performances among the comparative methods on the test sets.

There are several areas in which we can improve this work. First, when we try to analyze the data structure of the software data sets, we only use one type of discrimination methods. However, we may try other data analysis methods to get more information of the data for building defect predictors. Second, in the future we will try to investigate other kernel based algorithms for software defect prediction on more software data sets.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was partially supported by NSFC (61070151, 61373147) and Xiamen Scientific Research Foundation (3502Z20123037). The authors thank the anonymous reviewers for their great helpful comments.

References

[1] T. M. Khoshgoftaar, X. Yuan, and E. B. Allen, "Balancing misclassification rates in classification-tree models of software quality," *Empirical Software Engineering*, vol. 5, no. 4, pp. 313–330, 2000.

[2] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[3] J. Fan, Y. Feng, and X. Tong, "A road to classification in high dimensional space: the regularized optimal affine discriminant," *Journal of the Royal Statistical Society B*, vol. 74, no. 4, pp. 745–771, 2012.

[4] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Computer Science and Scientific Computing, Academic Press, Boston, Mass, USA, 2nd edition, 1990.

[5] A. Samant and H. Adeli, "Feature extraction for traffic incident detection using wavelet transform and linear discriminant analysis," *Computer-Aided Civil and Infrastructure Engineering*, vol. 15, no. 4, pp. 241–250, 2000.

[6] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[7] K. Torkkola, "Linear discriminant analysis in document classification," in *Proceedings of the IEEE ICDM Workshop on Text Mining*, pp. 800–806, 2001.

[8] M. Sakai, N. Kitaoka, and K. Takeda, "Acoustic feature transformation based on discriminant analysis preserving local structure for speech recognition," *IEICE Transactions on Information and Systems*, vol. 93, no. 5, pp. 1244–1252, 2010.

[9] T. V. Bandos, L. Bruzzone, and G. Camps-Valls, "Classification of hyperspectral images with regularized linear discriminant analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 3, pp. 862–873, 2009.

[10] J. B. Li, S. C. Chu, and J. S. Pan, "Kernel discriminant analysis based face recognition," in *Kernel Learning Algorithms for Face Recognition*, pp. 101–133, Springer, New York, NY, USA, 2014.

[11] C. Moulin, C. Largeron, C. Ducottet, M. Géry, and C. Barat, "Fisher linear discriminant analysis for text-image combination in multimedia information retrieval," *Pattern Recognition*, vol. 47, no. 1, pp. 260–269, 2014.

[12] A. Shrivastava, H. V. Nguyen, V. M. Patel, and R. Chellappa, "Design of non-linear discriminative dictionaries for image classification," in *Computer Vision–ACCV 2012*, pp. 660–674, Springer, Berlin, Germany, 2013.

[13] W. J. Lin and J. J. Chen, "Class-imbalanced classifiers for high-dimensional data," *Briefings in Bioinformatics*, vol. 14, no. 1, pp. 13–26, 2013.

[14] L. C. Briand, V. R. Basili, and C. J. Hetmanski, "Developing interpretable models with optimized set reduction for identifying high-risk software components," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1028–1044, 1993.

[15] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Classification tree models of software quality over multiple releases," in *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, pp. 116–125, November 1999.

[16] N. Fenton, M. Neil, W. Marsh et al., "Predicting software defects in varying development lifecycles using Bayesian nets," *Information and Software Technology*, vol. 49, no. 1, pp. 32–43, 2007.

[17] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics A*, vol. 39, no. 6, pp. 1283–1294, 2009.

[18] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[19] Y. Ma, G. Luo, and H. Chen, "Kernel based asymmetric learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 95, no. 1, pp. 267–270, 2012.

[20] G. Luo, Y. Ma, and Q. Ke, "Asymmetric learning based on kernel partial least squares for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 95, no. 7, pp. 2006–2008, 2012.

[21] G. Luo, Y. Ma, and Q. Ke, "Active learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 95, no. 6, pp. 1680–1683, 2012.

[22] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE'08)*, pp. 47–54, May 2008.

[23] N. Ohlsson, M. Zhao, and M. Helander, "Application of multivariate analysis for software fault prediction," *Software Quality Journal*, vol. 7, no. 1, pp. 51–66, 1998.

[24] J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 423–433, 1992.

[25] E. Kocaguneli, T. Menzies, and J. W. Keung, "Kernel methods for software effort estimation," *Empirical Software Engineering*, vol. 18, no. 1, pp. 1–24, 2011.

[26] G. J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*, Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics, John Wiley & Sons, New York, NY, USA, 1992.

[27] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, New York, NY, USA, 2004.

[28] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Muller, "Fisher discriminant analysis with kernels," in *Proceedings of the 9th IEEE Workshop on Neural Networks for Signal Processing (NNSP'99)*, pp. 41–48, August 1999.

[29] A. T. Misirli, A. B. Bener, and B. Turhan, "An industrial case study of classifier ensembles for locating software defects," *Software Quality Journal*, vol. 19, no. 3, pp. 515–536, 2011.

[30] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, San Francisco, Calif, USA, 2nd edition, 2005.

[31] C. J. van Rijsbergen, *Information Retireval*, Butterworths, London, UK, 2nd edition, 1979.

[32] H. Zhang and X. Zhang, "Comments on 'Data mining static code attributes to learn defect predictors'," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 635–636, 2007.

[33] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM'08)*, pp. 346–355, Beijing, China, October 2008.

[34] A. G. Koru and H. Liu, "Building effective defect-prediction models in practice," *IEEE Software*, vol. 22, no. 6, pp. 23–29, 2005.

[35] A. Kaur and R. Malhotra, "Application of random forest in predicting fault-prone classes," in *Proceedings of the International Conference on Advanced Computer Theory and Engineering (ICACTE'08)*, pp. 37–43, Phuket, Thailand, December 2008.

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[37] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.