Hindawi
Complexity
Volume 2017, Article ID 2691474, 14 pages
https://doi.org/10.1155/2017/2691474

Research Article

Kernel Negative ε Dragging Linear Regression for Pattern Classification

Yali Peng,[1,2,3] Lu Zhang,[1,2] Shigang Liu,[1,2,3] Xili Wang,[1,2] and Min Guo[2,3]

[1]*Key Laboratory of Modern Teaching Technology, Ministry of Education, Xi'an 710062, China*
[2]*Engineering Laboratory of Teaching Information Technology of Shaanxi Province, Xi'an 710119, China*
[3]*School of Computer Science, Shaanxi Normal University, Xi'an 710119, China*

Correspondence should be addressed to Shigang Liu; shgliu@snnu.edu.cn

Received 27 August 2017; Accepted 9 November 2017; Published 10 December 2017

Academic Editor: Chuan Zhou

Linear regression (LR) and its variants have been widely used for classification problems. However, they usually predefine a strict binary label matrix which has no freedom to fit the samples. In addition, they cannot deal with complex real-world applications such as the case of face recognition where samples may not be linearly separable owing to varying poses, expressions, and illumination conditions. Therefore, in this paper, we propose the kernel negative ε dragging linear regression (KNDLR) method for robust classification on noised and nonlinear data. First, a technique called negative ε dragging is introduced for relaxing class labels and is integrated into the LR model for classification to properly treat the class margin of conventional linear regressions for obtaining robust result. Then, the data is implicitly mapped into a high dimensional kernel space by using the nonlinear mapping determined by a kernel function to make the data more linearly separable. Finally, our obtained KNDLR method is able to partially alleviate the problem of overfitting and can perform classification well for noised and deformable data. Experimental results show that the KNDLR classification algorithm obtains greater generalization performance and leads to better robust classification decision.

1. Introduction

Least squares regression (LSR) has been widely used for many fields of pattern recognition and computer vision. Owing to LSR being mathematically tractable and computationally efficient, in the past, many variants have been proposed. Notable LSR algorithms include weighted LSR [1], partial LSR [2], and other extensions (e.g., nonnegative least squares (NNLS) [3]). In the pattern recognition community, LSR is also referred to as minimum squared error algorithm [4–6]. Moreover, very competent extensions of least squares regression such as regularized least squares regression [7] are also proposed. Among extensions of least squares regression, sparse regression [8] and low-rank regression [9, 10] can obtain notable performance. The relationship between the regression and other methods such as locally linear embedding and local tangent space alignment is also studied [11]. In addition, LSR is also applied to semisupervised learning. Nie et al. [12] proposed adaptive loss minimization for semisupervised elastic embedding. Fang et al. [13] proposed learning a nonnegative sparse graph for linear regression for semisupervised learning, in which linear regression and graph learning were simultaneously performed to guarantee an overall optimum.

LSR can be simply described as follows. Before conventional least squares regression (CLSR) is applied for classification [3, 14, 15], it assigns different fixed class labels to training samples of different classes. Then it employs the least squares regression algorithm to achieve a mapping that is able to transform training samples into approximations of their class labels. Finally CLSR uses the obtained mapping to predict the class label of every test sample. In addition to classification problems, least squares regression is also applied to subspace segmentation [16], matrix recovery [17], and feature selection [18].

The sparse representation classification (SRC) [19–21], recently proposed, can be regarded as a special form of least squares regression. Differing from LSR, it achieves an approximation of a test sample via a sparse linear combination of all training samples. Also collaboration representation [22] and

linear regression classification [23] are similar. An overview of sparse representation is provided in [24]. However, for classification tasks, because SRC must solve a set of equations for classifying every sample, CLSR is computationally much more efficient than SRC.

Xiang et al. proposed discriminative least squares regression (DLSR) [25]. The core idea is, under the conceptual framework of least squares regression, to achieve a larger class margin than the class margin obtained using CLSR for classification algorithms by using the ε dragging technique, which plays a similar role in enlarging the margin as other large margin classifiers proposed in [26–28]. The idea of using slack variable to relax the model has been widely used in the related field [29]. When the distribution of training samples is in accordance with that of test samples, the classifier learned from training samples can well adapt to test samples. Under the condition, since the classifier learned from training samples has a very large class margin, it can also obtain a satisfactory class margin for test samples. Accordingly the original ε dragging technique can perform well. In other words, a high classification accuracy can be produced. However, in real-world applications, owing to the noise or deformability of the object, the difference between training samples and test samples from the same class may be much. For example, it is well known that face images are a kind of deformable objects (owing to varying poses, expressions, and illumination conditions). Two-face images from the same subject have much difference. This difference may be even greater than that of two-face images obtained from two distinctive subjects. In this case, a large margin classifier obtained by using training samples is not usually suitable for test samples. In other words, it probably performs badly in classifying the test samples. On the contrary, reducing the class margin usually achieves better classification accuracy for classification problems on noised data. Thus, we focus on determining a proper margin by using the negative ε dragging technique and producing a robust classifier for pattern classification on noised and deformable data.

Furthermore, we focus on introducing the kernel trick to improve the ε dragging linear regression. In machine learning, the kernel trick is originally utilized to construct nonlinear support vector machines (SVMs) [30–32]. In the last more than 10 years, many kernel based approaches have been proposed, such as well-known kernel principal component analysis (KPCA) [33, 34] and kernel Fisher discriminant analysis (KFDA) [35]. For classification, Yu et al. presented the kernel nearest neighbor (KERNEL-NN) classifier [36]. KERNEL-NN applies the nearest neighbor classification method in the high dimensional feature space. The KERNEL-NN classifier could perform better than the NN classifier by utilizing an appropriate kernel. Kernel sparse representation classification (KSRC) is also presented [37, 38]. So far, by using kernel tricks [39], almost all linear learning methods can be generalized to the corresponding nonlinear ones. The kernel trick [40] goes a large step toward the goal of classifying heterogeneous data. These kernel based algorithms improve the computational ability of the linear algorithms. They first implicitly map the data in the input space into a high or even infinite dimensional kernel feature space [18, 41] by a nonlinear mapping and then perform linear processing in the kernel feature space by using the inner products, which can be computed by a kernel function. As a result, these kernel based algorithms perform a nonlinear transformation with respect to the input space.

As is well known, kernel approach can change the distribution of samples by the nonlinear mapping. If an appropriate kernel function is utilized, kernel approach is able to make the data of different classes more linearly separable. Therefore, kernel based algorithms can perform classification well. This motivated us to integrate kernel method into linear regression for classification. If an appropriate kernel function is utilized, more samples from the same class are close to each other and samples from distinct classes are far from each other in the high dimensional feature space. Hence, in the high dimensional feature space, it is easy to learn a mapping that can well convert training samples into their class labels. Namely, linear transformation matrix learned in the high dimensional feature space can more appropriately map samples into their class labels and has more powerful discriminating ability.

Based on the above two aspects, we propose the kernel negative ε dragging linear regression (KNDLR) method in this paper. For KNDLR, samples are implicitly mapped into a high dimensional feature space first, and then linear regression with the negative ε dragging is performed in this new feature space. We prove that KNDLR in the high dimensional feature space can be formulated in terms of the inner products, while the inner products could be computed by kernel function. Thus KNDLR is easy to be implemented and has low computation cost. The classifier can generalize well because we propose and use the negative ε dragging technique, and kernel approach is also integrated into KNDLR. Comprehensive experiments demonstrated the superior characteristics of KNDLR. In summary, the contributions of the proposed method are as follows.

(1) It relaxes the strict binary label matrix that is used in conventional LR into a slack variable matrix which has more freedom to fit the sample. The proper margins between different classes are achieved by using the negative ε dragging technique. Previously researchers usually focus on enlarging the margin between different classes, whereas the negative ε dragging technique proposed by us seems to be a new contrary idea, which is useful to overcome the overfitting problem and to enhance the robustness of the algorithm on unseen samples, for example, test samples.

(2) The kernel approach is also integrated into our method. We show that KNDLR in the high dimensional feature space can be formulated in terms of the inner products, and the inner products could be computed by the kernel function. Thus KNDLR only needs to calculate the kernel function rather than directly calculating data in the high dimensional feature space corresponding to the kernel function.

(3) An algorithm named KNDLR is devised for the proposed method. The validity of the algorithms is tested on six image datasets.

The other parts of the paper are organized as follows. Section 2 briefly reviews works related to this paper. In

Section 3, our method is presented. In Section 4, analysis of our method is provided. Experimental results are reported in Section 5. Finally, Section 6 offers the conclusion of this paper.

2. Related Works

In this section, we first introduce the CLSR for classification. Then, the kernel trick is briefly reviewed.

2.1. Conventional Least Squares Regressions for Classification. The collection of n training samples is represented as a matrix $X = [x_1, \ldots, x_n]^T \subset R^{n\times m}$. x_i is a training sample in the form of column vector. If the training sample is a two-dimensional image, then it is converted into one column vector in advance. The objective function of conventional least squares regression (CLSR) for classification is as follows:

$$\min_{W} \quad \|XW - Y\|_F^2 + \lambda \|W\|_F^2, \tag{1}$$

where $Y = [y_1, \ldots, y_n]^T \subset R^{n\times c}$ ($c \geq 2$ is the number of class) is the binary class label matrix and the ith row y_i of Y is the class label vector of the ith sample.

For a three-class classification problem, in CLSR the class label matrix of four samples may be

$$Y = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 3}. \tag{2}$$

Y indicates that the first and second samples are from the first class, the third sample is from the third class, and the fourth sample is from the second class. W is the transformation matrix which converts the sample matrix X into the class label binary matrix Y. $\|\bullet\|_F^2$ stands for Frobenius norm of matrix. In the above CLSR for classification, the class label is predefined and fixed.

2.2. Kernel Trick. The kernel trick is a very powerful technique in machine learning. It has been successfully applied to many methods, such as SVM [31, 32], KPCA [33, 34], and KFDA [35]. By using kernel tricks, a linear algorithm can be easily generalized to a nonlinear algorithm.

Mercer kernel is generally used in kernel methods. It is a continuous, symmetric, positive semidefinite kernel function. Given a Mercer kernel $k : \chi \times \chi \to \mathbb{R}$, there is a unique associated reproducing kernel Hilbert space (RKHS) H_k. Usually, a Mercer kernel can be expressed as

$$k\left(x, x'\right) = \Phi\left(x\right)^T \Phi\left(x'\right), \tag{3}$$

where T denotes the transpose of a matrix or vector, x and x' are any two points in χ, and Φ is the implicit nonlinear mapping associated with the kernel function $k(\cdot, \cdot)$. When implementing kernel methods, we do not need to know what Φ is and just adopt the kernel function defined as (3). Here the kernel function is the connection between the learning algorithm and data. The linear kernels, polynomial kernels, Gaussian radial basis function (RBF) kernels, and wavelet kernels [18, 40, 41] are commonly used kernels in kernel methods. The polynomial kernel has the form of

$$k\left(x, x'\right) = \left(x^T x' + c\right)^d, \tag{4}$$

where c is a constant, d is the order of polynomial, and RBF kernels can be expressed as

$$k\left(x, x'\right) = \exp\left(-\gamma \left\|x - x'\right\|_2^2\right), \tag{5}$$

where γ is the parameter for RBF kernels and $\|x - x'\|_2$ is the distance between two vectors.

3. Our Method

3.1. Solving the Optimization Model. Training samples $\{x_1, x_2, \ldots, x_n\}$ in the input space χ are represented as a matrix $X = [x_1, \ldots, x_n]^T \subset R^{n\times m}$. Let Φ be the nonlinear mapping function corresponding to a kernel $k(\cdot, \cdot)$. Firstly, we implicitly employ Φ to map the data from input space χ to a high dimensional kernel feature space R^f. We have

$$\Phi\left(X\right) = \left[\Phi\left(x_1\right), \Phi\left(x_2\right), \ldots, \Phi\left(x_n\right)\right]^T \in R^f. \tag{6}$$

Then, for classification, we should transform samples set $\Phi(X)$ to a class label matrix. But the class label matrix Y in CLSR is a strict binary label matrix which has less freedom to fit the samples. It is expected that the original strict binary constraints in Y can be relaxed into the soft constraint so that it has more freedom to fit the samples and simultaneously produce a classifier with well generalization. To this end, the slack variable matrix Y_n which is different from Y_d in DLSR is used to substitute for the original class label matrix Y. The four samples in Section 2.1 are also taken as an example here and then the slack variable class label matrix Y_n is defined as follows:

$$Y_n = \begin{bmatrix} 1 & \varepsilon_{12} & \varepsilon_{13} \\ 1 & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & 1 \\ \varepsilon_{41} & 1 & \varepsilon_{43} \end{bmatrix}, \quad \text{s.t.} \quad \varepsilon_{ij} \geq 0. \tag{7}$$

It can be seen that Y_n can help to properly reduce the class margins of CLSR to generalize well. Formally, let M be a ε dragging matrix and defined as

$$M = \begin{bmatrix} \varepsilon_{11} & \cdots & \varepsilon_{1c} \\ \vdots & m_{ij} & \vdots \\ \varepsilon_{n1} & \cdots & \varepsilon_{nc} \end{bmatrix} \quad \left(i = 1, \ldots, n;\ j = 1, \ldots, c;\ \varepsilon_{ij} \geq 0\right). \tag{8}$$

Meanwhile, let B_n be the dragging coefficient matrix and defined as

$$B_{nij} = \begin{cases} 0 & \text{if } Y_{ij} = 1 \\ 1 & \text{if } Y_{ij} = 0; \end{cases} \tag{9}$$

then $Y_n = Y + B_n \odot M$, where $\odot$ is a Hadamard product operator of matrices. Relaxing Y into Y_n has an idea opposite to that of the ε dragging technique in DLSR; therefore we call this relaxation the negative ε dragging.

By virtue of the kernel feature space R^f, our method tries to construct a bridge between $\Phi(X)$ and Y_n. In particular, our goal is to learn a linear function W that makes $\Phi(X)W = Y_n$ be approximately satisfied. Thus our method has the following objective function:

$$\begin{aligned}(W^*, M^*) = \arg\min_{W,M} \quad & \left\|\Phi(X)W - (Y + B_n \odot M)\right\|_F^2 \\ & + \lambda \|W\|_F^2 \\ \text{s.t.} \quad & M \geq 0,\end{aligned} \tag{10}$$

where W is the transform matrix and λ is a positive regularization parameter.

Since Y is relaxed into Y_n, (10) has more freedom than (1) to fit the samples. Based on the knowledge of Linear Algebra, we know that

$$\begin{aligned}&\left\|\Phi(X)W - Y_n\right\|_F^2 + \lambda \|W\|_F^2 \\ &= \operatorname{trace}\left(\left(\Phi(X)W - Y_n\right)\left(\Phi(X)W - Y_n\right)^T + \lambda WW^T\right).\end{aligned} \tag{11}$$

It is easy to prove that objective function (10) is convex. Thus it has a unique solution. An iterative updating algorithm is devised to solve it. The first step of the algorithm is to solve W by fixing M.

Theorem 1. *Given M, the optimal W in (10) can be calculated as*

$$W^* = \left(\Phi(X)^T \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T Y_n. \tag{12}$$

Proof. According to matrix theory, the optimal W can be obtained by making the derivation of (10) with respect to W and set it to zero. That is,

$$\begin{aligned}&\frac{\partial\left(\left\|\Phi(X)W - Y_n\right\|_F^2 + \lambda \|W\|_F^2\right)}{\partial W} \\ &= \Phi(X)^T \Phi(X) W - \Phi(X)^T Y_n + \lambda W = 0 \Longrightarrow \\ &W^* = \left(\Phi(X)^T \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T Y_n.\end{aligned} \tag{13}$$

The second step of our algorithm is to solve M by fixing W. Then (10) can be rewritten as $\arg\min_M \|\Phi(X)W - (Y + B_n \odot M)\|_F^2$. M can be obtained by solving the following optimization problem:

$$\begin{aligned}\min_M \quad & \left\|G - B_n \odot M\right\|_F^2 \\ \text{s.t.} \quad & M \geq 0,\end{aligned} \tag{14}$$

where

$$G = \Phi(X)W^* - Y. \tag{15}$$

Considering the ith row and jth column element M_{ij} of M, we have

$$\begin{aligned}\min_{M_{ij}} \quad & \left(G_{ij} - B_{nij} M_{ij}\right)^2 \\ \text{s.t.} \quad & M_{ij} \geq 0.\end{aligned} \tag{16}$$

According to [25], the formula to calculate M_{ij} is

$$M_{ij} = \max\left(B_{nij} G_{ij}, 0\right). \tag{17}$$

Therefore, the optimal solution of M is

$$M = \max\left(B_n \odot G, 0\right). \tag{18}$$

In a word, the first step of the algorithm is to solve W by fixing M, and the second step of the algorithm is to solve M by fixing W. In other words, (12) should be calculated in the first step, and (15) and (18) should be calculated in the second step. These two steps should be repeatedly calculated till the termination condition is satisfied. □

3.2. Integrating the Kernel Trick into the Optimization Model. As mentioned above, we should repeatedly calculate (12) and (18). However, for (12) and (18), $\Phi(X)$ exists in kernel feature space R^f. Fortunately, we do not need to know what $\Phi(X)$ is and just adopt the kernel function (3). How to use the kernel function to eliminate denotation $\Phi(X)$ is presented as follows.

Let

$$P = \left(\Phi(X)^T \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T. \tag{19}$$

By using the following formula [42] on matrix manipulations:

$$\left(A^{-1} + B^T R^{-1} B\right)^{-1} B^T R^{-1} = AB^T \left(BAB^T + R\right)^{-1}, \tag{20}$$

we use λI, $\Phi(X)$, and I instead of A^{-1}, B, and R, respectively, having

$$\begin{aligned}&\left(\lambda I + \Phi(X)^T \Phi(X)\right)^{-1} \Phi(X)^T \\ &= \Phi(X)^T \left(\Phi(X)\Phi(X)^T + \lambda I\right)^{-1}.\end{aligned} \tag{21}$$

Then, we substitute it into (12); therefore

$$W^* = \Phi(X)^T \left(\Phi(X)\Phi(X)^T + \lambda I\right)^{-1} Y_n \Longrightarrow \tag{22}$$

$$W^* = \Phi(X)^T (K + \lambda I)^{-1} Y_n = PY_n, \tag{23}$$

where $K_{i,j} = \Phi(x_i)^T \Phi(x_j) = k(x_i, x_j)$, $(i, j = 1, 2, \ldots, n)$.

Actually, $\Phi(X)^T (K + \lambda I)^{-1}$ in (23) is changeless because it only depends on X and the utilized kernel function, while Y_n is changeable during the iteration; hence, for avoiding to directly calculate $\Phi(X)$, in the first step we only need to calculate

$$Y_n = Y + B_n \odot M. \tag{24}$$

Input: Training samples matrix X; Label matrix Y; dragging coefficient matrix B_n; test sample t; parameter λ;

Output: the slack variable class label matrix Y_n; predicted class k for test sample t;

Initialization: $M = 0_{n\times c}$;

Calculate $H = K(K + \lambda I)^{-1}$;

Set threshold h; Set $itr = 1$.

Repeat

(1) Given M, calculate $Y_n = Y + B_n \odot M$.

(2) Utilize $G = HY_n - Y$, then calculate $M = \max(B \odot G, 0)$.

Until the absolute value of the difference between objective functions of two consecutive loops is smaller than threshold h.

(3) For test sample t, calculate $t_y = t^T W^*$.

(4) If $k = \arg_j \max t_y^j$, then t is classified into the kth class. t_y^j is the jth entry of t_y.

Output: the transformation matrix W^*, k.

Algorithm 1

The second step of algorithm is to solve M by calculating (15) and (18). By substituting (23) into (15), we have

$$G = K\left(K + \lambda I\right)^{-1} Y_n - Y. \tag{25}$$

Hence, in the second step we need to calculate (25) and (18).

Then the predicted label for a test sample x is

$$Y\left(x\right) = \Phi\left(x\right)^T W. \tag{26}$$

Intuitively, W should be calculated by iteration and then it is utilized to calculate the predicted label $Y(x)$ for test sample x. However, by substituting (23) into (26), we have

$$\begin{aligned} Y\left(x\right) &= \Phi\left(x\right)^T \Phi\left(X\right)^T \left(K + \lambda I\right)^{-1} Y_n \\ &= \kappa\left(x\right)\left(K + \lambda I\right)^{-1} Y_n, \end{aligned} \tag{27}$$

where $\kappa(x) = [k(x, x_1), k(x, x_2), \ldots, k(x, x_n)]$.

Because $\Phi(X)^T(K + \lambda I)^{-1}$ depends on X and the utilized kernel function, we only need to calculate Y_n out by the iteration, and after the iteration is performed, the predicted label for a test sample x can be obtained by (27). As presented above, directly calculating $\Phi(X)$ can be avoided by utilizing the kernel function.

In summary, we do not need to know what Φ is and just adopt the kernel function during the iteration. The complete algorithm is summarized in Algorithm 1.

4. Analysis of Our Method

In our method, the negative ε dragging technique and kernel trick are simultaneously integrated into the LR model to obtain more robust classification result for noised and deformable data. We analyze our method from two aspects.

Firstly, we present the class margins of our method, DLSR, and CLSR for classification. For simplicity of description, the four samples in Section 2.1 are also taken as an example here. For our method, it is clear that

$$Y_n = \begin{bmatrix} 1 & \varepsilon_{12} & \varepsilon_{13} \\ 1 & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & 1 \\ \varepsilon_{41} & 1 & \varepsilon_{43} \end{bmatrix}. \tag{28}$$

For DLSR,

$$Y_d = \begin{bmatrix} 1+\varepsilon_{11} & -\varepsilon_{12} & -\varepsilon_{13} \\ 1+\varepsilon_{21} & -\varepsilon_{22} & -\varepsilon_{23} \\ -\varepsilon_{31} & -\varepsilon_{32} & 1+\varepsilon_{33} \\ -\varepsilon_{41} & 1+\varepsilon_{42} & -\varepsilon_{43} \end{bmatrix}. \tag{29}$$

Suppose that Y_n and Y_d have the same ε components. For the first and third samples (they, respectively, belong to the first and third classes), the distance between their class labels can be denoted by

$$d_n = \sqrt{\left(1 - \varepsilon_{31}\right)^2 + \left(\varepsilon_{12} - \varepsilon_{32}\right)^2 + \left(\varepsilon_{13} - 1\right)^2}. \tag{30}$$

For DLSR, the distance between the class labels of the first and third samples can be denoted by

$$\begin{aligned} &d_d \\ &= \sqrt{\left(1 + \varepsilon_{11} + \varepsilon_{31}\right)^2 + \left(\varepsilon_{12} - \varepsilon_{32}\right)^2 + \left(\varepsilon_{13} + \varepsilon_{33} + 1\right)^2}. \end{aligned} \tag{31}$$

For CLSR, the distance between their class labels can be denoted by $d = \sqrt{2}$.

We see that if Y_n and Y_d have same ε components, DLSR has the largest class margin whereas our method usually has the smallest class margin. In other words, we usually have $d_d \geq d \geq d_n$. Actually, because $\varepsilon_{ij} \geq 0$, it is absolutely certain

that $d_d \geq d$. As for $d \geq d_n$, it can be demonstrated below. First, $d_n = \sqrt{1 - 2\varepsilon_{31} + \varepsilon_{31}^2 + \varepsilon_{12}^2 - 2\varepsilon_{12}\varepsilon_{32} + \varepsilon_{32}^2 + 1 - 2\varepsilon_{13} + \varepsilon_{13}^2}$. Because $\varepsilon_{ij} \ll 1$ is usually satisfied, we can ignore the second-order terms and have $d_n \approx \sqrt{1 - 2\varepsilon_{31} + 1 - 2\varepsilon_{13}} \leq \sqrt{2}$. As a result, in the scenario of noised and deformable data, our method can effectively decrease the probability that the classifier learned from training samples too fits training samples and cannot be well applicable for test samples. In other words, our method can make the obtained classifier generalize well and is very suitable for the classification of noised and deformable data.

Secondly, we present effects of the kernel trick integrated into our method. In some real-world applications, samples from different classes are mixed up and are not linearly separable, because the difference between training samples from the same class may be much more than the difference between training samples from different classes. For instance, in the face recognition problem, the face images from the same person may be more different than the face images from distinct persons owing to variable expressions, poses, and illuminations. This is known as the problem of uncertain data [42, 43]. Under this situation, both CLSR and DLSR could not attain a good classification performance. The kernel approach can change the distribution of samples by the nonlinear mapping. If an appropriate kernel function is utilized, the kernel approach can make linearly nonseparable samples become linearly separable. The term linearly separable means that samples of different classes have good separability. Exactly, it is referred to as a linear boundary such as a line or plane that can separate samples from different classes without errors.

Here, kernel mapping is integrated into our approach so that, in the high dimensional kernel feature space, it is easy to learn a mapping that can well convert training samples into their class labels. Namely, the linear transformation matrix obtained in the high dimensional feature space can more appropriately map training samples into their class labels. Therefore, our kernel based approach can perform classification well.

If the two class samples are not linearly separable, CLSR and DLSR could not attain a good classification performance. KNDLR firstly makes a nonlinear mapping of the data to enhance the linear separability of samples; hence KNDLR is able to obtain higher classification accuracy than CLSR and DLSR. Moreover, our KNDLR just utilizes the kernel function to calculate transform matrix W and class label for test samples instead of directly calculating $\Phi(X)$.

In addition, the overall complexity of KNDLR is low, although it is solved iteratively. In each iteration, the main computation cost is in (25), where we need to calculate the matrix inverse $(K + \lambda I)^{-1}$. Since $K(K + \lambda I)^{-1}$ is only dependent on X and the utilized kernel function, it can be precalculated before the loop is carried out. Thus the speed of calculating G in (25) is very fast. Moreover, it is obvious that K is an $n \times n$ matrix (n is the number of training samples), while $(X^T X + \lambda I)^{-1}$ calculated in CLSR or DLSR is an $m \times m$ matrix (m is the number of features). Thus, when the number of samples is much less than the dimension of features, the size of K is small. Hence it is easy to calculate the matrix inverse $(K + \lambda I)^{-1}$. If the features are very high dimensional, calculation of the matrix inverse $(X^T X + \lambda I)^{-1}$ will be quite time-consuming and memory-consuming. In particular, although our KNDLR approach is similar to CLSR and DLSR in some aspects, it is much more efficient than them when classifying high dimensional data. However, when the number of samples is not much smaller than that of features and dimension of features is high, size of K is large. Hence calculating the matrix inverse $(K + \lambda I)^{-1}$ is complex as solving the inverse matrix $(X^T X + \lambda I)^{-1}$ and the efficient of our KNDLR is almost the same as that of CLSR and DLSR.

5. Experiments

In our experiments, KNDLR was compared with CLSR, DLSR, NDLR (the KNDLR without kernel trick), kernel support vector machine (K-SVM) in [31], k nearest neighbor method (KNN), the nonnegative least squares method (NNLS) proposed in [3], sparse representation based classification (SRC (l1_ls)), and linear regression based classification (LRC). We use five face image databases and a handwriting digit dataset, namely, Georgia Tech (GT), FERET, LFW, AR, YaleB, and MNIST dataset. The subsets of the last two datasets, which are available at "http://www.cad.zju.edu.cn/home/dengcai/Data/data.html," were used to perform our experiments. All methods were directly performed on image, with no extracting feature from image in advance. Our method, CLSR, DLSR, and NDLR all have a parameter λ. The parameter was set to 0.0001, 0.0005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, and 0.5, respectively. The best accuracy of each method is given for comparison. Threshold h is set to 0.0001. For KNDLR, we used polynomial kernel $k(x, x') = (x^T x' + c)^d$ on LFW, AR, YaleB, and MNIST and Gaussian radial basis function (RBF) kernel $k(x, x') = \exp(-\gamma \|x - x'\|_2^2)$ on GT and FERET, respectively. The parameters of polynomial kernels c and d were set to 1 and 2, respectively. The parameter of RBF kernel γ was set to the median value of $1/(\|x_i - \overline{x}\|^2)$, $i = 1, \ldots, n$, where $\overline{x}$ is the mean of all training samples. For K-SVM, package libsvm-mat-3.0-1 is used. The libsvm_options of the function "svmtrain" were set to ["$-s$ 0 $-t$ 2 $-g$ 1.0e $-$ 1"], where "$-t$ 2" indicates Gaussian radial basis function (RBF) kernel. The value of hyperparameter C was selected from the candidate set $\{0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$ by cross-validation approach. For KNN, k was set to 1, and Euclidean distance metric was used to find the nearest neighbor.

5.1. Experiment on the GT Database. The Georgia Tech (GT) face database contains 750 images from 50 subjects. For each subject 15 face images are available. The pictures show frontal and/or tilted faces with different facial expressions, lighting conditions, and scales. Figure 1 presents some face images from the GT face database. In our experiments, all images in the database were manually cropped and resized to 30×40. After the image cropping, most of the complex background has been excluded. They are further converted to gray level images for both training and testing purposes.

Figure 1: Some face images from the GT face database.

Table 1: Accuracies (%) of different methods on the GT database.

Number of training samples per class	5	6	7	8	9	10
Our method	**70.80**	**73.53**	**76.05**	**80.26**	**81.33**	**82.36**
DLSR	50.48	50.93	52.80	53.77	53.77	54.56
CLSR	63.58	65.33	67.30	70.57	71.00	71.40
NNLS	68.90	71.02	73.88	76.83	78.27	79.68
K-SVM	70.42	72.20	74.93	79.46	80.17	81.92
KNN	59.30	61.76	63.85	67.49	69.23	69.40
SRC	56.18	58.44	60.13	63.09	63.57	64.36
LRC	68.90	71.22	73.78	77.71	79.20	80.68
NDLR	65.66	66.76	69.38	73.60	73.57	74.56

Table 2: Accuracies (%) of different methods on the FERET database.

Number of training samples per class	1	2	3	4	5
Our method	**35.52**	**55.84**	**66.29**	**79.03**	**78.18**
DLSR	19.45	31.18	36.99	45.95	44.15
CLSR	28.03	46.38	55.18	65.03	65.07
NNLS	34.26	53.48	64.38	75.84	76.33
K-SVM	32.01	54.73	64.43	75.89	73.96
KNN	32.01	44.87	54.44	68.14	65.70
SRC	23.54	41.48	53.12	68.51	70.46
LRC	32.01	55.34	65.89	77.12	74.76
NDLR	33.77	54.28	63.50	72.64	72.60

In our experiments, we randomly took n (= 5, 6, 7, 8, 9, 10) face images of each subject as original training samples, respectively, and treated the remaining face images as testing samples. For each given n, we take the average value of classification rates calculated from 10 random splits as final classification rate. The experimental results are presented in Table 1. From this table, we can conclude that the proposed method obtains the best classification accuracy.

5.2. Experiment on the FERET Face Dataset. A subset of the FERET face dataset was used in the experiment. This subset includes 1442 face images from 206 subjects and each subject has seven different face images. This subset was composed of the images in the original FERET face dataset whose names are marked with two-character strings: "ba," "bj," "bk," "be," "bf," "bd," and "bg". Figure 2 shows some image examples. We resized all face images to 40 by 40 matrices.

In our experiments, n (= 1, 2, 3, 4, 5) samples of each subject were randomly taken as training samples and the remaining samples were treated as test samples. For each given n, we take the average value of classification rates calculated from 10 random splits as final classification rate. Experimental results of classification accuracies are shown in Table 2. Table 2 demonstrates that our method performs better than the other methods.

5.3. Experiment on the LFW Face Dataset. The LFW dataset is a face image dataset for unconstrained face recognition. Images in this dataset vary much in clothing, pose, and background more than the other face datasets. There are more than 13000 faces images collected from the web. Every face image is manually labeled. We use only a subset composed of 1251 images from 86 subjects to conduct experiments. Figure 3 shows some example face images. Each image is cropped and resized to 32×32 image.

A random subset with n (= 6, 7, 8, 9, 10) images per individual was taken with labels to form the training set, and the rest of the database was considered to be the testing set. For each given n, there are 10 random splits. The average value of classification rates calculated from 10 random splits was taken as final classification rate. The classification accuracies were shown in Table 3. It is clear that our method performs better than the rest of methods.

5.4. Experiment on the AR Face Dataset. The AR dataset contains over 4000 face images of 126 subjects, including frontal views of faces with different facial expressions, lighting conditions, and occlusion. We use only a subset composed of 3120 images from 120 subjects and each subject has 26 different face images. Figure 4 shows some example face images. Each image is cropped and resized to 40×50 image.

Figure 2: Some face images from the FERET face database.

Figure 3: Some face images from the LFW face database.

A random subset with n (= 1, 2, 3, 4, 5) images per individual was taken with labels to form the training set, and the rest of the database was considered to be the testing set. For each given n, there are 10 random splits. The average classification rate calculated from 10 random splits was taken as final classification rate. The classification accuracies were shown in Table 4. It is clear that our method performs better than the rest of methods.

5.5. Experiment on the YaleB Face Dataset. For this database, we simply use the cropped images and resize them to 32×32 pixels to conduct experiments. Figure 5 shows some example face images.

A random subset with n (= 5, 6, 7, 8) images per individual was taken with labels to form the training set, and the rest of the database was considered to be the testing set. For each given n, there are 10 random splits. The average

Figure 4: Some face images from the AR face database.

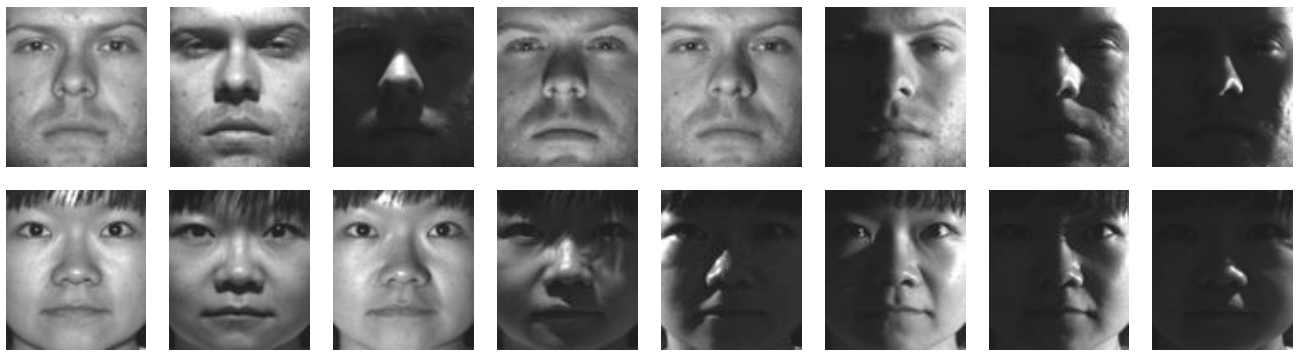


Figure 5: Some face images from the YaleB face database.

Table 3: Accuracies (%) of different methods on the LFW database.

Number of training samples per class	6	7	8	9	10
Our method	**35.95**	**38.00**	**39.13**	**40.27**	**41.10**
DLSR	14.46	14.92	14.58	15.43	14.68
CLSR	34.45	35.93	35.15	36.65	35.27
NNLS	33.86	36.58	36.04	36.69	37.31
K-SVM	32.88	34.76	35.70	36.52	38.59
KNN	20.34	21.28	21.12	21.26	21.53
SRC	34.15	36.70	38.06	39.31	39.00
LRC	32.12	35.16	34.78	36.50	40.05
NDLR	35.22	37.49	38.40	39.29	40.69

Table 4: Accuracies (%) of different methods on the AR database.

Number of training samples per class	1	2	3	4	5
Our method	**64.69**	**75.87**	**83.74**	**90.46**	**95.12**
DLSR	56.83	70.01	78.34	85.89	91.37
CLSR	59.73	74.00	82.55	89.05	93.89
NNLS	59.93	67.08	78.03	85.09	92.90
K-SVM	58.95	66.39	74.56	83.64	91.69
KNN	58.95	61.45	68.88	75.37	84.03
SRC	61.81	74.36	82.72	89.95	94.35
LRC	58.95	65.42	73.94	82.06	91.15
NDLR	64.00	74.87	83.38	90.02	94.63

classification rate calculated from 10 random splits was taken as final classification rate. The classification accuracies were shown in Table 5. It is clear that our method performs better than the rest of methods, except for SRC. However, SRC is time-consuming, which is shown in Section 5.7.

5.6. Experiment on the MNIST Dataset. The MNIST database of handwritten digits from Yann LeCun's page has a training set of 60,000 examples and a test set of 10,000 examples. We use only a subset composed of the first 2k training images and first 2k test images to conduct experiments. The size of each image is 28×28 pixels, with 256 gray levels per pixel. Thus, each image is represented by a 784-dimensional vector. Figure 6 shows some example images. Experimental results of classification accuracies are shown in Table 6. From this table, we can conclude that the proposed method obtains the best classification accuracy.

5.7. Computing Time. Aforementioned experiments were performed on an Intel machine (Core (TM) i5-6600 CPU, 3.30 GHz, 8 GB RAM, with 64-bit Win 10 Chinese operating system). All methods, except for the SVM methods, were implemented by software MATLAB 2010a. The libSVM3.0 toolbox in the language C was utilized for performing SVM.

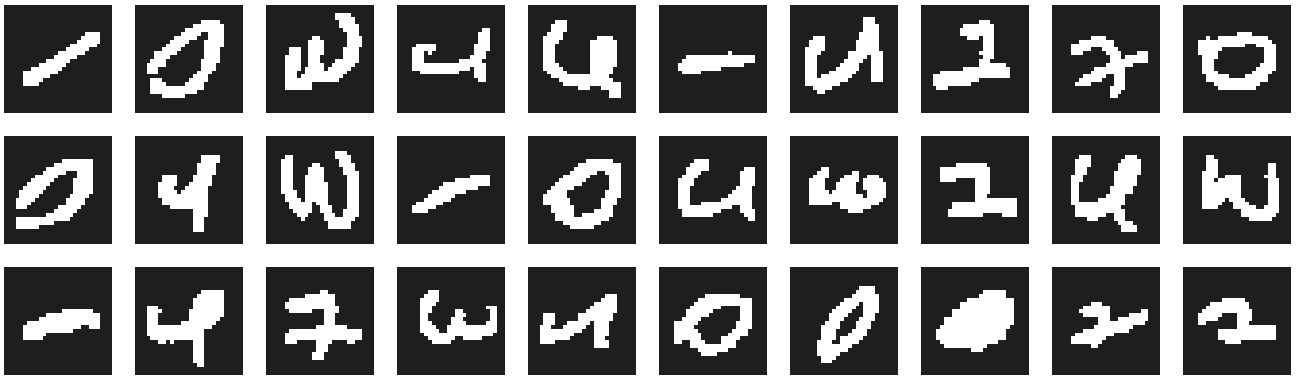


Figure 6: Some images from the MNIST database.

Table 5: Accuracies (%) of different methods on the YaleB database.

Number of training samples per class	5	6	7	8
Our method	80.48	81.67	82.28	84.55
DLSR	76.05	78.48	79.33	77.54
CLSR	76.83	78.87	80.11	77.85
NNLS	68.98	73.39	73.38	75.14
K-SVM	74.45	76.53	75.58	76.93
KNN	57.56	60.42	60.14	60.92
SRC	**82.53**	**84.54**	**85.22**	**85.29**
LRC	79.14	81.01	80.87	84.36
NDLR	77.02	79.40	80.25	78.73

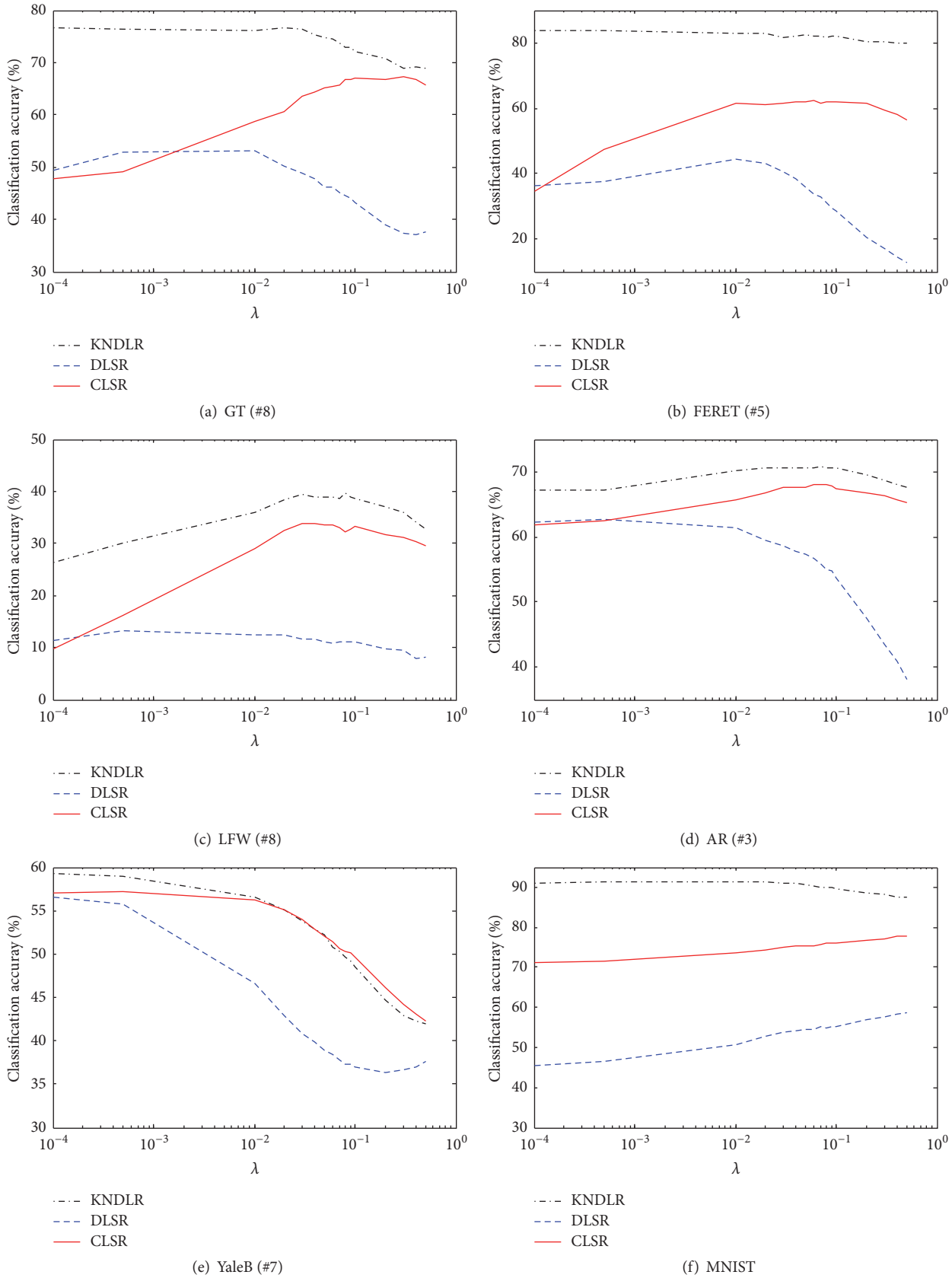
Besides classification accuracies, because the computing time is significantly different for each method, we select the experiment on GT and AR to show the computing time of each method. The GT database only contains a small number of samples, while the AR database contains a relatively large number of samples, which represent two different cases. Here, the computing time of each method is the sum of time spent on learning from samples and time spent on classification of new samples when training samples and test samples have been given. We use MATLAB instruction tic and toc to get the time. Table 7 shows the computing time of the methods on GT and Table 8 shows that on AR.

First, it can be clearly seen that our KNDLR approaches are very fast as DLSR, CLSR, NDLR, SVM, and KNN on GT having a small number of samples and AR having relatively large number of samples. Second, KNDLR is much faster than NNLS and SRC, especially on AR. Third, it is shown that the computing time of KNDLR on AR is only a little longer than that on GT, while the computing time of some methods on AR, such as NNLS and SRC, is far longer than that on GT. In particular, SRC becomes very time-consuming when the number of samples is large. One of the reasons for the efficiency of our methods is that, in our approach, the procedure of learning is executed only once, then the results are saved for classifying all new samples. SRC needs to learn a linear combination of all training samples for every new sample; thus when the number of samples is large, SRC becomes extremely time-consuming. This demonstrates that our KNDLR is efficient.

5.8. Parameter λ and Convergence. In order to further illustrate the properties of KNDLR, the classification accuracies corresponding to different values of λ and convergence are shown in Figures 7 and 8, respectively, where (#n) represents that the first n samples were utilized for training and the remaining for testing. KNDLR, DLSR, and CLSR are similar to each other to some degree. All of them apply the least squares regression and have a regularization parameter λ. In Figure 7, it is shown that KNDLR is relatively more robust to λ than DLSR and CLSR. Especially, for GT, FERET, AR, and MNIST, the classification accuracies obtained by utilizing KNDLR vary in a small range. It is also observed that a relative large value of λ cannot bring more preferable classification accuracy and λ could be limited to $[10^{-4}, 0.5]$. In real application, the cross-validation method is utilized to determine the optimal value of λ from this range. More importantly, in Figure 8, it is shown that KNDLR converges very fast on six datasets, especially on FERET database.

6. Conclusions

This paper proposed a kernel negative ε dragging linear regression method for pattern classification, which simultaneously integrated the negative ε dragging technique and the kernel method into linear regression for robust pattern classification under the condition that the consistency and compatibility between the test samples and training samples are poor. The negative ε dragging technique learns a classifier with a proper margin from noised and deformable data. Meanwhile, the kernel approach can make linearly nonseparable samples become linearly separable. Based on effect of the negative ε dragging technique and kernel collaboration, our method can better perform classification for noised and deformable data. Comprehensive experiments on six different datasets demonstrate that proposed KNDLR outperforms existing LR method for classification and some other commonly used methods such as SVM, NNLS, SRC, and LRC, and our KNDLR is efficient.



(a) GT (#8)

(b) FERET (#5)

(c) LFW (#8)

(d) AR (#3)

(e) YaleB (#7)

(f) MNIST

Figure 7: The classification accuracies (%) versus value λ on the six datasets.

Table 6: Accuracies (%) of different methods on the MNIST database.

Alg.	Our method	DLSR	CLSR	NNLS	K-SVM	KNN	SRC	LRC	NDLR
Accur.	**91.5**	58.60	77.70	90.25	86.25	88.90	84.50	82.70	79.00

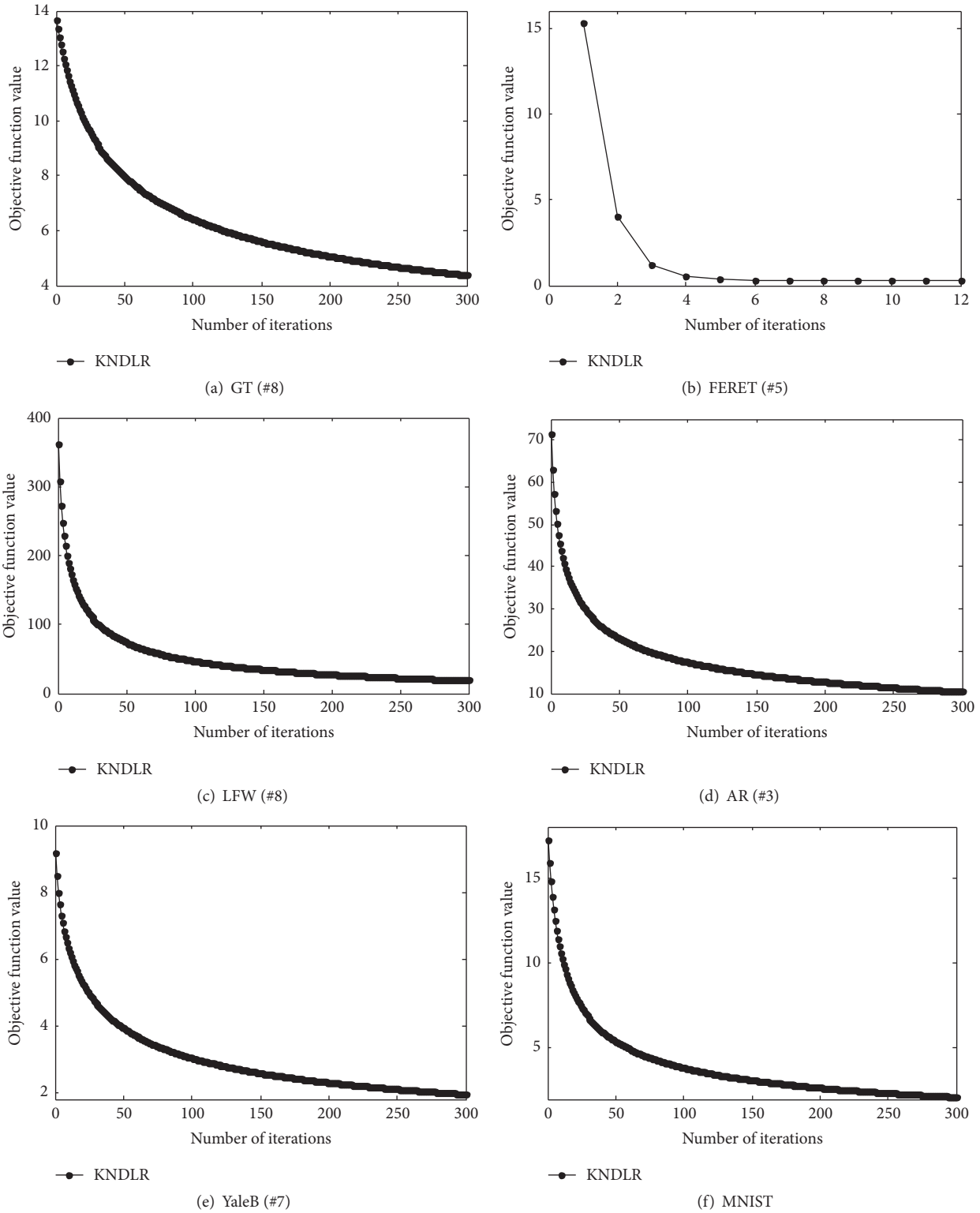


Figure 8: Convergence curves of KNDLR on the six datasets.

Table 7: Computing time (s) of different methods on the GT database.

Number of training samples per class	5	6	7	8	9	10
Our method	1.94	2.41	2.72	3.24	3.69	4.21
DLSR	3.67	4.01	4.06	4.24	4.46	4.65
CLSR	1.80	2.01	1.80	1.85	1.84	1.88
NNLS	7.56	10.71	13.54	16.46	18.99	22.08
SVM	**0.41**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**
KNN	0.75	0.80	0.85	0.85	0.88	0.80
SRC	63.94	77.34	94.32	103.43	115.84	124.28
LRC	3.63	3.44	3.58	3.44	3.52	3.35
NDLR	3.72	3.90	4.06	4.25	4.39	4.60

Table 8: Computing time (s) of different methods on the AR database.

Number of training samples per class	1	2	3	4	5
Our method	2.88	5.76	8.62	12.10	14.74
DLSR	18.28	19.69	21.17	22.76	24.20
CLSR	10.30	10.46	10.62	10.73	10.84
NNLS	14.70	50.14	120.81	251.74	447.02
SVM	**1.41**	**2.55**	**3.67**	**4.88**	**6.04**
KNN	3.16	6.24	10.18	23.05	23.99
SRC	231.48	803.65	1564.73	2375.6	3616.28
LRC	21.83	32.54	41.38	48.06	62.64
NDLR	18.25	19.81	21.16	22.73	24.19

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (nos. 61672333, 61402274, and 41471280), the Program of Key Science and Technology Innovation Team in Shaanxi Province (no. 2014KTC-18), the Key Science and Technology Program of Shaanxi Province, China (no. 2016GY-081), the Fundamental Research Funds for the Central Universities (no. 2017CSY024), the Industry University Cooperative Education Project of Higher Education Department of the Ministry of Education (no. 201701023062), and the Interdisciplinary Incubation Project of Learning Science of Shaanxi Normal University.

References

[1] T. Strutz, *Data fitting and uncertainty: a practical introduction to weighted least squares and beyond*, Vieweg, Wiesbaden, Germany, 2010.

[2] S. Wold, A. Ruhe, H. Wold, and W. J. Dunn III, "The collinearity problem in linear regression. The Partial Least Squares (PLS) approach to generalized inverses," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 735–743, 1984.

[3] Y. Li and A. Ngom, "Nonnegative least-squares methods for the classification of high-dimensional biological data," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 2, pp. 447–456, 2013.

[4] Y. Xu, X. Fang, Q. Zhu, Y. Chen, J. You, and H. Liu, "Modified minimum squared error algorithm for robust classification and face recognition experiments," *Neurocomputing*, vol. 135, pp. 253–261, 2014.

[5] C. Zhou, W. Lu, P. Zhang, J. Wu, Y. Hu, and L. Guo, "On the minimum differentially resolving set problem for diffusion source inference in networks," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 79–85, 2016.

[6] C. Xu, D. Tao, Y. Li, and C. Xu, "Large-margin multi-view Gaussian process," *Multimedia Systems*, vol. 21, no. 2, pp. 147–157, 2014.

[7] U. Brefeld, T. Gärtner, T. Scheffer, and S. Wrobel, "Efficient co-regularised least squares regression," in *Proceedings of the Twenty-Third International Conference (ICML 2006)*, pp. 137–144, Carnegie Mellon University, Pittsburgh, USA, June 2006.

[8] B. Du, M. Zhang, L. Zhang, R. Hu, and D. Tao, "PLTD: Patch-Based Low-Rank Tensor Decomposition for Hyperspectral Images," *IEEE Transactions on Multimedia*, vol. 19, no. 1, pp. 67–79, 2017.

[9] T. Liu, M. Gong, and D. Tao, "Large-Cone Nonnegative Matrix Factorization," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[10] C. Liu, C. Zhou, J. Wu, H. Xie, Y. Hu, and L. Guo, "CPMF: A collective pairwise matrix factorization model for upcoming event recommendation," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1532–1539, Anchorage, AK, USA, May 2017.

[11] S. M. Xiang, F. P. Nie, and C. S. Zhang, "Regression reformulations of LLE and LTSA with locally linear transformation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 41, no. 5, pp. 1250–1262, 2011.

[12] F. P. Nie, H. Wang, H. Huang, and C. Ding, "Adaptive loss minimization for semi-supervised elastic embedding," in *Proceedings of the in proceedings of the twenty-third international joint conference on artificial intelligence*, 2013.

[13] X. Z. Fang, Y. Xu, X. Li, Z. Lai, and W. K. Wong, "Learning a nonnegative sparse graph for linear regression," *IEEE Transactions on Image Processing*, vol. 24, no. 9, pp. 2760–2771, 2015.

[14] S. Liu, L. Li, Y. Peng, G. Qiu, and T. Lei, "Improved sparse representation method for image classification," *IET Computer Vision*, vol. 11, no. 4, pp. 319–330, 2017.

[15] T. Strutz, *Data Fitting and Uncertainty*, Vieweg, Wiesbaden, Germany, 2010.

[16] B. Du and L. Zhang, "A discriminative metric learning based anomaly detection method," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 11, pp. 6844–6857, 2014.

[17] T. Liu and D. Tao, "On the performance of Manhattan nonnegative matrix factorization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 9, pp. 1851–1863, 2016.

[18] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, "Stacked Convolutional Denoising Auto-Encoders for Feature Representation," *IEEE Transactions on Cybernetics*, vol. 47, no. 4, pp. 1017–1027, 2017.

[19] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 210–227, 2009.

[20] Y. Xu, D. Zhang, J. Yang, and J. Y. Yang, "A two-phase test sample sparse representation method for use with face recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 9, pp. 1255–1262, 2011.

[21] Y. Xu, Q. Zhu, Z. Fan, D. Zhang, J. Mi, and Z. Lai, "Using the idea of the sparse representation to perform coarse-to-fine face recognition," *Information Sciences*, vol. 238, pp. 138–148, 2013.

[22] S. Liu, Y. Peng, X. Ben, W. Yang, and G. Qiu, "A novel label learning algorithm for face recognition," *Signal Processing*, vol. 124, pp. 141–146, 2016.

[23] I. Naseem, R. Togneri, and M. Bennamoun, "Linear regression for face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 11, pp. 2106–2112, 2010.

[24] Z. Zhang, Y. Xu, J. Yang, X. Li, and D. Zhang, "A survey of sparse representation: algorithms and applications," *IEEE Access*, vol. 3, pp. 490–530, 2015.

[25] S. Xiang, F. Nie, G. Meng, C. Pan, and C. Zhang, "Discriminative least squares regression for multiclass classification and feature selection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 11, pp. 1738–1754, 2012.

[26] Q. Li, B. Xie, J. You, W. Bian, and D. Tao, "Correlated logistic model with elastic net regularized for multilabel image classification," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3801–3813, 2016.

[27] C. Gong, D. Tao, W. Liu, L. Liu, and J. Yang, "Label propagation via teaching-to-learn and learning-to-teach," *IEEE Transactions on Neural Networks Learning Systems*, vol. 28, no. 6, pp. 1452–1465, 2017.

[28] C. Gong, T. Liu, D. Tao, K. Fu, E. Tu, and J. Yang, "Deformed Graph Laplacian for Semisupervised Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2261–2274, 2015.

[29] C. Xu, D. Tao, and C. Xu, "Large-Margin Multi-Label Causal Feature Learning," in *Proceedings of the 29th Conference on Artificial Intelligence (AAAI '15)*, pp. 1924–1930, 2015.

[30] T. Liu, D. Tao, M. Song, and S. J. Maybank, "Algorithm-dependent generalization bounds for multi-task learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 2, article A4, pp. 227–241, 2017.

[31] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[32] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[33] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[34] Y. Xu, D. Zhang, F. Song, J.-Y. Yang, Z. Jing, and M. Li, "A method for speeding up feature extraction based on KPCA," *Neurocomputing*, vol. 70, no. 4–6, pp. 1056–1061, 2007.

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

[36] K. Yu, L. Ji, and X. Zhang, "Kernel nearest-neighbor algorithm," *Neural Processing Letters*, vol. 15, no. 2, pp. 147–156, 2002.

[37] L. Zhang, W.-D. Zhou, P.-C. Chang et al., "Kernel sparse representation-based classifier," *IEEE Transactions on Signal Processing*, vol. 60, no. 4, pp. 1684–1695, 2012.

[38] Y. Xu, Z. Fan, and Q. Zhu, "Feature space-based human face image representation and recognition," *Optical Engineering*, vol. 51, no. 1, Article ID 017205, 2012.

[39] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 12, no. 2, pp. 181–201, 2001.

[40] M. Kim, "Accelerated max-margin multiple kernel learning," *Applied Intelligence*, vol. 38, no. 1, pp. 45–57, 2013.

[41] C. Xu, T. Liu, D. Tao, and C. Xu, "Local Rademacher complexity for multi-label learning," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1495–1507, 2016.

[42] K. Petersen and M. Pedersen, The Matrix Cookbook, 2006, http://matrixcookbook.com/.

[43] Y. Xu, X. Z. Fang, X. L. Li, J. Yang, J. You, and H. Liu, "Data uncertainty in face recognition," *IEEE Transactions on Cybernetics*, 2014.