Journal of University of Science and Technology Beijing
Volume 10, Number 5, October 2003, Page 73

Automation

Online LS-SVM for function estimation and classification

Jianghua Liu[1], *Jia-pin Chen*[1], *Shan Jiang*[2], *and Junshi Cheng*[1]

1) Information Storage Research Center, Shanghai Jiaotong University, Shanghai 200030, China
2) Makuhari Systems Laboratory 1-9-3 Nakase, Mihama-ku, Chiba, Chiba 261-8588, Japan
(Received 2002-11-28)

Abstract: An online algorithm for training LS-SVM (Least Square Support Vector Machines) was proposed for the application of function estimation and classification. Online LS-SVM means that LS-SVM can be trained in an incremental way, and can be pruned to get sparse approximation in a decremental way. When a SV (Support Vector) is added or removed, the online algorithm avoids computing large-scale matrix inverse. Thus the computation cost is reduced. Online algorithm is especially useful to realistic function estimation problem such as system identification. The experiments with benchmark function estimation problem and classification problem show the validity of this online algorithm.

Key Words: least-square support vector machine; online training; function estimation; classification

[*This project was financially supported by the National Natural Science Foundation of China (No. 69889050).*]

1 Introduction

Due to Vapnik and a lot of other researchers' job, statistical learning theory (including SVM) has experienced rapid development [1]. Based on standard SVM, a lot of variations of SVM have been put forward such as LS-SVM (Least Square Support Vector Machine) [2]. The difference between LS-SVM and standard SVM is that the constraint condition is inequality, not inequality. LS-SVM needs only to solve linear equations, not quadratic programming. LS-SVM has been applied to classification, function estimation and nonlinear system optimal control problems [3-5]. But the existed LS-SVM algorithm is trained offline in batch way. Offline training algorithm is not fit for the practical applications such as online system identification and control problems, where the data come in sequentially. Therefore the online training for function estimation and classification is needed urgently.

Ahmed has brought forth an incremental training algorithm for SVM classification problem [6]. The basic idea is that only the SVs (Support Vectors) are preserved, and these SVs plus the new coming data are used for training again. The main problem is that the training is not exactly incremental. It is approximately incremental and the coefficients corresponding to SVs are not updated online. To overcome this problem, Cauwenberghs (2000) [4] and Csato (2001) have put forward an exact incremental and decremental training algorithm for classification. Friess (1999) and Vijayakumar (1999) proposed sequential gradient methods [7-10], where the main drawback is that the training process converges slowly. This article proposes the incremental and decremental algorithm for LS-SVM, which makes the application of online LS-SVM to system identification and control problems possible.

2 Online LS-SVM for function estimation

2.1 Standard LS-SVM for function estimation

Firstly the standard LS-SVM for function estimation is briefly reviewed as follows [3]. Consider a training set containing N data points $\{x_k, y_k\}, k=1,...,N$, with input $x_k \in R^n$ and output $y_k \in R$, the following regression model is used.

$$y(x) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(x) + b \tag{1}$$

where $\boldsymbol{\varphi}(x)$ maps the input data into a higher dimensional feature space, $\boldsymbol{w}$ is the coefficient, b is the bias. In LS-SVM for function estimation, the object function is defined as follows:

$$\begin{cases} \min\limits_{\boldsymbol{w},e} J(\boldsymbol{w},e) = \dfrac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \dfrac{C}{2}\sum\limits_{k=1}^{N} e_k^{\,2} \\ y_k = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(x_k) + b + e_k, \quad k = 1,...,N \end{cases} \tag{2}$$

Corresponding author: Jianghua Liu, **E-mail:** jhliu99@163.com

where C is the user defined constant, which balances the model complexity and approximation accuracy. e_k is the approximation error.

The corresponding Lagrangian is given by

$$L(\boldsymbol{w},b,\boldsymbol{e};\boldsymbol{\alpha}) = J(\boldsymbol{w},\boldsymbol{e}) - \sum_{k=1}^{N}\alpha_k\{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(x_k)+b+e_k-y_k\} \tag{3}$$

where α_k is Lagrange multipliers. According to Kun-Tucker conditions, have

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{w}} = 0 \rightarrow \boldsymbol{w} = \sum_{k=1}^{N}\alpha_k\boldsymbol{\varphi}(x_k) \\ \dfrac{\partial L}{\partial b} = 0 \rightarrow \sum_{k=1}^{N}\alpha_k = 0 \\ \dfrac{\partial L}{\partial e_k} = 0 \rightarrow \alpha_k = Ce_k \\ \dfrac{\partial L}{\partial \alpha_k} = 0 \rightarrow \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(x_k)+b+e_k-y_k = 0 \end{cases} \tag{4}$$

for k=1, ..., N. Eliminate $e_k, \boldsymbol{w}$, Equation (4) can be changed into:

$$\begin{bmatrix} 0 & \vec{\boldsymbol{I}}^{\mathrm{T}} \\ \vec{\boldsymbol{I}} & \boldsymbol{\Omega}+C^{-1}I \end{bmatrix}\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{Y} \end{bmatrix} \tag{5}$$

where

$$\boldsymbol{Y} = [y_1;\ldots;y_N], \vec{\boldsymbol{I}} = [1;\ldots;1]^{\mathrm{T}}, \boldsymbol{\alpha} = [\alpha_1;\ldots;\alpha_N].$$

and Mercer condition has been applied to matrix $\boldsymbol{\Omega}$ with

$$\boldsymbol{\Omega}_{kl} = \boldsymbol{\varphi}(x_k)^{\mathrm{T}}\boldsymbol{\varphi}(x_l),\ k,l = 1,\ldots,N \tag{6}$$

According to equations (1) and (4), LS-SVM model for function estimation is obtained:

$$y(x) = \sum_{k=1}^{N}\alpha_k\Phi(x,x_k)+b \tag{7}$$

where $\boldsymbol{\alpha}, b$ are the solutions to equation (5). $\Phi(x,x_k)$ is the kernel function. RBF (Radius Base Function) kernel is chosen, *i.e.*

$$\Phi(x,x_k) = \exp\{-\frac{\|x-x_k\|_2^2}{\sigma^2}\},$$

where σ is the user-defined constant.

2.2 Online LS-SVM for function estimation

Considering that LS-SVM model based on the first N pairs of data has been built, and the new data pair (x_{N+1}, y_{N+1}) is coming in.

In equation (5), Let

$$\begin{bmatrix} 0 & \vec{\boldsymbol{I}}^{\mathrm{T}} \\ \vec{\boldsymbol{I}} & \boldsymbol{\Omega}+\boldsymbol{C}^{-1}\boldsymbol{I} \end{bmatrix} = \boldsymbol{A}_N, \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \boldsymbol{a}_N \text{ and } \begin{bmatrix} 0 \\ \boldsymbol{Y} \end{bmatrix} = \boldsymbol{Y}_N.$$

Then equation (5) changes into

$$\boldsymbol{A}_N\boldsymbol{a}_N = \boldsymbol{Y}_N \Rightarrow \boldsymbol{a}_N = \boldsymbol{A}_N^{-1}\boldsymbol{Y}_N \tag{8}$$

The subscript N means that the current model is based on the first N pairs of data. For N+1 pairs of data, have

$$\boldsymbol{a}_{N+1} = \boldsymbol{A}_{N+1}^{-1}\boldsymbol{Y}_{N+1} \tag{9}$$

where

$$\boldsymbol{A}_{N+1} = \begin{bmatrix} \boldsymbol{A}_N & \boldsymbol{b}_1 \\ \boldsymbol{b}_2 & c \end{bmatrix} \tag{10}$$

$$\boldsymbol{b}_1 = [1\ K_{1,N+1}\ K_{2,N+1}\ \ldots\ K_{N,N+1}]^{\mathrm{T}} \tag{11}$$

$$\boldsymbol{b}_2 = \boldsymbol{b}_1^{\mathrm{T}} \tag{12}$$

$$c = K_{N+1,N+1} \tag{13}$$

$$K_{i,j} = \Phi(x_i,x_j) \tag{14}$$

$$\boldsymbol{Y}_{N+1} = \begin{bmatrix} \boldsymbol{Y}_N \\ y_{N+1} \end{bmatrix} \tag{15}$$

According to matrix textbook [11], the following two lemmas exit:

Lemma 1 For matrix $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix}$, where $\boldsymbol{A}_{11}^{-1}, \boldsymbol{A}_{22}^{-1}$ exist, the following equation is true

$$\boldsymbol{A}^{-1} = \begin{bmatrix} [\boldsymbol{A}_{11}-\boldsymbol{A}_{12}\boldsymbol{A}_{22}^{-1}\boldsymbol{A}_{21}]^{-1} & \boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}[\boldsymbol{A}_{21}\boldsymbol{A}_{22}^{-1}\boldsymbol{A}_{12}-\boldsymbol{A}_{22}]^{-1} \\ [\boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}-\boldsymbol{A}_{22}]^{-1}\boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1} & [\boldsymbol{A}_{22}-\boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}]^{-1} \end{bmatrix} \tag{16}$$

Lemma 2 For matrix $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$, where $\boldsymbol{A}^{-1}, \boldsymbol{C}^{-1}$ exist, the following equation is true

$$(\boldsymbol{A}+\boldsymbol{BCD})^{-1} = \boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{C}^{-1}+\boldsymbol{DA}^{-1}\boldsymbol{B})^{-1}\boldsymbol{DA}^{-1} \tag{17}$$

From lemma 1 and lemma 2, theorem 1 can be inferred as follows:

Theorem 1 The matrix $\boldsymbol{A}_{N+1}^{-1}$ in equation (9) can be computed from $\boldsymbol{A}_N^{-1}$ without the need of computing the matrix inverse.

Proof According to equation (16), equation (10) can be changed into

$$A_{N+1}^{-1}=\begin{bmatrix}A_N & b_1\\ b_2 & c\end{bmatrix}^{-1}=\begin{bmatrix}\left[A_N-\frac{1}{c}b_1b_2\right]^{-1} & A_N^{-1}b_1\left[b_2A_N^{-1}b_1-c\right]^{-1}\\ \left[b_2A_N^{-1}b_1-c\right]^{-1}b_2A_N^{-1} & \left[c-b_2A_N^{-1}b_1\right]^{-1}\end{bmatrix}^{-1} \tag{18}$$

Applying equation (17) to the top left sub matrix in equation (18), have

$$\left[A_N-\frac{1}{c}b_1b_2\right]^{-1}=A_N^{-1}-A_N^{-1}b_1\left[-c+b_2A_N^{-1}b_1\right]^{-1}b_2A_N^{-1} \tag{19}$$

Let $\Delta=\left[c-b_2A_N^{-1}b_1\right]^{-1}$, equation (18) can changes into:

$$A_{N+1}^{-1}=\begin{bmatrix}A_N^{-1} & \vec{0}\\ \vec{0}^{\mathrm{T}} & 0\end{bmatrix}+\Delta\begin{bmatrix}A_N^{-1}b_1\\ -1\end{bmatrix}\begin{bmatrix}b_2A_N^{-1} & -1\end{bmatrix} \tag{20}$$

It is clear that A_{N+1}^{-1} can be computed from A_N^{-1} without the need of computing the matrix inverse.

End of proof.

With equation (20) A_{N+1}^{-1} is computed in an incremental way, which avoids expensive inversion operation. Therefore the corresponding coefficients and bias $a_{N+1}=[b\ \ \alpha]^{\mathrm{T}}$ can be computed according to equation (9). Then function estimation can work. Given a new input x, the corresponding function value $y(x)$ can be estimated by equation (7). In this way the incremental algorithm of LS-SVM for function estimation is got.

It is noted that the solutions of stand LS-SVM are not sparse. In realistic application, there is no enough space to store so many coefficients. And the computational cost for function estimation is very huge too. To get sparse solutions, Suykens [3] proposed a pruning method, which removes all support vectors whose coefficients α_k (in equation (7)) are below some threshold. When LS-SVM' Support Vectors are pruned one by one, this pruning method is called as decremental algorithm.

Decremental algorithm means that a SV is removed when a pair of training data is removed. Take an example, if α_k is below some predefined threshold, then the kth support vector (x_k, y_k) must removed. At the same time the coefficients corresponding to other SVs need be changed too. Similar to the case of incremental algorithm, to avoid compute the matrix inverse, A_N^{-1} must be from A_{N+1}^{-1}. Here A_N^{-1} is the matrix without the kth row and the kth column.

For decremental way, the update rule was obtained [4].

$$a_{ij}\leftarrow a_{ij}-a_{kk}^{-1}a_{ik}a_{kj} \tag{21}$$

where, $i,j=1,...,N;\ i,j\neq k$. a_{ij} stands for the item at the ith row and jth column of A_{N+1}^{-1}. k stands for the support vector to be removed. According to equation (21), A_N^{-1} can be obtained from A_{N+1}^{-1}. Then the coefficients of LS-SVM, $a_N=[b\ \ \alpha]^{\mathrm{T}}$, can be updated with equation (8).

The incremental and decremental algorithms for LS-SVM have been presented as above, which make the online training for LS-SVM possible. It is noted that sparse approximation can't be attained through batch training for standard LS-SVM. Therefore the combination of incremental and decremental algorithm is needed to get the sparse approximation of function.

The online algorithm is described as following.

Step 1 Initialize (); // Set C in equation (2) to 100; Set the RBF kernel parameter σ to 1;

//According to equation (5), $Y_1=[0\ \ y_1]^{\mathrm{T}}$; $A_1=[0\ \ 1;\ 1\ \ \Omega_{11}+1/C]^{\mathrm{T}}$; $a_1=A_1^{-1}Y_1$.

Step 2 Input_New_Training_data(x_{N+1}, y_{N+1});

$[b_1,b_2,c]$= Compute_Parameter(x,y); // According to equations (11)-(13).

A_{N+1}^{-1}=Incremental_Add (A_N^{-1},b_1,b_2,c); // According to equation (20).

a_{N+1} = Upadate_Coefficients (A_{N+1}^{-1},Y_{N+1}); // According to equation (9). $a_{N+1}=[b\ \ \alpha]^{\mathrm{T}}$

$N=N+1$; // The number of support vector;

Step 3 If (Length $(a_N)>$Threshold1) // When the number of support vectors exceeds a threshold

{For ($i=1$; $i<=N$; i++)

{If ($\alpha_i<$Thershold 2) //α_i: coefficients corresponding to i th Support Vector

{ A_{N-1}^{-1}= Decremental_Remove (i, A_N^{-1}); // Ac-

cording to equation (21).

a_{N-1} = Upadate_Coefficients (A_{N-1}^{-1},Y_{N-1}); // According to equation (8).

$N = N-1$; }}}

Step 4 Return to **Step 2.**

3 Online LS-SVM for classification

3.1 Standard LS-SVM for classification

Firstly stand LS-SVM for classification is reviewed as follows [2]. Consider the case of two classes, given a training set of N pairs of data points $\{y_k, x_k\}_{k=1}^{N}$, where $x_k \in R^n$ is the input, $y_k \in R$ is the output pattern. The support vector method aims at constructing a classifier of the following:

$$y(x) = \mathrm{sign}[\sum_{k=1}^{N} \alpha_k y_k \Phi(x, x_k) + b] \tag{22}$$

where α_k is support value and b is a constant bias. As for $\Phi(x, x_k)$, RBF kernel function is also selected.

For the case of two classes, assume

$$\begin{bmatrix} 0 & Y^{\mathrm{T}} \\ Y & ZZ^{\mathrm{T}} + \gamma^{-1}\mathrm{I} \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{1} \end{bmatrix} \tag{23}$$

where

$Y = [y_1; \ldots; y_N]^{\mathrm{T}}$, $Z = [\varphi(x_1)^{\mathrm{T}} y_1, \ldots, \varphi(x_N) y_N]$,

$Y = [y_1; \ldots; y_N]^{\mathrm{T}}$, $\vec{1} = [1; \ldots; 1]$,

$e = [e_1; \ldots; e_N]$, $\alpha = [\alpha_1; \ldots; \alpha_N]$.

Mercer's condition is applied to the matrix $\Omega = ZZ^{\mathrm{T}}$ with

$$\Omega = y_k y_l \varphi(x_k)^{\mathrm{T}} \varphi(x_l) = y_k y_l \Phi(x_k, x_l) \tag{24}$$

The support values α_k are proportional to the errors at the data points in the LS-SVM case.

3.2 Online LS-SVM for classification

Assume that a LS-SVM model based on the first N data points have been built, and now the new data point (x_{N+1}, y_{N+1}) is coming in, equation (23) can change into

$$A_N \alpha_N = B_N \Rightarrow \alpha_N = A_N^{-1} B_N \tag{25}$$

The subscript N suggests that the current model is based on the first N data points. For N+1 data points, have

$$\alpha_{N+1} = A_{N+1}^{-1} B_{N+1} \tag{26}$$

where

$$A_{N+1} = \begin{bmatrix} A_N & b_1 \\ b_2 & c \end{bmatrix} \tag{27}$$

$$b_1 = [y_{N+1} \; K_{1,N+1} \; K_{2,N+1} \; \ldots \; K_{N,N+1}]^{\mathrm{T}} \tag{28}$$

$$b_2 = b_1^{\mathrm{T}} \tag{29}$$

$$c = K_{N+1,N+1} + \frac{1}{\gamma} \tag{30}$$

$$K_{i,j} = \Phi(x_i, x_j)\, y_i y_j \tag{31}$$

$$B_{N+1} = \begin{bmatrix} B_N \\ 1 \end{bmatrix} \tag{32}$$

Similar to the function estimation case, the incremental and decremental forms for classification can be obtained (similar to equations (20) and (21)).

4 Experiments

To verify the proposed online training algorithm, experiments with benchmark function estimation and classification data were presented. One is Boston Housing regressing problem and the other is the two-spiral classification problem. The online training algorithm is compared with batch training algorithm. The code is written in Matlab 5.3, under P3 450, Windows 2000, which is available by mailing a request to jhliu99@163.com.

4.1 Benchmark dataset for function estimation

The benchmark dataset "Boston Housing" [10] was chosen. 350 data points are randomly selected for function estimation. The input is 13 dimensional and output is 1 dimensional. The parameter C in equation (2) is set to 10, and the parameter σ in RBF kernel function is set to 1. The online estimation result is the same as batch training, and MSE (Mean Square Error) is 0.6182. But online algorithm is faster than batch training. The estimation result is shown in **figure 1**. The x axes stands for 350 data points, and the y axes stands for the corresponding estimated output. The dashed line is the real data. The dot is the real data point. And the solid line stands for the estimated data. Using online decremental way (equation (21)), a spare approximation can also be obtained. When the threshold is set as 0.6, the number of SV can be reduced to 250 (71.4% of total 350 sample points). And MSE is 6.0150.

4.2 Two-spiral benchmark classification problem

To test the incremental LS-SVM algorithm for classification, the two-spiral benchmark classification problem was used [2]. The training data are shown as **figure 2**, where tow classes are individually indicated

by '□' and '+' (194 points with 97 for each class) in a two dimensional input space. The excellent generalization performance is clearly shown from the decision boundaries (the white line as shown in figure 3). In this case $\sigma = 1$ and $\gamma = 1$. The incremental LS-SVM gets the same results as the standard LS-SVM classification algorithm (100% correct recognition rate). The incremental LS-SVM is shown to be with low computational cost for the avoidance of matrix inverse. LS-SVM gives good results over a wide parameter range of σ and γ values. The shortcoming of LS-SVM is that the sparse feature has been lost compared with standard SVM, which can be remedied by the pruning rule and decremental updating way as the case of function estimation. When the threshold is set as 0.5, the number of SV can be reduced to 173 (89.18% of total 194 data points). And the same 100% correct classification rate can also be obtained.

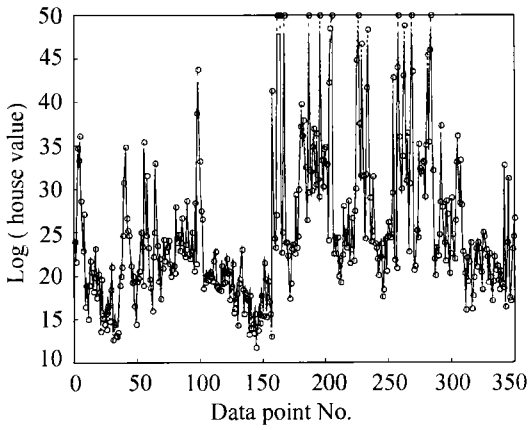

Figure 1 The Boston Housing data.

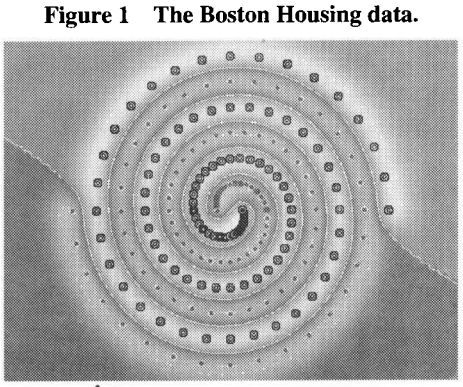

Figure 2 The two-spiral benchmark classification problem.

From above two experiments it is shown that online algorithm can reach the same performance and be faster than batch offline algorithm. But it is more important that online algorithm fits for real application, that is, the data is input one by one.

5 Conclusions

This article proposes an online LS-SVM training algorithm for function estimation and classification. Incremental and decremental means for the update of LS-SVM are inferred. Therefore an online and sparse approximation can be obtained. This algorithm provides a preparation for the application of LS-SVM in online system identification and control problems. Future job is to find the online algorithms for the cases of multi-input-multi output and multi-class.

References

[1] N. Vapnik, *The Nature of Statistical Learning Theory* [M], Springer, 1995.

[2] J.A.K. Suykens and J. Vandewalle, Least squares support vector machine classifiers [J], *Neural Process. Lett.*, 9(1999), No.3, p.293.

[3] J.A.K. Suykens and J. Vandewalle, Sparse approximation using least squares support vector machines, [in] *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS 2000)* [C], Geneva, Switzerland, 2000, p.757.

[4] G. Cauwenberghs and T. Poggio, Incremental and decremental support vector machine learning, [in] *NIPS 2000* [C], Cambridge, MA: MIT Press, 13(2001), p.426.

[5] J.A.K. Suykens, Optimal control by least squares support vector machines [J], *Neural Networks*, 14(2001), No.1, p.23.

[6] S.N. Ahmed, H. Liu, K.K. Sung, Incremental learning with support vector machines, [in] *Workshop on Support Vector Machines* [C], Stockholm, Sweden, 1999, p.924.

[7] T.T. Friess, N. Cristianini, and C. Campbell, The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines, [in] *Machine Learning Proc. 15th Int. Conf. (ICML'98)* [C], CA: Morgan Kaufmann, 1998, p.188.

[8] S. Vijayakumar and S. Wu, A gradient based technique for generating sparse representation in function approximation, [in] *ICONIP'99* [C], Perth, Australia, 1999, p.314.

[9] L. Csato, M. Opper, Sparse representation for gaussian process models, [in] *Adv. Neural Information Processing Sytems (NIPS'2000)* [C], Cambridge, MA: MIT Press, 13(2001), p.444.

[10] C.L. Blake and C.J. Merz, *UCI Repository of Machine Learning Databases Irvine* [D], University of California, CA, 1998.

[11] G.H. Golub and L.C.F. Van, *Matrix Computations* [M], Baltimore MD: Johs Hopkins University Press, 1989.